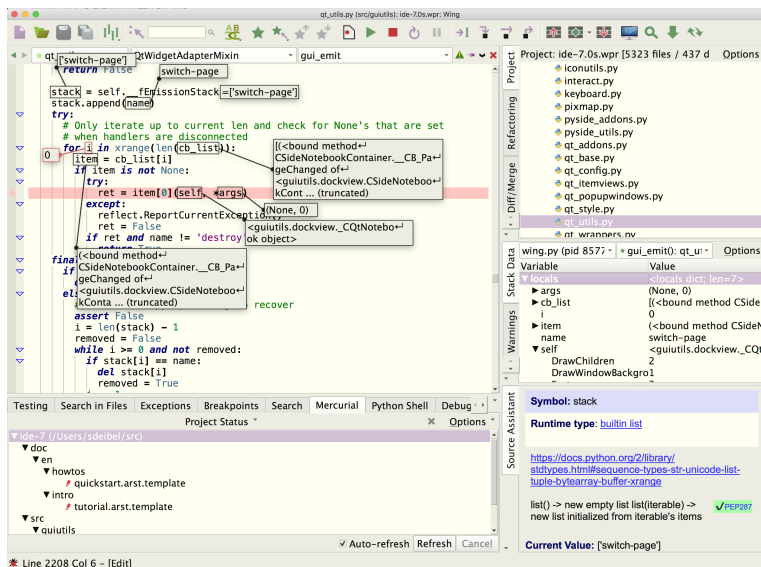# Wing Pro Reference Manual

This manual documents the entire feature set of Wing Pro, which is a Python IDE designed for professional developers.

It covers installation, customization, setting up a project, editing, searching, refactoring, comparing files and directories, navigating source code, using the integrated Python shell, executing operating system commands, unit testing, debugging, version control, source code analysis, PyLint integration, remote development, and extending the IDE with user-defined scripts.

Trouble-shooting information is also included, for installation and usage problems, as well as a complete reference for Wing Pro's preferences, command set, and available key bindings.

If you are looking for a gentler introduction to Wing's feature set, try the **Tutorial** in Wing's **Help** menu. A more concise overview of Wing's features is also available in the Quick Start Guide.

Our How-Tos explain how to use Wing with specific Python frameworks for web and GUI development, 2D and 3D modeling, rendering, and compositing applications, matplotlib, Raspberry Pi, and other Python-based libraries and tools.

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

Contents

Introduction

This chapter describes how to install and start using Wing Pro. See also the Quick Start Guide and Tutorial.

1.1. Product Levels

This manual is for the Wing Pro product level of the Wing family of Python IDEs, which currently includes Wing Pro, Wing Personal, and Wing 101.

Wing Pro is the full-featured Python IDE for professional developers. It is a commercial product for sale on our website, and may be licensed either for Commercial Use or Non-Commercial Use. You may download Wing Pro for free and then use it on a 30-day trial period or with a purchased license.

Wing Personal is a simplified Python IDE that contains a subset of the features found in Wing Pro. It is designed for students, hobbyists, and other users that don't need all the features of Wing Pro. Wing Personal is free to download and use.

Wing 101 is a heavily scaled back IDE that was designed specifically for teaching entry level computer science courses. It omits most of the features of Wing Pro and Personal, and is free to download and use.

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

For a list of the features in each product level, see https://wingware.com/downloads

1.2. Licenses

Wing Pro requires a separate license for each developer working with the product, or a site license configured for the licensed number of users. For the end user license agreement, see the Software License.

To run for more than 10 minutes, Wing Pro 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.

Purchased 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 Pro 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 activating licenses.

1.3. Supported Platforms

Wing 7 is available for Microsoft Windows, Linux, and Mac OS X. Some additional platforms and devices are supported through remote development in Wing Pro only.

Windows

Wing runs on Windows 7, Windows 8, and Windows 10 for Intel processors. Earlier versions of Windows are not supported and will not work.

Older Windows 7 and Windows 8 systems may need to have the Microsoft Visual C++ Redistributable for Visual Studio 2017 installed. It is available from https://support.microsoft.com/en-us/help/2977003/the-latest-supported-visual-c-downloads If this library is needed, Wing will not start and an error dialog indicating that the vcruntime140.dll file cannot be found will appear.

Mac

Wing runs on OS X / macOS 10.12+ as a notarized native application.

Linux

Wing runs on 64-bit Intel Linux versions with glibc version 2.12 or later (such as Ubuntu 12.04+, CentOS 6+, Kali 1.1+, and Fedora 20+). It requires libraries that are typically installed for a graphical desktop environment including libX11, libxcb, libxcb-xkb, libglib, and libexpat. It also requires an X windows server with the X keyboard extension.

Remote Development

Wing Pro's remote development features are fully supported on the same platforms as those listed for the IDE above, with the following additions:

- 32-bit and 64-bit Intel Linux systems that are compatible with the manylinux1 policy as defined in PEP 513 -- glibc version 2.5 or later (such as CentOS and RHEL 5.5+, Ubuntu 9+, and Debian 5.0+)
- ARMv6 and ARMv7 Linux running on Raspberry Pi -- glibc 2.19 and later
- ARMv7 Linux running on the Jolla phone -- glibc 2.19 and later
- Other ARMv6 and ARMv7 Linux systems -- glibc 2.19 and later

Partial support for remote development is available on all other systems that can be accessed via SSH, as described in Manually Installing the Remote Agent.

1.4. Supported Python versions

Wing 7 supports versions 2.6 to 2.7 and 3.3 to 3.8 of Python from python.org, Anaconda, ActivePython, EPD, Stackless Python, cygwin, MacPorts, Fink, Homebrew, and other distributions based on CPython.

OS X and Linux come with Python. On Windows, you will need to install one of the above before using Wing.

Wing can also be used with alternative Python implementations such as PyPy, IronPython, and Jython, but the debugger and Python Shell will not work.

Both 32-bit and 64-bit compilations of Python are supported on Windows. On Linux and OS X only 64-bit Python is supported. However, Linux 32-bit Python can be debugged using Wing Pro's remote development feature.

Wing Pro users can also compile Wing's debugger on other operating systems, and against custom versions of Python (requires NDA).

1.5. Technical Support

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

Wingware Technical Support can also be contacted by email at support at wingware.com, or online at https://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 and debug process internals. Whenever possible, these should be included with email-based bug reports.

1.6. Prerequisites for Installation

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

  • A downloaded copy of Wing
  • 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

Before installing Wing, 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 is referred to as **WINGHOME**. On OS X this is the name of Wing's **.app** folder.

Windows

Install Wing by running the downloaded executable. Wing's files are installed by default in **C:\Program Files (x86)\Wing Pro 7.2**, but this location may be modified during installation. Wing will also create a 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 7.2 of Wing. 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.

The **/verysilent** command line option has the same effect as **/silent** but also prevents display of a progress bar.

Linux

Use the RPM, Debian package, or tar file installer as appropriate for your system type. Installation from packages is at **/usr/lib/wingpro7** or at the selected location when installing from the tar file. Wing will also create a Settings Directory in **~/.wingpro7**, which is used to store preferences and other settings.

Wing Pro is also available in the Snapcraft Store.

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

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

On Windows, launch Wing from the start menu in the lower left.

On OS X, start Wing by double clicking on the app folder.

On Linux, execute **wing7.2** (which is on the **PATH** by default for RPM and Debian installs) or execute **wing** located inside the Wing installation directory. If you installed on Linux from the Snapcraft Store, then the executable is instead named **wing7**.

To run Wing from the command line see Command Line Usage.

1.9. Installing Your License

Wing Pro requires a license in order to run, either a trial license obtained from Wing at startup, or a purchased license. If Wing is running without any license at all it displays the following dialog at startup:
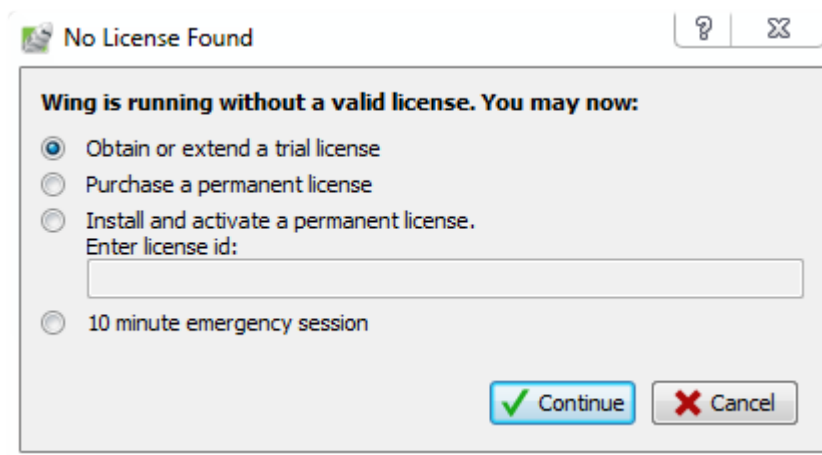
From here, you can choose either to start a trial, visit the Wingware store to purchase a license, activate a purchased license, or start a 10-minute emergency session running without any license.

Starting a Trial

To start a full-featured time-limited trial, select the first option in the dialog above and then press **Continue**. You will be offered the option to connect directly to wingware.com to complete the activation of the trial, or to activate manually, as described below.

Trials are valid for 10 days, with an option to extend twice for up to 30 days total (or more on request by sending the trial license id to sales@wingware.com).

While Wing is running on a trial license, a reminder dialog is shown at startup, with the option to obtain or activate a purchased license.

If you run into problems or need additional evaluation time, please email your trial license id to sales@wingware.com.

Activating a Purchased License

Purchased licenses may either be non-expiring **Perpetual** licenses for a particular major version of Wing or expiring **Annual** licenses. See License Terms for details on the available license types.

All licenses must be activated on each machine where they are used by entering the license into the license dialog at startup. If Wing Pro already has a valid trial license, a different dialog is shown initially, with a button for activating a permanent license. Or, if Wing has already been started and is running on a trial license, **Enter License** in the **Help** menu can be used to enter the permanent license.

In all of these cases, the purchased license id from your license delivery email must be pasted or typed into the activation dialog. Then press **Continue** to select how to activate the license.

The most convenient way to activate a license is to ask Wing Pro to connect directly to **wingware.com** (which it does via https, TCP/IP port 443):
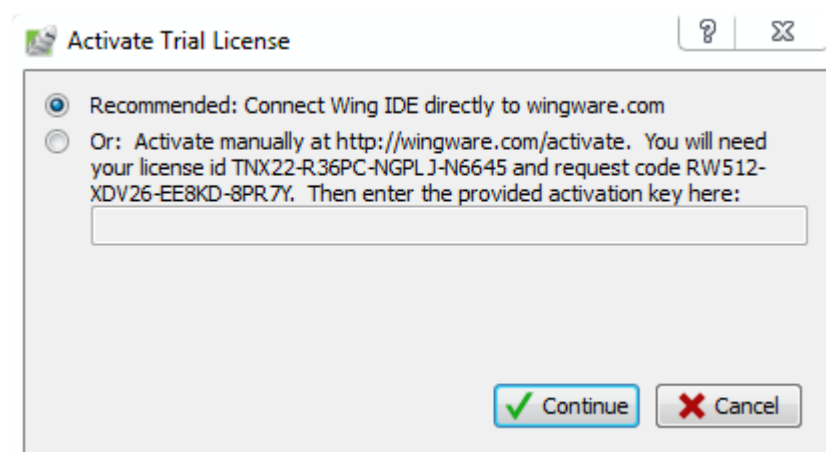
If you're unable to connect directly, you can go to https://wingware.com/activate in your browser or on another device to enter the license id and activation request code obtained from the license

dialog (the second option in the above screenshot). You will be given an activation key which you can then enter into Wing's dialog box to complete the activation. This is exactly the same exchange of information that occurs when Wing Pro 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.

Obtaining Additional Activations

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

Transferring a License

Wing Pro Commercial Use licenses may be transferred from one individual to another, as needed from time to time as employees come and go or change roles. To do this the current user must stop using Wing before the new user starts using the license. The license activation may be removed from the current user's machine as described in the next section.

Deactivating a License

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

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

Activating on Shared Drives

If Wing's Settings Directory (where the license activation is stored) is accessed from several different computers, the license must be reactivated once on each computer. The resulting extra activations will be stored in the settings directory as **license.act1**, **license.act2**, and so forth, and Wing will automatically select the appropriate activation depending on where it is running.

A Vendor File (described below) can be used to automate activation on each additional computer.

Computer Labs

Computer labs consisting of identical hosts mirrored from a master may accept a single activation of a license for all the hosts. This may be used for site licenses and free permanent educational use licenses as follows:

1. Activate the license on the master host
2. Move the **license.act** file from the Settings Directory to the Wing installation directory (on macOS, place it into **Content/Resources** within the application bundle)
3. Mirror the activation to all the clones

Note that Wing's acceptance of a shared activation in this configuration *in no way* relieves you of the responsibility to pay for one license per user.

Vendor Files

To make it easier to reactivate in a case where Wing is on a shared drive, and for computer labs where the above method does not work, you can store your license code in the file **resources/vendor** in your Wing installation, in the following form:

```
license="XXXXX-XXXXX-XXXXX-XXXXX"
```

This file should be named **vendor** (without any extension) and go into the existing **resources** directory in the top level of your Wing installation (or on macOS, within **Content/Resources/resources** inside the application bundle). You will need to create the file if it does not exist.

Once this is done Wing will read this file at startup and try to automatically activate the license, prompting you only if the activation fails. If many activations are expected, you will need to contact sales@wingware.com to obtain additional activations for your license.

1.10. Settings Directory

The first time you run Wing, it will create your **Settings Directory** automatically. This directory is used to store your license, preferences, default project, history, and other files used internally by Wing. It also contains any user-defined snippets, scripts, color palettes, syntax colors, file sets, and shared perspectives.

Wing cannot run without this directory. If it cannot be created, Wing will exit.

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

On Windows the settings directory is called **Wing Pro 7** and is placed within the per-user application data directory. For Windows running on **c:** with an English localization the location is:

```
c:\Users\${username}\AppData\Roaming\Wing Pro 7
```

On Linux and OS X the settings directory is a sub-directory of your home directory:

```
~/.wingpro7
```

Cache Directory

Wing also creates a Cache Directory that contains the source analysis caches, auto-save directory, and a few other things. This directory is also listed in Wing's **About box**, accessed from the **Help** menu.

On Windows, the cache directory is located in the **AppData\Local** area. On Linux, it is **~/.cache/wingpro7** and on OS X, it can be found with the symbolic link **~/.wingpro7/cache-dir-symlink**.

Overriding Settings and Cache Directories

The default location of the settings directory can be changed by passing **--settings=fullpath** on the command line, where **fullpath** is the full path of the directory to use. If the directory does not exist it will be created only if its parent directory exists. Otherwise, Wing falls back to using the default location for the settings directory.

Similarly, the default location of the cache directory can be changed with **--cache=fullpath**.

Shared Settings Directory

Another way to override the default settings directory is to create a directory named **user-settings** inside of the Wing installation directory. When this is present, Wing will use it instead of the default location.

Creating this directory allows settings to move with Wing if your installation is on a portable drive, or to be shared among multiple users that log into the same machine. Permissions on the directory need to allow read and write for all users that will be using Wing.

This is not recommended if multiple users log into the same machine concurrently because settings changed by one user will be overwritten by another user without any notice, and the default project file will be locked if opened by multiple users.

1.11. Upgrading

Upgrades within the same major version number (for example from 7.0.4 to 7.0.5 or 7.1) can usually be made with **Check for Updates** in Wing's **Help** menu. Once you have upgraded, your previous preferences and settings remain in place. After restarting Wing, you should immediately be able to start using the new version.

The current version number is shown at startup and can be found in Wing's **About box**. A list of retained updates is also available here, allowing you to revert back to recent versions.

Upgrading Without an Internet Connection

If the machine where Wing is running does not have an internet connection, you will need to generate an update and https://wingware.com/update using a machine that does have an internet connection, move it to the target machine, and then apply it manually with **Apply Update** in Wing's **Help** menu.

Upgrading to a New Major Release

If you are upgrading across major releases (for example from Wing 6 to Wing 7), then you will need to download and install Wing as described in Installing. This will install the new version along side your older major release of Wing, and they can be used independently.

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.

See also Migrating From Older Versions.

Upgrading Your License

Valid annual licenses for Wing Pro, and perpetual licenses covered by Support+Upgrades can upgrade to any new release for free.

Other licenses must be upgraded before they can be activated in a newer major release. This can be done in the online store.

1.11.1. Migrating From Older Versions

Moving to Wing 7 from earlier versions should be easy. The first time you start Wing 7, it will automatically convert your preferences from the most recent older version of Wing found on your system, and place them into your Wing 7 Settings Directory.

Wing 7 can be installed and used side by side with older major releases (Wing 6 and earlier) and operates completely independently. Projects from earlier versions of Wing will be converted and opened as untitled, and should be saved to a new file name since older versions of Wing cannot open Wing 7 projects.

Compatibility Changes in Wing 7

Wing 7 makes some incompatible changes, including changes in supported Python versions, functional changes, and changes to the API and command set.

Changes in Supported Python Versions

- Dropped support for Python 2.5 and 3.2
- Dropped support for 32-bit Python on OS X

See Supported Python Versions for details.

Important Functional Changes

- Renamed the **Debug Probe** to **Debug Console**
- Automatically restart the **Python Shell** by default when changing projects; you can disable this by unchecking **Auto-restart when Switch Projects** in its **Options** menu
- Change default drag and drop action within the **Project** tool to move files (copying now requires pressing a modifier key)
- Change default for maximum number of non-sticky editors to keep open to 5
- Evaluate in **Python Shell** and **Debug Console** will use the current line when there is no selection even if the **Evaluate Only Whole Lines** option in its **Options** menu is unchecked
- Debugger shows unicode characters in **Stack Data** instead of code points under Python 3
- Wing now sets **sys.argv** in the **Python Shell** when **Evaluate File in Python Shell** is used. The value matches the arguments used when debugging the selected file and persists until another file is evaluated.
- Wing no longer reloads a file when a disk file is removed while it is open in the editor. Instead, the editor is marked as changed with its current contents.

Changes in Scripting API and Command Set

- Rename **CAPIProject.Get/SetMainDebugFile** to **Get/SetMainEntryPoint**
- Rename command **set-current-as-main-debug-file** to **set-current-as-main-entry-point**
- Rename command **set-selected-as-main-debug-file** to **set-selected-as-main-entry-point**
- Rename command **clear-project-main-debug-file** to **clear-project-main-entry-point**
- Deprecate API and commands affected by the above changes
- Change the **bookmarks-remove-all** command to confirm the removal by default
- **GetStackFrame** and **SetStackFrame** API calls are deprecated; use **GetStackIndex** and **SetStackIndex** instead
- Changed command names: **hide-detail** is now **hide-debug-value-detail** and **show-detail** has been replaced with **show-debug-value-as-array** and **show-debug-value-as-text**
- **bin/wingapi.py** has been moved to **src/wingapi.py**
- **bin/wingdb.py**, **src/wing.py**, and **remoteagent.py** have been moved to **bootstrap**
- Dropped the 'sheet' parameter to **CAPIApplication.ShowMessageDialog** in the scripting API

Other Minor Changes

- Disable the old PyLint integration since the new **Code Warnings** tool offers a better option for invoking pylint and indicating its warnings inline in editors
- OS X application name for Wing Pro is now "Wing Pro" and not "WingIDE"
- Exceptions in the **Debugger > Exceptions > Never Report** preference are also ignored when they occur in code run in the **Python Shell** and **Debug Probe**
- Don't display extension module scraping results in the status area
- Remove old and undocumented option to use temp files when running tests
- Increase default for **Source Analysis > Max Cache Size** preference to 2GB
- Trial license numbers are computed differently
- Increased the default for the **Debugger > Line Threshold** preference to 95 to more often include useful data
- Moved size threshold preferences to the new **Debugger > Data Display** page
- Moved introspection preferences to the new **Debugger > Introspection** page
- Don't switch to or from fullscreen mode when starting wing, opening a new project or switching perspectives
- Use https (and not http) for all communication with wingware.com for license activation, update checks, and submitting feedback and bug reports
- Set **TERM=dumb** when **Start Terminal** is used

1.11.2. Fixing a Failed Upgrade

If an upgrade installed via **Check for Updates** causes problems, Wing can be reverted to the earlier installed version from the **About box**. If Wing won't start, use the command line option **--disable-updates** to start Wing without loading the update and then revert to the desired version.

A corrupted installation, resulting in random or bizarre behaviors and crashing, may be fixed by completely uninstalling Wing and manually removing any remaining files before installing again.

If this does not solve the problem, the **ide.log** file in User Settings may contain clues to the problem. Or try moving aside that directory while Wing is not running and then start Wing again. If this solves it, try restoring files from the old settings directory one by one to find the broken file. Files that could cause problems if corrupted include **default.wpr**, **license.act***, **preferences** and **recent***.

If you encounter any problems with an update, please email **support@wingware.com** or submit a bug report from Wing's **Help** menu so we can try to fix the problem for you.

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/wingpro7**. Only 64-bit Linux is supported, although in Wing Pro remote development can be used to develop on a 32-bit host.

Installing Wingware's Public Key

Some systems will complain when you try to install Wing without first installing our public key into your key repository. The key is available here. Copy and paste the key into a file **wingware.pub** and then use the following to import the key.

For RPM systems:

```
sudo rpm --import wingware.pub
```

For Debian systems:

```
sudo apt-key add wingware.pub
```

An alternative is just to bypass the key check with **--nogpg** command line option for **rpm**, **--nogpgcheck** for **yum**, and **--no-debsig** for **dpkg**.

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 wingpro7-7.2.9.0.x86_64.rpm** as root or use your favorite RPM administration tool. Most files for Wing are placed under the **/usr/lib/wingpro7** directory and the **wing7.2** 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.

To install, run **dpkg -i wingpro7_7.2.9.0_amd64.deb** as root or use your favorite package administration tool. Most files for Wing are placed under the **/usr/lib/wingpro7** directory and the **wing7.2** command is placed in the **/usr/bin** directory.

It may be necessary to install some dependencies before the installation will complete, as requested by **dpkg**. The easiest way to do this is **sudo apt-get -f install** -- this installs the missing dependencies and completes the configuration step for Wing's package.

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/wingpro7**. Unpacking this archive with **tar -zxvf wingpro-7.2.9.0-linux-x64.tar.gz** will create a **wingpro-7.2.9.0-linux-x64** directory that contains the **wing-install.py** script.

Running the **wing-install.py** script will prompt for the location to install Wing, and the location in which to place the executable **wing7.2**. These locations default to **/usr/local/lib/wingpro** 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.

Installing from the Snapcraft Store

Wing Pro, Wing Personal, and Wing 101 are also available through the Snapcraft Store. Assuming you have **snap** on your system, you can install Wing as follows:

```
sudo snap install wing7 --classic
```

Notice that you must specify the **--classic** option for **snap** to indicate that you understand Wing uses an unrestricted application confinement model, which is necessary so that it can work with files on your local disk and start sub-processes for debugging, testing, and other IDE operations.

Configuring Wing for High DPI Displays

Wing's UI is implemented with the Qt toolkit, which includes support for high DPI displays, but the support varies depending on the desktop environment in use:

On **KDE**, as of early 2019, Wing should display correctly.

On **Gnome**, as of early 2019, Wing may suggest an interface scale factor based on the size of a character on the primary display.

If Wing is not displaying correctly, the user interface may be scaled manually. To scale icons, windows, and other elements other than fonts, use the **User Interface > Other > Icon and Window Scale Factor** preference. To scale the entire UI, including fonts, use **Presentation Mode** in the common configuration menu, which is accessed from the menu icon in the top right of Wing's window.

The **QT_*** environment variables described at https://doc.qt.io/qt-5/highdpi.html may also be used to scale Wing's display.

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.

XKEYBOARD extension needed

The graphics toolkit that Wing uses, Qt 5, requires the XKEYBOARD extension for the keyboard to work properly. This is an extension to the X11 protocol but has been available for 20+ years. However, there are X11 servers that do not support it including a few used for vnc.

If the keyboard isn't working correctly with Wing, check to see if the X11 server supports XKEYBOARD; sometimes it can be enabled in the server configuration. If it can't be enabled, consider switching to a server that does support the XKEYBOARD extension or try executing **export XKB_DEFAULT_RULES=base** before starting wing. Setting other environment variables is possible according to a bug report at https://bugreports.qt.io/browse/QTBUG-44938

Speeding up the Connection

To improve performance, in most cases you should avoid using 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 disables 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.
- In Wing Pro, another option is not to display Wing remotely but instead to use the remote development feature to access the remote host from Wing running on another machine.

1.12.3. Source Code Installation

Source code is available to licensed users of Wing Pro 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. Backing Up and Sharing Settings

To back up your license, preferences, and other settings, you only need to back up the Settings Directory, which is listed in Wing's **About box**, accessed from the **Help** menu.

The process of restoring Wing or moving to a new machine consists simply of installing Wing again, restoring the above directory, and (in Wing Pro) reactivating your license if necessary.

The only other Wing-specific data that the IDE will write to your disk is in your project files (***.wpr** and ***.wpu** if you are using the **Shared** style of project in Wing Pro; see Project Types for details). We recommend using the default **Shared** project type and checking the ***.wpr** into revision control.

The ***.wpu** contains user-specific and machine-specific data such as environment, path, window position, list of open files, and other GUI state. The file is worth backing up, but usually not hard to recreate if lost.

Wing also writes to a cache directory (also listed in the About box) and your OS-provided temporary directory, but those can be recreated from scratch if lost. The only possible exception to this is **autosave** in the cache directory, which contains unsaved files open in the IDE.

For more information on the location of these directories, see User Settings Directory.

Sharing Settings

Many of the settings found in the Settings Directory can be shared to other machines or with other users of Wing. This includes the following files and directories:

- **filesets** -- shared file sets used for selecting files to search or include in the project.
- **launch** -- shared launch configurations used for defining environment for debugging and executing code.
- **palettes** -- any user-defined color palettes used for configuring the user interface.
- **perspectives** -- shared perspectives which store particular configurations of tools and editors.
- **oscommands** -- shared OS Commands which can be used from any project file.
- **preferences** -- Wing's preferences, as configured in the **Preferences dialog**.
- **remote-hosts** -- shared remote hosts configurations used for remote development.
- **scripts** -- scripts that extend IDE functionality.
- **snippets** -- user-defined code snippets for quick entry of predefined blocks of code.
- **syntax** -- user-defined syntax colors for file types available in the editor.

Follow the links above to find details on the file formats involved. Most are simple textual formats that are easy to generate or modify if necessary. Wing does need to be restarted when replacing these files, and may overwrite changes made while it is running.

1.14. Removing Wing

Windows

On Windows, use the Add/Remove Programs control panel, select **Wing Pro 7** and remove it.

Linux/Unix

To remove an RPM installation on Linux, type **rpm -e wingpro7**.

To remove an Debian package installation on Linux, type **dpkg -r wingpro7**.

To remove a tar archive installation on Linux/Unix, invoke the **wing-uninstall** script in the install directory listed in Wing's **About** box. 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.

User Settings

You may also want to remove the User Settings directory and cache directories if you don't plan to use Wing again on your system.

1.15. Command Line Usage

You can run Wing from the command line as follows:

On Windows, the executable is called **wing.exe** and is located in the **bin** directory in your Wing installation. This is not on the **PATH** by default, but may be added with the Windows Control Panel.

On Linux, when installing from Debian or RPM packages, the executable is named **wing7.2** and when installing with Snap it is named **wing7**. In both these cases, the executable is available on the **PATH**. When installing with the Tar installer, and in all other cases as well, the executable is available as **wing** in the installation directory, which is not on the **PATH** by default.

On OS X, the executable is called **wing** and is located in **Contents/Resources** within the **.app** bundle directory. This is not on the **PATH** by default, but could be added either by adding that directory to **PATH** in **~/.profile** (for example, **PATH="/Applications/WingPro.app/Contents/Resources:${PATH}"; export PATH**) or by placing a symbolic link (for example, by typing **sudo ln -s /Applications/WingPro.app/Contents/Resources/wing wing7.2** in a directory that is already on the **PATH**).

Opening Files and Projects

Once you have established a way to start Wing from the command line, you may specify a list of files to open after the executable name. 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**:

```
wing.exe mysource.py README Makefile myproject.wpr
```

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 line 100 of the README file.

Command Line Options

The following 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 to open files specified on the command line. This option turns off this behavior and forces creation of a new instance of Wing. 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.

--settings=fullpath -- Use the given fullpath instead of the default location for the Settings Directory.

--cache=fullpath -- Use the given fullpath instead of the default location for the cache directory.

--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.

--disable-updates -- Load Wing without applying any updates made since the last installation from an installer package. If you are having problems with an update, the update can be reverted from the **About box**.

--check-install -- Check the installation and all updates to make sure that files have not changed or been removed since they were installed. Combine this with **--disable-updates** to check only the original installation that was made from an installer package.

--get-login-env=yes|no -- *(macOS only)* This option specifies whether Wing will get the inherited environment from a login shell. If this option is not specified, Wing will get the login environment when it is started from the Finder or via open from the command line. The login environment is the environment used when you run a shell or python in a Terminal window.

--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. 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.

Customization

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

2.1. High Level Configuration Options

Wing displays a menu icon in the top right of the window, as part of the toolbar. This provides easy access to some of the most commonly used configuration options.

Display

Dark Mode toggles between light and dark display modes. The default light mode uses color palette **Classic Default** with native UI, while the default dark mode uses color palette **One Dark** applied throughout the UI. Wing will replace these defaults with the most recently used configuration made with the **Color Palette** and **Use Color Palette Throughout the UI** preferences.

Presentation Mode enters a mode where Wing scales the entire user interface, for presentations, meetings, or other situations where temporary scaling is useful. Entering and exiting this mode requires restarting the IDE, but your current project will be reopened.

Show Menubar allows toggling the menu bar on Windows and Linux. When the menu bar is hidden, its menus are included in this configuration menu.

Keyboard

Keyboard Personality selects the overall keyboard emulation mode. Wing can emulate VI/Vim, Emacs, Visual Studio, Eclipse, and several other editors.

Configure Tab Key changes the action of the tab key. See The Tab Key for details.

Custom Key Bindings can be used to enter additional key bindings for any of Wing's documented commands or commands added by extension scripts.

Editor

Configure Auto-Completion can be used to control details of how the editor's auto-completer works. See the Auto-completion for details.

Configure Auto-Editing can be used to control Wing's high-level editing features. See Auto-Editing for details.

Show Line Numbers shows and hides line numbers in the editor.

Show White Space shows and hides visible space, tab, and end-of-line characters in the editor.

Enable Folding controls whether structural folding is enabled in the editor. See Folding for details.

User Items

Additional items can be added to this menu by writing extension scripts that use the **kContextCommonMenu** display context as described in Adding Scripts to the GUI.

2.2. User Interface Options

Wing provides many options for customizing the user interface to your needs, by changing display style and colors, the number and type of windows, layout of tools and editors, type of toolbar, and text font and size.

2.2.1. Display Style and Colors

By default Wing runs with native look and feel for each OS (except on Linux where it cannot use the system-provided UI libraries), and with a classic white background style for the editor.

Editor Color Configuration

The colors used for the user interface are selected with the **User Interface > 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.

Most of the defaults set by the color palette preference can be overridden on a value-by-value basis in preferences. For example, the **Editor > Selection/Caret > 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 an arbitrary color from a color chooser dialog.

In Wing Pro and Wing Personal, the colors used for syntax highlighting code in the editor can be configured separately, as described in Custom Syntax Coloring.

UI Color Configuration

To apply the color palette also to the UI outside of the editor, enable the **User Interface > Use Color Palette Throughout the UI** preference. This is the way to change Wing to using a uniform background color. For a dark background use this option together with **One Dark** or one of the other dark themed palettes.

Add Color Palettes

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

2.2.2. Windowing Policies

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

- **Combined Toolbox and Editor Windows** -- This is the recommended default, with a single window that contains all the editors and two toolbox areas.
- **Separate Toolbox Windows** -- This mode moves all the tools out to a separate window.

The windowing policy is used to describe the initial configuration and behavior 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.

In Wing Pro and Wing Personal, it is possible to create additional windows, and editors and tools can be moved to a new window or among existing windows without changing the windowing policy. This is described below.

2.2.3. User Interface Layout

The layout and behavior of the toolboxes, toolbar, and editor area are configurable. This configuration is stored in your project file and will be restored each time that project is opened. To share a user interface layout between projects, use shared Perspectives.

Configuring Toolboxes

The contents of the toolbox areas can be configured by right-clicking or using the options drop down in the toolbox tab area to add or remove instances of tools.

The number of tool box sections Wing shows by default depends on your monitor size. Each of the toolboxes can be split or joined into any number of sections along the long axis of the toolbox. This is done with **Add Toolbox Split** or **Remove Toolbox Split** in the options drop down menu accessed from the top right of the toolbox or by right-clicking on the toolbox tabs. The tools will automatically be reallocated among the new number of toolbox splits.

Toolbox splits can also be added or removed by dragging tools around by their tabs, within each toolbox, to a different toolbox, or out to a new window. To create a new split, hover over one end of an existing toolbox split until a red split indicator appears.

The size of splits is changed by dragging the divider between them.

The toolboxes as a whole, including all their tools, can be moved to the left or top of the IDE window with **Move to Left** or **Move to Top** in the options dropdown or right click menu. Individual splits or the whole toolbox can also be moved out to a new window from here.

Using the Toolboxes

All the 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.

The **Set Key Binding** item in the options drop down menu can be used to assign a key binding to a particular tool.

Clicking on an already-active toolbox tab will cause Wing to minimize the entire toolbox so that only the tabs remain visible. Clicking again will return the toolbox to its former size. The **F1** and **F2** keys also toggle between these modes.

The command **Maximize Editor Area** in the **Tools** menu (**Shift-F2**) can be used to completely hide and later reshow both tool areas and the toolbar.

Configuring the Toolbar

Wing's toolbar can be configured by right-clicking on it to change the icon size and style, to select which toolbar groups are shown, to turn tooltips on and off, and to customize the icon colors or add custom tools.

The toolbar can also be hidden completely with the **User Interface > Toolbar > Show Toolbar** preference.

Configuring the Editor Area

Editors can be dragged by their tabs to move them to a new split or out to a new window. To create a new split, drag onto the editor surface area and pause above the top, right, left, or bottom portion of the editor until a red split indicator appears. The options drop down menu, accessed from the top right of the editor area or by right-clicking on the editor header, can also be used to add or remove splits.

By default, when multiple splits are shown, all the open files within the window are available within each split, allowing work on any combination of files or different parts of the same file. To open files in only one split, uncheck **Show All Files in All Splits** in the options drop down and then close unwanted duplicates.

In the same menu, **Hide Notebook Tabs** can be used to replace editor tabs with a popup menu that selects among open files. This may be preferable, when many files are open.

Other options in the drop down menu control tab order and sorting of the symbol index menus, among other things.

Creating Additional Windows

In addition to moving existing editors or tools to new windows, Wing Pro and Wing Personal can also create new tool windows (initially with a single tool) and new document windows from the **Window** menu.

2.2.4. Text Font and Size

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

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

2.3. Keyboard Personalities

The default keyboard personality for Wing implements the most common keyboard equivalents found in a many text editors.

> **Note**
>
> Before doing anything else, you may want to set Wing's keyboard personality to emulate another editor, such as VI/Vim, Emacs, Visual Studio, Eclipse, XCode, MATLAB, or Brief. This is done with the **Edit > Keyboard Personality** menu or with the **User Interface > Keyboard > Personality** preference.
>
> See the Key Binding Reference for a list of the key bindings supported for each keyboard personality.

Under the VI/Vim and Emacs personalities, key strokes can be used to control most of the editor's functionality, by interacting with a 'mini-buffer' at the bottom of the IDE window where the current line number and status messages are displayed.

Other preferences that alter keyboard behavior include **User Interface > Keyboard > Tab Key Action** and **Editor > Auto-completion > Completion Keys**.

In Wing Pro and Wing Personal it is also possible to add, alter, or remove individual key bindings in each of these personalities. See the following sub-sections for details.

2.3.1. Key Bindings

The command a key binding invokes may be modified with the **User Interface > Keyboard > Custom Key Bindings** preference. A custom key binding will override any binding found in the current keyboard personality.

To add a binding, click the **Insert** button, press the key binding you wish to use in the **Key** field, and then enter the name of the command to invoke in the **Command** field. To unbind a key that Wing defines by default, leave the **Command** field blank.

To determine what command a key is currently bound to, select **Command by Name** from the **Edit** menu, type **describe-key-briefly** and then press the key binding followed by **Enter**.

Key Bindings

Key bindings consist of one or more key presses, including any regular key and one or more modifier keys (**Shift**, **Ctrl**, **Alt**, and/or **Command**). Multiple modifiers may be pressed at once; **Ctrl-Shift-X** is distinct from **Ctrl-X**.

The **Shift** key is treated as a modifier only for keys where there is a lower case and upper case variant. For example, **Shift-M** is a valid binding for capital **M** while **Shift-9** will result in a different key binding (**Parenleft** on a US keyboard). The dialog for adding key bindings from the Custom Key Bindings preference takes care of this detail.

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

Commands

The command for a key binding may be any of Wing's internal commands, as documented in the Command Reference, or (in Wing Pro and Wing Personal) any user-defined command provided by an extension script.

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

If multiple comma-separated commands are specified, the key binding will execute the first available command in the list. 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.

Some commands take arguments, which can be specified in the binding, for example using **enclose(start="(", end=")")** in the **Command** field will enclose the current selection with **()**. 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 the keyboard personality or overridden by the Custom Key Bindings preference will be shown in menu items that implement the same command. If a command is given more than one key binding, only the last binding found will be displayed, although all the bindings will work from the keyboard.

The Alt Key on OS X

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

This preference should be set according to keyboard type, so that key bindings in Wing do not prevent entering characters composed with the **Option** keys.

For example, on German keyboards, **[**, **]**, and other common symbols are entered with using **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.

2.3.2. Key Maps

Wing ships with several keyboard maps, found at the top level of the installation directory as **keymap.***. These implement the keyboard personalities in the **User Interface > Keyboard > Personality** preference.

In order to develop an entirely new key binding, it is possible to create and select a custom key map with the **User Interface > Advanced > Key Map File** preference.

In a key map file, each key binding is built from the names listed in Key Names. These names are the same as the bindings produced when adding a binding with the **User Interface > Keyboard > Custom Key Bindings** preference, with some additional options. They may include:

1. A single unmodified key, which is specified by its name alone. For example, **'Down'** for the down arrow key.

2. Modified keys, which are specified by hyphenating the key names, for example **'Shift-Down'** if pressing the down arrow while **Shift** is held down. Multiple modifiers may be specified, as in **'Ctrl-Shift-Down'**. However, **Shift** should only be used for keys that have a lower case and upper case variant. For example, **'Shift-5'** is invalid and should be replaced with the key actually produced (**Percent** on US keyboards).

3. Multi-part key bindings can be specified by several bindings separated by a space. For example, to define a key binding that consists of first pressing and releasing **Ctrl-X** and then pushing the **A** key by itself, use **'Ctrl-X A'** as the key binding.

4. The special modifier **Timeout** may be used in multi-part key bindings with otherwise unmodified keys, to indicate a provisional key that is emitted as a regular key if no matching key binding is found within the timeout period. For example, **Timeout-J K** requires typing **jk** in rapid succession. If only **j** is typed, it will be entered after the timeout elapses. If **jp** is typed and there is no binding for **Timeout-J P** then both **j** and **p** will be entered as soon as **p** is pressed. Bindings using **Timeout** only work while the focus is in the editor. Otherwise, they are ignored. The timeout used is configured with the **User Interface > Keyboard > Typing Group Timeout** preference.

5. The **Release** modifier can be used with any single-part key binding to specify that a command should be bound to the release of a key combination. For example, **'Release-Ctrl-X'** invokes a command only when releasing **Ctrl-X**.

6. Special modifiers are defined for VI/Vim mode: **Visual**, **Browse**, **Insert**, and **Replace**. These correspond with the different editor modes, so that the binding will only work in that mode. These modifiers only work if the **User Interface > Keyboard > Keyboard Personality** preference has been set to **VI/Vim**.

The command portion of the key binding may be any of the commands listed in the Command Reference. See Key Bindings and the examples below for details.

Includes

Key maps can include other keymaps. For example, all the default keymaps include a basic map that defines the action of the arrow keys, function keys, and other common functionality:

```
%include keymap.basic
```

The referenced file must be in the same directory as the keymap that contains the include, or a full path.

Examples

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

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

This example undefines a key binding from an earlier definition (usually, from an included key map file):

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

These can be combined to change 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 separated by commas, Wing executes the first command that is available. For example, the following will restart the debug process whether or not one is already 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-console")'
```

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'
```

Here is an example that combines several of the above with the **Release** modifier:

```
'Shift-Space': 'debug-show-value-tips', 'send-keys(keys=" ")'
'Release-Shift-Space': 'debug-hide-value-tips'
```

2.3.3. Key Names

The best way to obtain the names of keys is to enter a new key binding in the **User Interface > Keyboard > Custom Key Bindings** preference. Alternatively, refer to the following enumeration of all keys.

Modifier keys supported for key bindings are:

- **Ctrl** -- Either Control key.

Customization

- **Shift** -- Either Shift key. This modifier is ignored with some key names, as indicated below.
- **Alt** -- Either Alt key. This is not recommended for general use because bindings using it tend to conflict with menu accelerators and operating system or window manager operations. On OS X, the **User Interface > Keyboard > Alt Key** preference is used to configure which keys invoke Alt-key bindings.
- **Command** -- Mac OS Command key. This is intended for use only on OS X.

Unmodified keys such core western alphabet keys are specified as follows:

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

Digits and most punctuation can be specified but the **Shift** modifier will be ignored to accomodate different international keyboards:

0 1 2 3 4 5 6 7 8 9

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

Special keys can also be used with any modifier:

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 ignore the **Shift** modifier include:

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

2.4. Preferences

Wing provides many preferences to control the behavior of the IDE. These are available from the **Preferences** item in the **Edit** menu, or **WingPro** menu on OS X. Preferences are organized by category. Documentation for each preference is displayed when the mouse is hovered over it.

All preferences are also documented in the Preferences Reference.

2.4.1. Preferences File Layers

Wing stores preferences in four layers, as follows:

1. For each preference, Wing defines a default internally.
2. A preferences file that defines defaults for all users may be placed inside the **Install Directory** listed in Wing's **About** box.
3. Each individual user's preferences file is stored in their User Settings Directory.
4. Additional preferences files may be specified on the command line with one or more **--prefs-file** options. For example:

```
wing7.2 --prefs-file /path/to/myprefs
```

Values found in a lower layer override values found higher up.

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 Settings Directory. Wing will never modify the installation-wide preferences file.

If a preference is set to the default value, as determined by the layers further up the list, then Wing removes that 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.

2.4.2. Preferences File Format

While we recommend using the **preferences dialog** 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 the file from which 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.

2.5. Custom 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 the **Editor > Syntax Coloring > Syntax Formatting** preference. 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 a directory called **syntax** within the Settings Directory. This directory must be created if it is not already present.

Your custom syntax coloring configuration files can be modeled on the system-wide defaults, which are stored in **resources/syntax** within the Install Directory listed in Wing's **About** box. Copy only the files you intend to edit. 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. To make this easier to do while working on syntax color configurations, bind a key to the command **restart-wing** or right-click on the toolbar to add an icon for this command.

Overriding Preferences

Note that any non-default syntax coloring preferences will take precedence over syntax files found in the settings directory or system-wide. So if you have previously set syntax colors in preferences, you will need to undo those settings. One way to do this is to edit the **preferences** file in your Settings Directory and remove the value for **edit.syntax-formatting**. You'll need to do this when Wing is not running, or edit a copy of the file in Wing and move it into place while Wing is not running.

Color Palette-Specific Configuration

To override syntax colors only for one particular **User Interface > Color Palette** preference choice, 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.

Print-Only Colors

To override syntax colors for printing only, place the syntax file in a **print** sub-directory of the **syntax** directory. For example, use **syntax/print/python.stx** to specify colors to use in Python files when printing.

Automatic Color Adjustment

If the currently selected color palette in the **User Interface > Color Palette** preference uses a non-white background for the editor, or if the **Editor > Syntax Coloring > Background Color** preference 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 applies both to colors set in preferences and colors in a ***.stx** file. However, automatic color adjustment 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.

Color Names for Python

The syntax color names shown in preferences and the ***.stx** files vary by file type. For Python they are defined as follows:

- **default** -- any text that is not covered by the following
- **commentline** -- a comment starting with a single **#**
- **number** -- any integer, float, binary, octal, or hexadecimal number
- **string** -- a string with double quotes **"like this"**
- **character** -- a string with single quotes **'like this'**
- **word** -- any Python keyword, like **if**, **else**, **for**, **try**, etc.
- **triple** -- a triple quoted string with single quotes **'''like this'''**
- **tripledouble** -- a triple quoted string with double quotes **"""like this"""**
- **classname** -- the name of a class when just after the keyword **class**
- **defname** -- the name of a function or method when just after the keyword **def**
- **operator** -- any operator, like **+**, **-**, **/**, **==**, and so forth
- **identifier** -- any variable including function or class names if not at point of definition
- **commentblock** -- a comment starting with **##**
- **stringeol** -- indicates an unterminated string
- **word2** -- any Python builtin like **open**, **file**, **ord**, **int**, **isinstance**, and so forth
- **decorator** -- a function, method, or class decorator starting with **@**
- **fstring** -- a double-quoted f-string **f"like this"**
- **fcharacter** -- a single-quoted f-string **f'like this'**
- **ftriple** -- a triple quoted f-string with single quotes **f'''like this'''**
- **ftripledouble** -- a triple quoted f-string with double quotes **f"""like this"""**

2.6. Perspectives

Wing Pro and Wing Personal allow you to create and switch between particular arrangements of the IDE's tools. This allows adjusting the user interface 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's name is shown in **[]** brackets in the lower left of Wing's window.

Perspective Manager

Manage Perspectives in the **Tools** menu displays the **Perspective Manager**. This dialog shows the name of each perspective, whether or not the perspective is shared by all projects, 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 **User Interface > Perspectives > Shared Perspective File** preference, instead of in the project file. This makes the shared perspectives available to 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.

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 to set a key binding that 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:

- **User Interface > Perspectives > 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.
- **User Interface > Perspectives > Shared Perspective File** -- This is used to specify where shared perspectives are stored on disk. The default is a file **perspectives** in the 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 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 the Default Toolset

In Wing Pro, the **Tools** menu item **Restore Default Toolset** will restore the tools appropriate for the current perspective. The state that is restored will differ for **edit**, **debug**, and other perspectives.

2.7. File Filters

The **Files > File Types > File Filters** preference allows you to define filters that constrain file selection for the project and searching. When adding or editing a filter, the following information may be entered:

- **Name** -- The display name for 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 File Name** -- The specification in this case is a wildcard that must match the file name. The wildcards supported are ***** to match any string, **?** to many any single character, **[seq]** to match any character in a sequence, and **[!seq]** to match any character not in a sequence. Sequences may be a list of characters or a range specifier such as **a-z** or **0-9**. If the specification contains no wildcard characters, it is treated as a file extension.
- **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. If additional file extensions need to be mapped to a MIME type, use the **Files > File Types > 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** tool.

Project Manager

Wing's **Project** manager provides quick access to the files in your software development project and collects information needed by Wing's debugger, editor, search, version control, and other features.

3.1. Creating a Project

To create a new project, use the **New Project** in the **Project** menu. This prompts to save any changes to your currently open project and then creates a new untitled project. You can then save your project to disk from the **Project** menu.

Project Types

In the **New Project** dialog you will be able to select from among a number of different project types, including:

Empty Python Project creates a new empty project that can be manually configured for any Python project.

Create New Virtualenv runs virtualenv to create a new virtual environment to use with the project. You will be prompted for the virtualenv name, the base Python installation to use, and the parent directory for the new virtualenv. See Using Wing with virtualenv for more information.

Use Existing Virtualenv creates a new empty project using a previously created virtualenv.

Connect to Remote Host via SSH sets up a project that will work with source code that resides on a remote host, container, or virtual machine. See Remote Hosts for more information.

Connect to Vagrant sets up a project that works with source code that is stored in a Vagrant container.

Connect to WSL sets up a project that works with source code that is stored on a Linux installation running under Windows Subsystem for Linux.

A number of other project types are offered as well, for working with specific frameworks and tools documented in How-Tos, such as Django, Jupyter, matplotlib, Raspberry Pi, and so forth.

Each project type creates a new untitled project, performs the necessary basic project configuration, and then displays documentation for the selected project type.

Python Environment

When you create a new project, you should make sure that the **Python Executable**, **Python Path** (if your code modifies **sys.path**) and other values in Project Properties match the Python environment needed by your code. This allows Wing to find and inspect all the modules your code imports, so that editor and debugger features work properly.

Adding Files and Directories

Adding your source files to the project tells Wing which files you are working on, which is important for searching, **Open From Project**, and other features. Usually only the source base you are working on should be added to the project, while Python's standard libraries and other frameworks

and libraries used by your code can be left out of the project and instead found, as needed, through the **Python Path**.

The best way to add code is **Add Existing Directory** in the **Project** menu. This allows you to control which files to include, and whether or not sub-directories are included. The list of files shown in the project updates as files matching the criteria are added and removed on disk.

Individual files can be added with **Add Current File** and **Add Existing File**.

Add New File can be 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 on the **Project** tool.

Removing or Omitting Files and Directories

To remove a specific file or directory, select it and use **Remove Selected Entry** in the **Project** menu or **Remove/Exclude From Project** in the right-click context menu on the **Project** tool.

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.

Saving a New Project

To save a new project, use **Save Project** in the **Project** menu. Once a project file has been saved the first time, it will be re-saved automatically as you work with Wing.

3.2. Moving Projects

Wing's project files reference the files and directories that were added to the project by using relative paths, which it interprets from the location of the project file.

If you need to move a project file to a new location relative to the location of the files and directories it references, without also moving those files and directories, use **Save Project As...** in the **Project** menu. This will update the relative paths so that the project will continue to work from its new location.

3.3. Display Options

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

- **View As Tree** -- This displays the project files in tree form. The tree structure is based on the relative path from the project file to the files and directories added to the project.
- **View As Flattened Tree** -- This view shows files organized according to their location on disk. Each directory is shown at the top level with path names shown as relative paths based on the location of the project file.

The **Options** menu also contains items that control the sorting of files within their directory:

- **Sort by Name** -- Show files in alphabetical order

- **Sort by Mime Type** -- Show files grouped by mime type
- **Sort by Extension** -- Show files grouped by their extension

The **List Files Before Directories** option controls whether files or directories are shown first in the tree view.

3.4. Opening Files

Files can be opened from the **Project** tool by double clicking or middle clicking on the file name, or by right-clicking and using the **Open in Wing** menu item.

Files can be shown within their directory in the native file explorer for the OS by right-clicking on the item in the **Project** tool and selecting **Show in Explorer** (on Windows), **Show in Finder** (on OS X), or **Show Directory** (on Linux).

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 OS X, this opens the file as if double clicked in the OS file browser. On Linux, the preferences **Files > External Display > File Display Commands** and **Files > File Types > Extra File Types** are used to configure how files are opened.

Navigation Options

The **Options** menu in the **Project** tool provide options that control how navigation of the directory tree works:

Follow Selection can be checked to cause Wing to open any file selected in the **Project** tool, regardless of how the selectioni is mode. To avoid clutter, files are visited in transient mode, except if double clicked.

Follow Current Editor causes the current selection on the **Project** tree to track the current editor file.

Once it has the focus, the **Project** tool's tree 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.

3.5. File Operations

The **Project** tool's right-click context menu can be used to execute, debug, and search files, interact with the active revision control system, and define named sets of files to edit and search. The set of operations that will be shown in the context menu are configurable from the menu, with **Configure Menu**.

Executing

You can execute Makefiles, Python source code, and any executable files by right-clicking on the **Project** tool and selecting **Execute Selected**. This executes outside of the debugger with any input/output occurring in the OS Commands tool, where the runtime environment for the execution can be configured.

Debugging

Python files listed in the **Project** tool can be debugged by right-clicking and selecting **Debug Selected**. A particular file can be marked as the main entry point by selecting **Set as Main Entry Point**. Once this is done, the file will be debugged when starting debug from the **Debug** menu or toolbar. For more information on debugging, see Debugger.

Searching

The contents of files and directories may be searched from the **Project** tool by right-clicking on them and selecting **Search in Selected**. This displays the Search in Files tool and sets the **Look in** search scope to the selected item. Focus is placed on the **Search** field so the search string can be entered.

Version Control

In Wing Pro, the **Project** tool will show version control status superimposed on the file and directory icons, and version control operations are available in the right-click context menu. See Integrated Version Control for details.

File Sets

Arbitrary sets of files can be selected on the **Project** tool by clicking, shift-clicking, and ctrl-clicking (or command-clicking on OS X). When this is done, the set of files can be named for later reference by right-clicking on one of the selected items and choosing **Name Selected File Set**. The file set will then appear in the **File > File Sets** menu. See File Sets for details.

3.6. Creating, Renaming, and Deleting Files

The **Project** tool supports creating, renaming, and deleting files and directories on disk. In Wing Pro, changes are tracked also into any active version control system's repository.

Creating Files, Directories, and Packages

The right-click context menu in the **Project** tool contains items for creating new files, directories, and Python packages:

- **Create New File** prompts for a new file's name and opens the file in the editor. The file is not actually created on disk until it is saved from the editor. If a version control system is active, the file will automatically be added to the repository.
- **Create New Directory** prompts for a new directory's name and creates the directory on disk. The directory will automatically be added to the active version control repository, if that version control system tracks empty directories.
- **Create New Package** prompts for a new directory's name, creates the directory on disk, creates a file **__init__.py** inside that directory, and opens it in the editor. If a version control system is active, the file and directory will automatically be added to the repository.

Renaming Files and Directories

Files and Directories can be renamed by clicking on an already-selected item in the **Project** tool and editing the name in place. When **Enter** is pressed to complete the edit, the item will be renamed on

disk. In Wing Pro, if there is an active version control system then the rename will also be tracked in the repository.

Items can also be renamed by right-clicking on them and selecting **Rename**.

Deleting Files and Directories

Files and Directories can be deleted by right-clicking on an item in the **Project** tool and selecting **Move to Trash** (or **Delete** on Windows). The item will be moved to the trash or recycling bin provided by the operating system. In Wing Pro, if there is an active version control system then the removal will also be tracked in the repository.

3.7. Project Properties

Each project has a set of properties that can be accessed and edited from the **Project Properties** item in the **Project** menu. These are used to configure the Python environment that is used when debugging, executing, testing, or inspecting Python code for source code analysis. Correct configuration of project properties is important to auto completion, refactoring, error detection, and other features of the IDE. Project properties are also used to set options for the project, and to enable and configure extensions for Django, matplotlib, and Zope.

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

Environment

The following properties control the Python environment:

Python Executable specifies the Python executable that should be used with the source code in this project. This can be set to **Command Line** to enter the full path to the Python executable, a command that can be found on the **PATH**, or a command line that invokes the Python with all provided arguments. When using a virtualenv or Anaconda environment, this should be set to **Activated Env** to enter the command that activates the environment. In this case, Wing starts Python by running **python** in that environment. The drop down menu to the right of this field lists recently used and automatically found environments. When **Use default** is selected, Wing tries to use the Python started 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 on the **PATH** (on Linux and OS X) or in the registry (on Windows).

An easy way to get the full path to use here is to type **import sys; print(sys.executable)** in the Python you wish to use. This can also be done in the copy of IDLE that is associated with the Python installation.

In Wing Pro, this property can be used to select a Python to run on a remote host. When this is done, the default directory used for other fields in **Project Properties**, and for adding files and directories to the project, will be the base directory defined for the selected remote host. See Remote Hosts for details.

When this property invokes Anaconda Python, Wing will automatically run **conda activate base** before it starts Python. This is needed to avoid failure to import some modules as a result of missing environment. See About Anaconda Environments in the Anaconda How-To for details.

Python Path sets the **PYTHONPATH** environment variable to use whenever Python is launched for debugging, execution, unit testing, or running the Python Shell. When **Use default** is selected, the **PYTHONPATH** environment variable inherited by Wing at startup is used. Otherwise, when **Custom** is selected, the specified **PYTHONPATH** is used instead.

Setting this property is usually only necessary if your code changes **sys.path** at runtime in a way that Wing can't auto-detect or if it depends on **PYTHONPATH** being set from the outside. You should not add Python standard library directories here. Python already knows those and Wing will be able to obtain them by inspecting your selected **Python Executable**.

This property allows displaying the entered Python Path either as a list or as text using the path separator appropriate for the OS. If you need to paste in a path, select **View as Text** first and then right-click to **Paste**. The path is stored internally as a list, so the same configuration can work on multiple OSes.

Both Python and Wing use the selected **PYTHONPATH** to locate modules that are imported with the **import** statement. If necessary directories are missing from the configured path, Python will raise **ImportError** for modules it cannot find, and Wing will fail to provide auto-completion, goto-definition, and other code intelligence on imported modules.

Environment is used to specify values that should be added, modified, or removed from the environment used for debugging and executing code from Wing, including also when running unit tests or version control commands. The values defined here are also used to expand environment variable references used in other properties.

Each entry is in **var=value** form, without any quotes around the value, and must be specified one per line. 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. On OS X the environment inherited by Wing may differ according to whether you launched Wing from the command line or with the Finder.

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

Analyze main entry point for sys.path changes controls whether Wing tries to find changes to **sys.path** in your main entry point. It does nothing if you have not set a main entry point from the **Debug/Execute** tab in **Project Properties**.

Debug/Execute

The following properties control environment for debugged and executed code:

Main Entry Point 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 select a file or named entry point where debug should always start, regardless of which file is current in the editor.

For files, the debug environment defined in **Project Properties** may be overridden by clicking on the file and selecting **Properties**.

Initial Directory sets the initial working directory used for debugging and executing code. When **Use default** is selected, this will be the directory where the debugged or executed 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 source code analysis, and defines the default initial directory used in OS Commands. For these, Wing will use the directory of the **Main Entry Point** in the project as the default initial directory, or the directory of the project file if there is no defined main entry point.

Build Command specifies a command to execute before starting debug. This is useful to make sure that extension modules, Cython modules, and other compiled build targets are rebuilt before each run. The build is configured and run by the OS Commands tool.

Python Options specifies the command line options sent to the Python interpreter while debugging or executing code. 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.

Note that these are *not* the command line arguments to send to your code, but instead options sent to Python itself. To send arguments to your code, select **Debug Environment** from the **Debug** menu. Alternatively, right-click on the Python file, select **Properties**, and then set **Run Arguments** under the **Debug/Execute** tab.

Debug Server Port sets the TCP/IP port on which the debugger listens for externally initiated debug processes. Using this allows multiple instances of Wing using different projects to concurrently listen for and accept externally initiated debug connections. See Advanced Debugging Topics for details.

Automatic Perspectives can be enabled to cause Wing to create and automatically switch between the **Edit** and **Debug** perspectives when debugging is stopped and started. See Perspectives for details.

Options

The following project options are provided:

Project Type (Wing Pro only) selects whether or not the project will be shared among several developers. When shared, the project will be written to two files, ***.wpu** and ***.wpr**. The latter can be checked into revision control and used by other developers or on other machines. See Project Types for details.

Default Encoding sets the text encoding to use for files whose encoding cannot be determined from the contents of the file. This applies to all files edited 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 **Files > 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** tool and elsewhere. It is also used as the default directory in which the Python Shell subprocess is launched and for the starting directory when the **Files > Default Directory Policy** preference is set to **Use Project's Home Directory**.

Preferred Line Ending and **Line Ending Policy** control whether or not the project prefers a particular line ending style, 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, and for new files Wing uses the **Files > New File EOL** preference.

Preferred Indent Style and **Indent Style Policy** control whether or not the project prefers a particular type of indentation style for files, 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, and for new files Wing uses the **Editor > Indentation > Default Indent Style** preference.

Strip Trailing Whitespace controls whether or not to automatically remove whitespace at the ends of lines when saving a file to disk. By default, this falls back to the **Editor > Strip Trailing White Space** preference.

Extensions

These properties are used to control and configure framework-specific extensions:

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 enables Matplotlib-specific event loop support 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 implemented import magic that works differently from Python's **import**. Wing's source analyzer needs this extra clue to find and inspect source files.

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.

Note that this option is no longer needed in Zope 3+.

Testing

In Wing Pro, these options control Wing's integrated unit testing support:

Test File Patterns specifies which files in the project should be shown as unit tests in the Testing tool. Files may be selected by using any combination of wildcards and/or regular expressions that are matched with the full path of all the files in the project.

Default Test Framework defines the testing framework to use for test files that do not specify another framework in their File Properties.

Environment can be used to select environment for running unit tests that differs from the environment configured in **Project Properties**. This also allows setting command line arguments to send to all unit tests.

Use File Properties on an individual test to set different arguments for each test.

Process Model specifies whether Wing should start one test process for each test module, or one for each test package. Different testing frameworks and test suites may require one or the other approach.

Number of Processes sets the number of test processes that Wing will run concurrently. Setting this to a value greater than **1** will allow Wing to take advantage of multiple CPU cores, although it can also cause problems if tests assume they are run in a series.

Run as Package Modules controls whether a test file in a package is run as part of a package or as a stand-alone module. The default depends on the requirements of each unit test framework, and some unit test frameworks ignore this setting.

Save in Project File chooses how much of the test results shown in the **Testing** tool are saved into the project file for redisplay in future sessions. Wing can save all results and output, only results to avoid storing large amounts of output, or no results or output.

VCS

In Wing Pro, this tab can be used to override the **Version Control** preferences:

Version Control selects whether to use preferences settings, override preferences and disable version control entirely, or select another version control configuration. This is used most often when working with a remote host that requires different version control settings than the local host.

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 Wing's Project Properties and File Properties, is used to expand variable references.

Special Environment Variables

The following special variable names are defined by Wing:

- **WING:FILENAME** -- Full path of currently selected file, either in the editor, in the **Project** tool or in other places where files can be selected.
- **WING:FILENAME_DIR** -- Full path of the directory containing the currently selected file. Note that the full path does not include a trailing slash.
- **WING:LINENO** -- Current line number in the currently selected file.
- **WING:SCOPE** -- Dotted name of the current scope in the currently selected file (if Python)
- **WING:PROJECT** Full path of current project, including the ***.wpr** project file name.

- **WING:PROJECT_DIR** -- Full path of the directory containing the current project's ***.wpr**. Note that the full path does not include a trailing slash.
- **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**. Note that the full path does not include a trailing slash.
- **WING:SELECTION** -- The text selected on the current editor, if any.
- **WING:HOSTNAME** -- (Wing Pro only) The remote configuration's **Hostname** for the current project, or the empty string if not a remote project.
- **WING:PYTHON** -- The Python interpreter being used in the current project.

These can be use the same way as other environment variables, for example **${WING:FILENAME}**. Values based on the currently selected file or selection will evaluate to an empty string when there is none.

3.8. File Properties

Properties can be set for individual files to define how Wing reads and inspects the file, how it is displayed in the editor, and to override some of the properties in Project Properties when the file is debugged, executed, or run as a test.

File properties are set by right-clicking on a file in the editor and selecting **Properties**, or by by right-clicking on the **Project** tool and selecting **File Properties**.

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

File Attributes

Properties on this tab affect how Wing reads and inspects the file:

File Type specifies the file type for a given file, overriding the type determined automatically from its file extension and/or content. This property should be used only when the **Files > Files Types > Extra File Types** preference cannot be used to map the file extension to a mime type.

Encoding specifies the text encoding for a file when it cannot be determined from the file's contents. For Python code, it is better to use a PEP 263 **coding** comment, rather than setting this property, and in almost all cases the encoding should be **utf-8**. Similarly, the standard encoding specifier should be used in HTML, XML, and gettext PO files. This is because saving a file without specifying the encoding inside the file may make it impossible for other editors or other Wing projects to read the file. Wing stores the encoding selected by this property in the project, but no mark is written into the source file itself, except in cases where the selected encoding naturally uses a Byte Order Mark (BOM), such as for utf_16_le, utf_16_be, utf_32_le, or utf_32_be.

When this property is altered for an already-open file, Wing will ask whether it should reload the file using the new encoding, save using the new encoding, or to cancel the change. Choose to reload if the file was opened initially with the wrong encoding.

The encoding cannot be altered with this property if the file contains an encoding comment. In that case, the file should edited to change the encoding comment and Wing will save the file using the new encoding.

Line Ending Style specifies which type of line ending to use in the file. When altered, the file will be opened in an editor and converted to the selected style. The change does not take effect until the file is saved to disk.

Indent Style 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 Tool, accessed from the **Convert Indents** button shown next to this property.

Read-only on Disk changes the file's permissions on disk. Permissions are changed only for the owner of the file. On Linux and OS X, group and world permissions are never altered.

Editor

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

Show Whitespace overrides the **Editor > Show White Space** preference on a per-file basis. When enabled, Wing shows spaces and tabs as visible characters in the editor.

Show EOL overrides the **Editor > Show EOL** preference on a per-file basis. When enabled, Wing shows end-of-line (EOL) characters as visible characters in the editor.

Show Indent Guides overrides the **Editor > Indentation > Show Indent Guides** preference on a per-file basis. When enabled, Wing shows vertical indent guides in the editor.

Ignore Indent Errors overrides the **Editor > Indentation > Show Python Indent Warning Dialog** preference on a per-file basis. When checked, Wing will never report indent errors for the current file.

Ignore EOL Errors is used when the project's **Line Ending Policy** is set to **Warn About Conflicts**, in order to disable warnings for this file.

Ensure Ending EOL overrides the **Editor > Ensure File Ends With EOL When Saving** preference on a per-file basis. When enabled, Wing makes sure there is an end-of-line (EOL) at the end of any file it saves to disk.

Debug/Execute

This tab is used to control debug and execution environment for the file:

Environment specifies the environment to use when debugging or executing the file and sets run arguments for it. By default, the environment defined in Project Properties will be used with the specified run arguments. Alternatively, the file may be launched as a named module using **python -m** with the specified run arguments or launched with a different environment defined by a launch configuration.

Show this dialog before each run controls whether this tab is shown in the **Debug Environment** dialog each time this file is debugged. If run arguments often need to be changed, it may be easier to use Named Entry Points to set up different arguments for the same file.

Testing

In Wing Pro, the testing tab contains a subset of the fields described for the Project Properties **Testing** tab.

Test Framework selects which test framework should be used with a test file.

Environment specifies the environment and command line arguments to use when running this file as a test file.

3.9. Sharing Projects

By default Wing Pro stores each project in two similarly named files:

1. The ***.wpr** file contains the sharable data for the project, which can be checked into revision control, used on other machines, and shared with other users.
2. The ***.wpu** file contains user and machine-specific data.

Project Type, under the **Options** tab of **Project Properties** can be set to **Single User (One File)** in order to store both branches of the project into a single ***.wpr** file. This is rarely necessary, except when moving a project to or from Wing Personal, which can only read single-user projects.

Making Project Files More Sharable

In most cases sharing the ***.wpr** file will just work. File paths are stored in a platform-independent way, and relative to the project's location on disk, so they will work on different hosts and OSes.

If revision control conflicts arise among different users of a **.wpr** file, environment variables can be used in any conflicting **Project Properties** to make the shared project file uniform for all users and on all machines. Environment can 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.

Changing Which Properties are Shared

Another way to make a project more sharable is to alter which properties are stored in the shared ***.wpr** file. This is done by editing the **.wpr** file with a text editor and setting the **proj.shared-attribute-names** property. This is a list of properties to add or remove from the default set of shared properties. Each item in the list is an property name preceded by **-** to move a shared property to the non-shared ***.wpu** file, or **+** to move a non-shared property to the shared ***.wpr** file. This specification is applied to the default set of shared properties in order to determine which properties to share in this project.

The following example would move the commands defined in the **OS Commands** tool into the user-specific ***.wpu** file and would share the **Python Executable** and **Python Path** 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 values are valid and uniform 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 properties 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
proj.debug-sub-processes
debug.named-entry-points
proj.launch-config
debug.launch-configs
console.toolbox
```

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

File Format

The **.wpr** and **.wpu** project files use the same textual file format that is used Wing's preferences file. See Preferences File Format for details.

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.

3.10. Launch Configurations

Launch configurations define environment in a way similar to **Project Properties** but in a form that can be applied to an individual file through File Properties, in the creation of named entry points, and for running the **Python Shell**.

They are managed from **Launch Configurations** in the **Project** menu. Use the icons or right click to create, edit, duplicate, or delete items.

Launch configurations contain the following properties. For all of these, environment variable references may be used, as described in Environment Variable Expansion:

Python Tab

Python Executable selects the Python that should be used when running code with this launch configuration. This can be set to **Command Line** to specify the full path to the Python executable, a command that can be found on the **PATH**, or a command line that invokes the Python with all provided arguments. To use a virtualenv or Anaconda environment, this should be set to **Activated Env** to specify the command that activates the environment. In this case, Wing starts Python by running **python** in that environment. The drop down menu to the right of this field lists recently used and automatically found environments. When **Use default** is selected, Wing uses the value from **Project Properties**.

In Wing Pro, this property can be used to run Python on a remote host. When this is done, the launch configuration will only be usable with files that are also stored on that remote host. See Remote Hosts for details.

Python Path sets the **PYTHONPATH** that is used by Python to locate modules that are imported with the **import** statement. By default this uses the path set in **Project Properties**. When **Use default** is selected, the **PYTHONPATH** environment variable inherited by Wing at startup is used instead. Otherwise, when **Custom** is selected, the specified **PYTHONPATH** is used.

Setting this property is usually only necessary if your code changes **sys.path** at runtime in a way that Wing can't auto-detect or if it depends on **PYTHONPATH** being set from the outside. You should not add the Python standard library's **PYTHONPATH** entries here, since Wing will be able to obtain those by inspecting your selected **Python Executable**.

This property allows displaying the entered Python Path either as a list or as text using the path separator appropriate for the OS. If you need to paste in a path, select **View as Text** first and then right-click to **Paste**.

Python Options sets the command line options sent to the Python interpreter while debugging or executing code with this launch configuration. The default uses the setting in **Project Properties**. Using **-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.

Note that these are *not* the command line arguments to send to your code, but instead options sent to Python itself. To send arguments to your code, set **Run Arguments** under the **Environment** tab.

Environment Tab

Run Arguments sets the command line arguments to send to code debugged or executed with this launch configuration. 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 selects the initial working directory to use for processes started with this launch configuration. By default this uses the **Initial Directory** specified in Project Properties. When **Use default** is selected, the directory of the launched file is used instead. When **Custom** is selected, the specified directory is used instead. Use **${WING:PROJECT_DIR}** for the project's directory.

Environment specifies environment variables that should be added, modified, or removed from the environment when using this launch configuration. The drop down menu selects the environment to modify: **Add to inherited environment** modifies the environment inherited when Wing was started, and **Add to project values** modifies the environment from **Project Properties**. When **Use project values** or **Use inherited environment** is chosen, any entered values are ignored and the selected environment is used without changes.

Each entry is in **var=value** form, without any quotes around the value, and must be specified one per line. 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 (optionally, as modified in **Project Properties**) and not modifying an empty environment. On OS X the environment inherited by Wing may differ according to whether you launched Wing from the command line or with the Finder.

Build Command sets a command that will be executed before starting debug with this launch configuration. This is useful to make sure that extension modules, Cython modules, and other compiled build targets are rebuilt before each run. The build is configured and run by the OS Commands tool.

Shared Launch Configurations

By default each launch configuration is stored in the project file. The **Shared** checkbox in the launch configuration dialog causes Wing to store that launch configuration in the Settings Directory instead, in a file named **launch**. Shared launch configurations are 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, launch configurations must be configured so that they will work across projects, machines, and 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 **${VARNAME}** as described in Environment Variable Expansion. The referenced environment variables can be special environment variables defined by Wing, as in **${WING:PROJECT_DIR}**, or user-defined values that are set either system-wide, or in Project Properties. 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.

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 off the project's location is a good approach to maintaining some independence from the configuration and disk layout 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**.

Project Manager

47

Source Code Editor

Wing's source code editor implements a powerful suite of code editing and navigation features for Python, based on both static and dynamic (runtime) source code analysis.

4.1. Opening, Creating, and Closing Files

Opening Files

Files can be opened into the editor from the **File** menu, the toolbar, or by selecting them from the **Project** tool.

Open From Keyboard in the **File** menu provides a keyboard-driven way to navigate the disk in order to open files. The command works in a temporary input area at the bottom of the window. Typing shows a completer with possible directory and file names. **Tab** selects a completion and **Enter** opens the file.

See also File Sets, which makes it easy to name and open sets of files as a group.

Creating Files

Files can be created from the **File** menu, the toolbar, or from the **Project** tool as described in Creating, Renaming, and Deleting Files.

Switching Between Files

The **Window** menu and the tabs at the top of the editor can be used to switch between open files. If **Hide Editor Tabs** is selected in the options drop down at the top right of the editor, then the tabs are replaced with a menu at the top left of the editor, to navigate among the currently open files.

Open From Project in the **File** menu quickly switches to any project file, whether already open or not, using a fragment of the file name.

Closing Files

Open files can be closed from the **File** menu or with the close icon in the top right of the editor area.

In Wing Personal and Wing Pro, the **Open Files** tool in the **Tools** menu makes it easy to close a selected set of files. Right-click on the **Open Files** tool to **Close Selected** or **Close Others**.

4.2. File Status and Read-Only Files

Wing adds status indicators to the titles shown for files in editor tabs, menus, and the status area in the lower left of the window:

> *** indicates that the file has been edited and has unsaved changes.

> **(r/o)** indicates that the file is read-only.

> **(r/p)** (in Wing Pro) indicates that reading the file from a remote host is in progress.

Files that are read-only on disk are opened in a read-only editor. The file can be made writable by right-clicking to select **Properties** and then toggling **Read-Only on Disk** under the **File Attributes**

tab. Permissions are changed only for the owner of the file. On Linux and OS X, group and world permissions are never altered.

4.3. Transient, Sticky, and Locked Editors

In order to prevent accumulation of many briefly-visited open files, Wing can open files in several modes that control how and when they are closed. The mode being used is shown with an icon in the top right of each editor split:

Transient Mode -- Wing opens some files in a non-sticky transient mode that will automatically close the file again when unused and unedited. This is done for files opened when searching, debugging, navigating to a symbol's point of definition or points of use, and when using the **Project** or **Source Browser** tools with the **Follow Selection** checkbox enabled.

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. By default, Wing keeps five transient editors open at a time, and closes the least recently used ones as new transient files are opened.

Sticky Mode -- Files opened from the **File** menu (including **Open from Project** and **Open from Keyboard**), by File Set 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, even if they are not edited.

Locked Mode -- In Wing Pro and Wing Personal, when multiple splits are visible, a third mode is available, where the file is locked into the editor split. In this case, the split is not reused to display any newly opened or visited files, except when no other 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. Transient files that are edited are immediately converted to sticky mode and cannot be set back to transient mode until the changes are saved.

Right-click on the stick pin icon for a menu of files that were recently visited in the associated editor or editor split. Each item in the menu indicates whether it was last visited in transient or sticky mode.

4.4. Editor Context Menu

Right-clicking on the surface of the editor (and in most other places in the IDE's user interface) will display a menu of commonly used context-sensitive commands.

In the editor, this menu is divided into different functional groups for copy/paste, code navigation, evaluating selections, debugging, commenting regions, indentation, accessing File Properties and in Wing Pro also revision control, refactoring, and bookmarking. These can be shown or hidden from the **Configure Menu** item at the bottom of the menu.

In Wing Pro and Wing Personal, user-defined scripts may add items to this menu, as described in **GUI Contexts** in Script Syntax.

4.5. Navigating Source

The editor provides a number of features designed to make it easier to navigate Python code.

Source Index Menus

The menus at the top of the editor provide an index of the classes, methods, and functions in the current file. These can be used to navigate within the top-level scope and within any sub-scopes present at the current position. The menus update as you move the editor caret to other scopes or files.

For an index of all code in the Project, see the Source Browser tool.

Goto Definition

You can visit the point of definition of any Python symbol by right-clicking on it and selecting **Goto Definition** from the editor's context menu.

Alternatively, place the cursor or selection on a symbol and use **Goto Selected Symbol Defn** in the **Source** menu, or its keyboard equivalent.

Control-click (or **Command-click** on OS X) also jumps to the point of definition.

Find Points of Use

In Wing Pro, to view 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, from which you can visit each point of use.

Alt-click (or Option-click on OS X) on a symbol in the editor also displays points of use.

For more information, see Find Uses.

Visit History

The history buttons in the top left of the editor area move forward and backward through recently visited places and editors in a manner similar to the forward and back buttons in a web browser. This is a good way to return from a point of definition or after visiting points of use.

Finding Symbols by Name

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.

Find Symbol in Project in Wing Pro works the same way but searches all files in the project for any symbol matching a fragment.

When a symbol is visited in the editor, Wing highlights it briefly with a callout, as configured from the **Editor > Callouts** preferences group.

4.6. Source Assistant

The **Source Assistant** tool in Wing Personal and Wing Pro displays detailed information about source symbols in the editor, auto-completer, and tools such as the **Project**, **Search in Files**, **Python Shell**, **Debug Console**, and **Source Browser**.

The display includes links to the point of definition of the selected symbol, the symbol's probable type or types, and a link to each type's point of definition. Depending on context and symbol type,

the **Source Assistant** will also display docstrings, call signature, return type, super-classes, overridden methods, and links into Python standard library documentation.

When invoking a function, method, or other callable object, the **Source Assistant** highlights the current argument in the call signature and displays information both for the invoked callable and the current argument or auto-completer selection.

The information displayed in the **Source Assistant** is based on a combination of static and runtime source code analysis. In some code, where static analysis is not successful, running the debugger to a breakpoint allows Wing access to complete and correct code analysis. See Helping Wing Analyze Code for more hints on helping Wing understand your source code.

Docstring Type and Validity

The **Source Assistant** can inspect and display documentation found in docstrings in various ways, either (1) focusing on displaying as much information as possible, even if the docstring cannot be parsed as structured text, or (2) focusing instead on providing parse error information so that docstring formatting can be improved. The display is configured with the **Source Assistant Options** described below.

By default the **Source Assistant** displays a type and validity indicator, showing whether the docstring was successfully parsed or reformatted, and focuses on displaying as much information as cleanly as possible, even if docstrings have formatting problems.

The following indicator messages may appear with each docstring:

â■■ **PEP287** indicates the docstring parsed successfully using PEP 287 reStructuredText Docstring Format and is being rendered accordingly.

â■■ **PEP287** indicates that the docstring does not parse successfully as reStructuredText and is showing inline parse errors.

Rewrapped indicates that the docstring is being shown as plain text but Wing has heuristically rewrapped paragraphs.

Plain Text indicates the docstring is being shown as plain text, exactly as it appears in the source code. PEP 287 style docstrings may fall back to plain text if they cannot be parsed.

Source Assistant Options

There are several options available to select how Wing renders docstrings, and whether or not the display should focus on flagging docstring parse errors. These are accessed by right clicking on the **Source Assistant**:

Use PEP 287 docstrings causes Wing to attempt to render docstrings by treating them as PEP 287 reStructuredText Docstring Format. When disabled, docstrings are always shown as plain text instead.

Show PEP 287 parse errors is disabled by default to focus on showing as much information as possible and not on diagnosing docstring formatting errors. Wing will try to display docstrings as rendered reStructuredText even if they contain parse errors. Wing uses a set of heuristics to gloss

over common errors so the docstring can be rendered, or in more severe cases, falls back to showing the docstring as plain text. When this option is enabled, Wing will shift its focus to reporting PEP 287 parse errors that equal or exceed the **PEP 287 parse error threshold** in severity. Errors are shown in the context of its reStructuredText rendering of the docstring.

PEP 287 parse error threshold sets the error level at or above which Wing will determine that parsing the PEP 287 docstring has failed. When below this level, a best effort will be made to render the docstring without showing any errors. When above this level, Wing either shows the parse errors in the rendered docstring, if **Show PEP 287 parse errors** is enabled and the docstring can be parsed, or falls back to showing the docstring in plain text. The default is to treat warnings, errors, and severe errors as parse errors.

Rewrap plain text docstrings causes Wing to employ a heuristic to rewrap paragraphs in docstrings not being rendered as reStructuredText, in order to make better use of space. This option can be disabled to show the docstring exactly as it appears in the source code.

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

Always show docstrings causes Wing to show all docstrings for all symbols in the **Source Assistant**, even if it is displaying information both for an invocation and current argument type. This is disabled by default, to save space by showing only the docstring for the last symbol.

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

Goto Definition from Documentation

PEP 287 docstrings may include references that link to the point of definition of a named symbol in Python code. This is done using an interpreted text role in the following form:

```
:py:`symbol`
```

The symbol may be a simple name like **MyClass** or a dotted name like **modulename.MyClass** or **modulename.MyClass.SomeMethod**.

When docstrings containing symbol references are rendered in the **Source Assistant**, they will generate a link to the symbol's point of definition. Clicking the link will resolve the point of definition by looking first for the symbol in the same scope as the class, method, or function that the docstring describes, and if that is unsuccessful then by attempting to look up the name on the project's effective Python Path.

To return from the point of definition, use the back arrow in the top left of the editor area.

For example, specifying **:py:`path`** looks for **path** in the scope of the described symbol and then looks for a module named **path** on the Python Path. If **:py:`sys.path.abspath`** is used instead then the process looks for **sys.path.abspath** in the scope of the described symbol, then looks for a module named **sys** with an attribute **path.abspath**, and finally looks for a module named **sys.path**

with an attribute **abspath**. This works even if the referenced module is not imported in the scope of the described object.

In addition to the **:py:** role, Wing follows Sphinx to support the **py:mod**, **py:func**, **py:data**, **py:const**, **py:class**, **py:meth**, **py:attr**, **py:exc**, and **py:obj** interpreted text roles. However, there is no difference in how the point of definition is found for each of these.

Python Standard Library Documentation Links

For symbols in the Python standard library, Wing will attempt to compute a documentation URL whenever possible. Since there is no formal mapping from standard library code to documentation, these URLs are generated heuristically. They are often, but not always correct.

Standard library documentation URLs point to https://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. Folding

Wing's editor supports structural folding for Python, C, C++, Java, Javascript, HTML, JSON, Eiffel, Lisp, Ruby, and a number of other file types. This allows you to visually collapse logical hierarchical sections of code while you are working in other parts of the file.

Editor Fold Margin

When folding is enabled, a fold margin appears to the left of editors that contain a file type that can be folded. Left-clicking on marks in this margin collapses or expands that fold point.

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

Shift while clicking on a fold point expands that point and all its children recursively, so that the maximum level of expansion is increased by one.

Ctrl while clicking on a fold point collapses that point and all its children recursively so that the maximum level of expansion is decreased by one.

Ctrl-Shift while clicking on an expanded fold point collapses all child fold points recursively. When the clicked fold point is later re-expanded, its children will appear collapsed. Ctrl-Shift-click on a collapsed fold point forces re-expansion of all children recursively to maximum depth.

Folding Menus

Right-clicking anywhere on the fold margin displays a context menu with folding operations:

Toggle Fold collapses or expands the fold point.

Collapse More collapses the current fold point one more level.

Expand More expands the current fold point one more level.

Collapse Completely collapses all children recursively.

Expand Completely expands all children recursively to maximum depth.

Collapse All collapses the entire file recursively.

Expand All expands the entire file recursively.

Fold Python Methods collapses all methods in all classes in the file.

Fold Python Classes collapses all classes in the file.

Fold Python Classes and Defs collapses all classes and top-level function definitions in the file.

These are also available in the **Folding** section of the **Source** menu, where each menu item indicates the key equivalents assigned to the operation in your current Keyboard Personality. Items in this menu operate on the first fold point found in the current editor selection or on the current line.

Folding Preferences

You can turn folding on and off and adjust the style and color of fold marks with the **Editor > Folding** preferences.

4.8. Bookmarks

Wing Pro can set bookmarks, in order to navigate code and keep track of notes for unfinished tasks. Bookmarks are defined in a way that allows them to move with the bookmarked line, even if a file is edited outside of Wing.

Setting Bookmarks

Toggle Bookmark in the **Source** menu sets or removes a bookmark at the current line, or clicked line if using the editor's right-click context menu. A default name is used for the bookmark, based on where it is located.

Set Named Bookmark in the **Source** menu displays a dialog to enter a name, category, and notes for the bookmark.

In the VI/Vim keyboard personality, the **m** and **`** keys are supported, in addition to the operations in the **Source** menu, which allow creating bookmarks with names longer than one character. Emacs, Brief, and other keyboard personalities also support bookmarks with their native key bindings.

Bookmarks are shown on the editor with background color change or underline. The style and color of bookmark indicators can be changed with the **Editor > Bookmarks > Bookmark Style** and **Editor > Bookmarks > Bookmark Color** preferences.

Hovering the mouse over a bookmark in the editor shows a tooltip with the bookmark name, notes, and category.

Bookmark Categories

Bookmark categories provide a way to organize and filter which bookmarks are visible in the display. Categories can be added, renamed, and removed with **Edit Categories** in the **Bookmarks** tool, bookmarks toolbar group, and bookmarks popup at the top right of any editor with bookmarks.

Categories marked as **Shared** in the **Edit Categories** dialog are also stored in preferences, so that they will appear in all projects. Categories can also be exported and imported from this dialog.

Traversing Bookmarks

Traverse Bookmarks in the **Source** menu, and the key bindings shown there, can also be used to traverse all bookmarks. To traverse bookmarks in a single file, use the bookmark popup at the top right of the editor.

To visit a bookmark by name, use **Goto Bookmark** in the **Source** menu. This shows a dialog, or in some keyboard personalities an entry area at the bottom of the window, into which a bookmark name can be typed. A list of possible completions will be displayed as you type, and pressing **Tab** will select the current completion.

Filtering Bookmarks

The bookmarks that are visible on the display can be filtered by selecting a current category in the **Bookmarks** tool, the bookmarks toolbar group, or the bookmark popup in top right of the editor.

When **Match Fragment** is chosen, a fragment to match any of the bookmark properties can be entered into the **Bookmarks** tool, which will be displayed if not already visible.

When bookmarks are filtered by category or fragment, the marks shown on the editor, in the bookmarks menus, and in the **Bookmarks** tool will be limited to those that match the filter. This also limits traversal to only matching bookmarks.

Bookmarks Tool

A list of all defined bookmarks is available in the **Bookmarks** tool, from the **Tools** menu. The contents of this tool can be sorted by clicking on the column headers. A bookmark name or category in the list can be edited by clicking on it. Hovering the mouse over a bookmark will display any notes entered for that bookmark.

Right-click for a menu of operations, or select a bookmark and use the toolbar in the top right of the tool. Multi-selection is possible by holding down shift or other modifier keys. Double-clicking or middle-clicking will navigate to the selected bookmark.

When the **Bookmarks** tool has focus, keyboard navigation is possible with the arrow keys. Pressing Enter will navigate to the selected bookmark.

The selected bookmarks or all bookmarks visible in the currently selected category or filter can be exported and imported from the **Bookmarks** tool's **Options** menu and toolbar icons.

Bookmarks Toolbar

Bookmarks can be set, removed, filtered, and traversed from the bookmarks toolbar group, if it is shown. To display the bookmarks toolbar group, right-click on the toolbar and check **Bookmarks** in **Groups Shown**.

Tracking Bookmarks Across External Edits

Bookmarks are stored in the project and refer to a particular position within a selected file. Wing tries to store enough information about the bookmark so it can be moved to the correct location even if a file is edited outside of Wing.

- **For Python files** Wing makes use of the enclosing scope (method, class, or function), as well the contents of the bookmarked line to track the bookmark
- **For all other types of files** bookmarks are defined by file name, line number, and contents of the bookmarked line.

In either case, a bookmark's position may appear to slip if a file changes enough so that Wing cannot find the bookmarked line.

4.9. Syntax Coloring

To make code easier to read, Wing's editor colors a file's syntax according to its MIME type, which is determined by the file's extension or content. For example, any file ending in **.py** will be colored as Python code. Any file whose MIME type cannot be determined will display entirely in black regular text.

If you have a file that is not being recognized automatically, use the **Files > File Types > Extra File Types** preference to add a mapping for the file's extension.

When this is not possible, the file type can be set under the **File Attributes** tab in File Properties.

The colors and text styles used for syntax coloring can be configured as described in Custom Syntax Coloring.

4.10. Selecting Text

Wing can select text by characters, whole lines, or in rectangular blocks, and provides a number of commands for quickly making selections based on the structure of code. This makes it very easy to select code to delete, comment out, or move around.

Multiple selections are also supported, as a way to select and edit multiple parts of code simultaneously.

Selection Mode

When Wing is in selection mode, the current selection is automatically extended as the caret is moved around the editor. The **Selection Mode** sub-menu of the **Edit** menu specifies the type of selection to make as the caret moves:

Characters selects individual characters.

Line selects whole lines.

Block selects a rectangular block.

Cancel exits selecton mode so that moving the caret will not extend the selection. This also unselects the current selection.

The current selection mode is shown in the status area in the lower left of the editor window with one of **[Char Select]**, **[Line Select]**, and **[Block Select]**. When selection mode is canceled, no selection status is displayed.

Selection modes are also supported through the native key bindings emulated by keyboard personalities such as **Emacs** and **VI/Vim**.

If your selected **User Interface > Keyboard > Personality** preference does not support them, then you will need to define key bindings for them using the **User Interface > Keyboard > Custom Key Bindings** preference. The command names are **select-x**, **next-x**, and **previous-x** where **x** is either **statement**, **block**, or **scope**.

Quick Selections

The **Select** sub-menu of the **Edit** menu contains the following commands for quickly selecting sections of code:

Select All selects all of the current file.

Select More adds to the current selection incrementally in logical units. For example, if there is no selection then a word is selected, and if a word is selected then a dotted name or expression will be selected. Eventually, a whole statement is selected, then a whole block, a whole scope, enclosing scopes, and finally the whole file.

Select Less removes from the current selection incrementally in logical units, in opposite order of **Select More**.

Select Statement selects the whole statement at the current position. This may be one line or several lines of code.

Select Next Statement selects the statement after the current one.

Select Previous Statement selects the statement before the current one.

Select Block selects all of the current indented block of code. A block of code is a contiguous range lines delimited by blank lines.

Select Next Block selects the block after the current one.

Select Previous Block selects the block before the current one.

Select Scope selects all of the current indented scope. A scope is a whole **def**, **class** or module.

Select Next Scope selects the scope after the current one.

Select Previous Scope selects the scope before the current one.

4.10.1. Multiple Selections

Wing Pro and Wing Personal support making multiple selections in the editor, which is a powerful way to simultaneously edit two or more parts of your code. For example, all occurrences of a word such as **one** may be selected and then the **o** replaced with **O** to change all of the occurrences to **One** in a single operation.

Selecting Matching Text

The **selection-add-next-occurrence** command (**Ctrl-D**, **Command-D** on the Mac, and **Ctrl->** with the emacs personality) is a convenient way to add selections with matching text. If something is already selected, this command selects the next occurrence of the selected text. If nothing is selected, it will will select the current word.

Whether this search wraps, is case sensitive, or matches only whole words is controlled from the multi-selection toolbar icon or **Edit > Multiple Selections** menu.

To add the next occurrence while dropping the current one, press **Control-Shift-D**, **Command-Shift-D** on the Mac , or **Ctrl-Alt->** with the emacs personality.

Multiple matching selections can also be made quickly within a block, function, method, class, or file by clicking on the multi-selection toolbar icon or using the **Edit > Multiple Selections** menu.

Once multiple selections have been made, any typing, cursor movement, and clipboard commands will act on all selections simultaneously.

Selecting Arbitrary Text

It is also possible to make an arbitrary set of selections, where the selections do not necessarily contain the same text. This is done by holding the **Ctrl** and **Alt** keys down together (or the **Command** and **Option** keys on the Mac) while selecting text with the mouse.

Canceling Multiple Selection

When there are multiple selections, the **Escape** key (or **Control-G** with the emacs personality) will drop all of the extra selections.

Multiple Selections Window

While there are multiple selections in an editor, a floating selections window is shown to list all of the selections, even those that are not visible on screen. An individual selection may be dropped by clicking the X that appears when the the mouse is moved over its entry in the list. Closing the selections window will drop all of the extra selections.

By default, the selections window always appears when there are multiple selections. Use the **Editor > Selection/Caret > Display Selections Popup** preference to set the window to always visible or never visible.

The selections window may also be shown and hidden on a case-by-case basis from the multi-selection toolbar icon or **Edit > Multiple Selections** menu.

4.11. Copy/Paste

There are several ways to cut, copy, and paste text in the editor:

- Use the **Edit** menu items or their key bindings. This stores the copy/cut text in the system clipboard and can be pasted into or copied from other applications.
- Right-click on the editor surface and use the items in the context menu.
- Select a range of text and drag and drop it.
- 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 paste. This behavior may be disabled via the **Editor > Clipboard > Middle Mouse Paste** preference
- Use emulated key bindings for the current keyboard personality, such as **Ctrl-K** for Emacs and named text registers for **VI/Vim**. Note that some of these copy text to a private clipboard and not the system clipboard.

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 the **Editor > Clipboard > On Empty Selection** preference. The default is to use the whole line on copy but not cut.

Indent on Paste

Wing can adjust intentation style, size, and position when pasting lines of text into the editor. See Auto-Indent for details.

4.12. Auto-completion

Wing provides context-appropriate code completion in the editor, Python Shell and Debug Console. Using the auto-completer decreases the amount of typing needed to write code, and reduces the incidence of typos in symbol names.

When enabled with the **Editor > Auto-completion > Auto-show Completer** preference, the auto-completer appears and disappears automatically as you type. Items can be selected by typing until the correct symbol is highlighted, or by using the up and down arrow keys.

To cancel out of the auto-completer, press **Esc** or **Ctrl-G**. The auto-completer also disappears when you exit the source symbol by typing or clicking elsewhere, or if you press key bindings to invoke other commands.

Completion Keys

By default, **Tab** enters the completion it into the editor. Other completion keys can be added with the **Editor > Auto-completion > Completion Keys** preference. For printable keys such as **'.'**, **'('**, **'['**, and **':'** the completion character will be added to the editor after the completed symbol, and any appropriate auto-editing operations will be applied. If **'.'** is used as a completion key, the auto-completer will reappear immediately with the attributes of the completed symbol.

In Wing Pro, it is also possible to configure the auto-completer in Python code to treat any non-symbol key as a completion key. See Turbo Completion Mode for Python for details.

Configuration

In Wing Pro and Wing Personal, the completer can be reconfigured to display only after a specified number of characters, or after a time delay. Completion matching may be case sensitive or insensitive. The completer may also be resized, and can be auto-hidden after a specified timeout.

These and other configuration options are available in the **Auto-completion preferences**.

Code Snippets

In Wing Pro, the auto-completer also contains the names of snippets defined in the **Snippets** tool. Completing a snippet enters it into the editor and collect any snippet arguments inline in the editor, in fields that can be traversed with the **Tab** key. For details, see Snippets.

To prevent Wing from including snippets in the auto-completer, turn off the **Editor > Auto-completion > Include Snippets in Completer** preference.

4.12.1. Turbo Completion Mode for Python

In Wing Pro, when the **Editor > Auto-completion > Python Turbo Mode** preference is enabled, Wing uses a different completion mode for Python files, and in the Python Shell and Debug Console. This mode treats any key that could not be part of a symbol name as a completion key, in a context-appropriate way.

This allows typing until the correct symbol is selected in the completer and then immediately moving on to typing the code that should follow that symbol. For example, typing **+** will place the completion, enter **+** into the editor, apply any relevant auto-editing operations (such as auto-spacing), and show the completer 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.

Although this mode offers a much more efficient way to type Python code, it takes some getting used to before unwanted completions can be avoided. Specifically:

1. If you are trying to type a symbol name before it has been defined, Wing may choose a similarly named symbol from the completer if you do not first cancel out of the completer. As a result, it's usually easier to define symbols first, before writing other code that uses them.
2. Similarly, Wing may fail to recognize some contexts as defining a new symbol. To avoid completing a similarly named symbol, you must first cancel out of the completer.

To make canceling from the completer easier in these cases, **Ctrl**, **Alt**, and **Command** pressed alone are also treated as cancel keys, in addition to **Esc**.

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

4.12.2. Auto-completion Icons

The auto-completer contains two columns of icons that indicate the origin and type of the symbol.

Symbol Origin

⌐ A Python keyword

≙ A Python builtin

✂ A snippet defined in the **Snippets** tool

( ) An argument for the current function or method scope

⚙ A symbol found by introspecting the live runtime state

Symbol Type

🐍 A Python module

C: A class

📦 A Python package

M: A method

F: A function

{} A dictionary

( ) A tuple

[] A list

"" A string

1 An integer

1.0 A float

! An exception

F A Python stack frame

I An object instance of some other type

Symbol Type Annotation

Symbol type icons may be annotated, as in the following examples:

M: An upward pointing arrow indicates that the symbol was inherited from a superclass

F: A leftward pointing arrow indicates that the symbol was imported with **from x import <symbol>** style import statement

4.12.3. How Auto-completion Works

The information shown in Wing's auto-completer comes from several sources: (1) static analysis of Python code, (2) runtime introspection of extension modules, (3) inspection of keywords and builtins in the active Python version, (4) introspection of the live runtime state, when the debugger is active or when working in the **Python Shell** or **Debug Probe**, (5) enumeration of relevant code snippets, and (6) any user-provided interface description files.

See Source Code Analysis for more information on how analysis works and how you can help Wing determine the types of values.

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

4.13. Auto-Editing

Wing Pro's auto-editing operations help to reduce the amount of typing needed to write code by auto-entering text or making corrections as you type. The following operations are available and may be enabled or disabled in the **Auto-editing** preferences group:

Auto-Close Characters enters matching closing quotes, parentheses, brackets, braces, and comment characters. When this is enabled Wing also (1) skips over existing closing characters if they are typed anyway, and (2) 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. For example, quotes are only auto-closed at the end of a line or clause, 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 enters the default arguments for a function or method invocation. **Tab** or **Comma** 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. When this is enabled, the following options are available:

> **Auto-wrap Arguments** automatically re-wraps all the arguments to the configured **Reformatting Wrap Column** after auto-invocation ends.

> **Invoke After Completion** starts auto-invocation automatically after completion of a callable name. If invocation is not wanted, such as when passing a function or method as an argument, you will need to press **Delete** twice.

Apply Quotes to Selection surrounds a non-empty selection with quotes when the quote character is typed. In Python code, selecting a string and pressing a different quote character will convert that string to using the type of quote (either single or double quote). This also works if the caret is just after the closing quote of a string, within the opening or closing triple-quote, or one of the quotes is selected.

Apply Comment Key to Selection will comment or uncomment the currently selected lines, using the style configured in the **Editor > Block Comment Style** preference. This operation only works with single-key comment characters such as **'#'**. Otherwise, use **Toggle Block Comment** in the **Source** menu.

Apply (), [], and {} to Selection surrounds the currently selected text when an open parenthesis, bracket, or brace is typed.

Apply Colon to Selection creates a new block out of a range of selected lines and places the caret for entry of the block type (**if**, **try**, **for**, **with**, etc). When **try** is 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 adds spaces in Python code when typing operators or punctuation and refuses to enter redundant spaces in contexts where spacing is being enforced. For some cases, for example when typing **==**, spacing will be adjusted differently after the first and second keys are pressed. Some associated characters may also be entered, such as **','** after a dict item when **':'** is pressed. The following options are available:

> **Auto-Space After Keywords** auto-enters spaces after Python keyword names. No space is added when the keyword name matches a snippet in the auto-completer, so that the auto-completer is not hidden and snippets can still be used.

> **Enforce PEP 8 Style Spacing** prevents use of auto-spacing that does not adhere to PEP 8 style spacing. See PEP 8 Auto-formatting for other PEP 8 formatting options.

> **Spaces Around = in Argument Lists** overrides PEP 8 conventions and places spaces around equals signs in argument lists.

> **Spaces Elsewhere in Argument Lists** enables auto-spacing also in all other places in argument lists.

> **Spaces After : in Type Annotations** auto-enters spaces after **':'** when it is used in PEP 484 and PEP 526 style type hints.

Manage Blocks on Repeated Colon Key Presses creates new blocks automatically when the colon key is pressed. When the start of a new Python block is typed and **':'** is pressed, this auto-indents the current line, adds **EOL** (end-of-line), and auto-indents the newly created line.

Pressing **':'** a second time will remove the new line and instead indent the following existing line of code under the new block.

Pressing **':'** a third time will instead indent the next contiguous block of lines under the new block, up to any blank line or line that belongs to an enclosing block.

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 ambiguous due to the presence of multiple matching starting blocks. In that case, pressing **':'** repeatedly will toggle the indentation between the possible positions.

The following option is available to control how block management works:

Prefer Block Management Over := causes Wing to immediately manage blocks when **':'** is pressed even in contexts where **:=** could be used in Python 3.8+ code. When this is disabled, pressing **':'** a second time, after an existing colon, triggers block management.

Continue Comment or String on New Line auto-enters comment or string delimiters when **Enter** is pressed within the text of an existing comment or a string. This operation does not apply to triple-quoted strings.

Correct Out-of-Order Typing corrects common typos. For example, **x(.)** is replaced with **x().**, **x(:)** is replaced with **x():**, and Wing will add **'.'** when it is missing in **x().d**.

4.14. Auto-Reformatting

Wing can automatically reformat Python code to be compliant with the PEP 8 Style Guide for Python Code or using the Black or YAPF code formatting tools.

Installing Reformatters

Wing uses its own copy of **autopep8** for PEP 8 style formatting. If you plan to use Black or YAPF formatting then you must first install the formatter into the Python you are using with your code, with **pip** or other package manager. For example:

```
pip install black
pip install yapf
```

Manual Reformatting

The **Source > Reformatting** menu contains items for reformatting the current file or selection for PEP 8, or with Black or YAPF. A single **Undo** will undo the reformatting operation.

Note that reformatting large files may take several minutes, and Wing will lock the file so it cannot be edited during that time. The amount of time spent in reformatting a file is limited to the number of seconds specified with the **Editor > Auto-formatting > Reformatting Timeout** preference. After the timeout is reached, Wing will abort the reformat process and leave the file unchanged. The default timeout is 5 seconds, to avoid leaving an editor locked for a long period of time.

Reformatting PEP8 selections in locally stored files is not time-limited, so very large selections may lock up the IDE until the reformatting operation completes.

Automatic Reformatting

Wing can auto-format edited lines after the caret leaves the line, or whole files as they are saved to disk. This is enabled with the **Auto-Reformat** property under the **Options** tab in **Project Properties**, or with the **Editor > Auto-formatting > Auto-Reformat** preference.

The choices are:

• **Disabled** turns off all automatic reformatting. This is the default.

- **Lines After Edit** reformats individual logical lines (which may span multiple physical lines) after the caret leaves the edited line.
- **Whole Files Before Save** reformats whole files when they are saved to disk. This option is recommended only for users with small files, since reformatting larger files may take substantial amounts of time. The process is aborted and the file is saved without reformatting if the time required to reformat it exceeds the **Editor > Auto-Formatting > Reformatting Timeout** preference.

The formatter to use in auto-formatting can be selected with the **Reformatter** property under the **Options** tab in **Project Properties**, or with the **Editor > Auto-formatting > Reformatter** preference.

The available reformatters are **PEP 8** with **autopep8**, **Black**, and **YAPF**.

Encodings

All the reformatters used by Wing assume **utf-8** encoding if not otherwise specified in a source file with a PEP 263 Python encoding comment. Whole-file reformatting may fail even if Wing correctly guesses the file's encoding, since the coding comment is the only way to communicate a non-default encoding to the reformatters.

4.14.1. PEP 8 Reformatting Options

For PEP 8 reformatting, Wing uses an integrated copy of **autopep8**. There is no need to install anything to use this style of reformatting.

Several options for PEP 8 formatting are provided in the **Editor > Auto-formatting** preferences group:

- **Enforce Line Length** applies PEP 8 style line wrapping during reformatting, using the wrap column configured with the **Editor > Line Wrapping > Reformatting Wrap Column** preference. This is disabled by default, allowing any line length.
- **Reindent All Lines in Files** causes all lines to be reindented with 4-space indentation when PEP 8 reformatting an entire file. When this is disabled, reformatting may still alter indentation within logical lines of code. When reformatting selections, this preference is ignored and only indentation within logical lines may be changed. To convert indentation to other styles or sizes, use the Indentation Manager.
- **Spaces Around = in Argument Lists** overrides PEP 8 by inserting spaces around **=** in argument lists. This is disabled by default.
- **Spaces After #** can be disabled to override PEP 8 insertion of spaces after comment characters. This is enabled by default.
- **Move Imports to Top** can be enabled to enforce PEP 8 requirements to move all imports

to the top of the file. This is disabled by default.

4.14.2. Black Formatting Options

Wing invokes Black with **python -m black** using the Python you have selected in your project configuration. As a result, Black must be installed into your Python with **pip install black**, **conda install black** or other package manager.

Several options for formatting are provided in the **Editor > Auto-formatting** preferences group:

- **Enforce Line Length** during reformatting ensures that lines are wrapped during reformatting, using the wrap column configured with the **Editor > Line Wrapping > Reformatting Wrap Column** preference. This is disabled by default, allowing any line length.
- **Skip String Normalization** disables Black's conversion of string delimiters. This is enabled by default, to prevent Black from corrupting code where the choice of string delimiters is part of the coding standard.

4.14.3. YAPF Formatting Options

Wing invokes YAPF with **python -m yapf** using the Python you have selected in your project configuration. As a result, YAPF must be installed into your Python with **pip install yapf**, **conda install yapf** or other package manager.

None of the options in Wing's auto-formatting preferences are used with YAPF, which should instead be configured using YAPF's configuration system.

4.14.4. Other Reformatters

Reformatters other than autopep8, Black, and YAPF can be integrated with Wing Pro using the OS Commands tool to set up a command line that converts files in place. The command line can use **%s** for the current file name. After conversion on disk, Wing will automatically reload the file into the editor.

OS Commands may be given a key binding, to make them easier to invoke for the current file.

4.15. Code Snippets

Wing Pro's **Snippets** tool makes it easy to write code that contains commonly reused fragments required by coding standards or commenting and documentation conventions.

Snippets may contain arguments to collect when they are placed into the editor and they may be defined for specific file types or even specific contexts within a file, for example within a **class** definition or inside a string.

Snippets are invoked by name from the editor's auto-completer or from the key bindings assigned in the **Snippets** tool. If a snippet contains arguments, they are collected inline in the editor, in a data entry mode.

Although Wing comes with example snippets, in most cases you will want to define your own, to match your coding conventions and preferences.

Snippets Tool

The **Snippets** tool in the **Tools** menu is used to create, edit, delete, and manage snippets.

The option drop down 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, for example **py** for Python. The file extension is converted to a mime type internally so that its snippets can also be used in files that use a different valid file extension for the same mime type. The * file type, which is always present, allows defining snippets that can be applied to all file types.

To add, edit, rename, copy, and remove snippets, use the items in the context menu that appears when you right-click on the **Snippets** tool.

When a snippet is created, it is added to the currently selected file type, and the snippet definition file will be opened into the editor. See Snippet Syntax for details on how to write snippets.

Contexts

Variants of snippets may be defined for different contexts. For example, **def** may omit or include **self** depending on whether it is defining a function or a method in a class. The default set of snippets that ship with Wing illustrate this feature with the **def** and **class** snippet variants for Python.

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

The context for a snippet is changed by right-clicking on the item or clicking on the **Context** column to select a different value.

Key Bindings

The right-click context menu menu on the **Snippets** tool also allows assigning key bindings to snippets. To enter a key binding, just press the desired binding while focus is in the **Key binding** field. Bindings can consist of multiple parts, such as **Ctrl-H B**. Pressing multiple keys will create a key binding sequence, unless too much time elapses between the key presses. To reset the value to blank (no key binding), select all text and press **Backspace** or **Delete**.

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 snippet that matches the current editor context is chosen.

Execution and Data Entry

The easiest way to invoke snippets is from the auto-completer. Alternatively, they can be invoked by their assigned key bindings (if any), by double clicking on the **Snippets** tool, or from the right-click context menu in the **Snippets** tool.

When snippets are invoked, Wing chooses the snippet by name and places the correct variant according to the file type and the context within the current editor. If no context is matched, the snippet for context **all** is used. 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 placing a snippet into the editor, Wing will insert any default arguments, convert indentation and line endings to match the target file, and place the editor into inline data entry mode to collect additional arguments for the snippet.

In data entry mode, Wing moves between the 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.

While in data entry 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 whether changes in indentation also change the context in the editor, for example from **method** to **function**.

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.

Scripting Snippets

Wing's extension API exposes the editor's data entry mode and snippet processing capabilities. This can be used to write Python scripts that generate snippets and paste them into the editor for user data entry. This approach allows for more complex logic than Snippet Syntax supports.

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

4.15.1. Snippet Syntax

Snippets are text files that contain the snippet text along with markup that indicates 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:

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

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

Each part is defined as follows:

varname is the name of the variable.

Since arguments are collected inline, this name is used internally only. If a variable name is used multiple times in a snippet, the value is collected where it first occurs and then inserted multiple times.

@ prepended to the variable name indicates that the value should be wrapped to the column specified with the **Editor > Line Wrapping > Reformat Wrap Column** preference.

! prepended to the variable name indicates that the value should act as a tab stop even if its value is inserted from an earlier field with the same **varname**. This has no effect if the field name is unique.

type is the type of data to collect. This is one of:

string(length) expects a string with given maximum length (or 80 if length is omitted)

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

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

If the **type** field is omitted or empty, string is assumed.

default is the default value to use.

This may be the actual value, or may contain environment variable references in the form **$(envname)** or **${envname}**.

Environment variables can be specified in the environment that Wing inherits when it is launched, in the **Debug** tab of Wing's **Project Properties**, or may be selected from the set of special variables listed in Environment Variable Expansion. Environment variables that are not found expand to the empty string.

When the **default** field is omitted, the field will start 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. The indentation style and size will be determined according to content of the target file or for blank files by using the preferences **Editor > Indentation > Default Indent Style** and **Editor > Indentation > Default Indent Size** .

Similarly, line endings in snippets will be replaced with the appropriate type to match the file into which the snippet placed.

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

The character 'm' to re-indent as a block, so the first line is at the expected indent level for its context in the source.

An integer to re-indent as a block, so the first line is at the given number of indent levels.

The character 'm' followed by '+' or '-' and an integer to 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 caret 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.

4.15.2. Snippets Directory Layout

Snippets are stored in the directory **snippets** inside the Settings Directory. If this directory does not exist the first time the **Snippets** tool is used, it is created and populated by making a copy of the default set of snippets that ship with Wing. Changes and additions made subsequently in the **Snippets** will be stored here, and the directory can be copied to other machines in order to share its snippets with other installations of Wing Pro.

Snippets stored at the top level of this directory can be used with any file in the editor and are shown in the ***** tab of the **Snippets** tool.

Snippets designed for a particular file type are stored in directories named with the most common extension for the file type, for example **py** for Python.

Each of the file type directories may contain snippets that apply to any context in files of that type and sub-directories named **<context>.ctx** for snippets designed for a particular context. **<context>** is replaced with the desired context name.

Snippet file names are simply the name of the snippet with no extension. See Snippet Syntax for details on the snippet file format.

Wing also stores a file named **.config** in the **snippets** directory, which should not be altered or removed, as this may cause the loss of your snippet files.

Snippets Search Path

Additional directories for finding snippets can be specified with the **Editor > Snippets > Snippets Path** preference. Later directories on the path override earlier directories for the same snippet name. New snippets will be created in the last directory on the path.

When one or more directories have been added to the Snippets Path, the **Editor > Snippets > Include Default Snippets** preference can be used to disable displaying the default set of snippets in the Snippets tool.

4.16. Indentation

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

4.16.1. How Indent Style is Determined

Wing can work with files with different indentation styles, including tab-only, space-only, and tab+space indentation.

When an existing file is opened, it is scanned to determine what type of indentation is used in that file. Wing then matches new indentation added during editing to the form already found in the file. If mixed forms of indentation are found, the most common form is used. If no indentation is found, Wing uses the **Preferred Indent Style** set in **Project Properties**, or the **Editor > Indentation > Default Indent Style** and **Editor > Indentation > Default Indent Size** preferences.

Changing Indent Style

To change the indentation style in an existing file, use Indentation in the **Tools** menu.

You can use a different indentation style for non-Python files without first converting existing indent styles by changing the **Indent Style** property in **File Properties**, which is accessed by right-clicking on the editor. Wing will warn that you are entering inconsistent styles of indentation, but the warning can be disabled from the warning dialog or from the **Editor > Indentation > Show Override Warning Dialog** preference.

For Python files, where indentation has syntactic significance, the **Indent Style** cannot be altered without converting the whole file using the Indentation tool, which is accessed from the button next to the **Indent Style** property in **File Properties** or from the **Tools** menu.

Tab Size

The size of the tab character is controlled with the **Editor > Indentation > Default Tab Size** preference. This defines the position of tab stops, counting in multiples of tab size from the start of the line.

This preference is ignored in Python files with mixed tab and space indents, where the file is always shown in the way that the Python interpreter would see it.

Disabling Indent Analysis

Although not recommended, it is possible to disable any attempt to use file contents to determine the style of indentation to use while editing. This is done with the **Editor > Indentation > Use Indent Analysis** preference. When this is disabled, Wing always uses the **Preferred Indent Style** set in **Project Properties**, or the **Editor > Indentation > Default Indent Style** and **Editor > Indentation > Default Indent Size** preferences.

4.16.2. Indent Guides, Policies, and Warnings

In Wing Personal and Wing Pro, the editor can display light vertical lines that make indented code more readable. These are enabled with the **Editor > Indentation > Show Indent Guides** preference, or they can be added to individual files with **Show Indent Guides** under the **Editor** tab of File Properties.

Indent Policies

A preferred indentation style and enforcement policy can be specified with **Preferred Indent Style** and **Indent Style Policy** under the **Options** tab in **Project Properties**.

Indent Warnings

When a file is opened, Wing will indicate a potentially problematic mix of indentation styles found in the file, allowing you to attempt to repair the file. Files can be inspected more closely or repaired with Indentation in the **Tools** menu.

To turn off indentation warnings in Python files, use the **Editor > Indentation > Show Python Indent Warning Dialog** preference.

Wing also indicates suspiciously mismatched indentation in source code by underlining the indent area of the relevant lines in red or yellow. An error or warning message is displayed when the mouse is hovered over the marked area of code.

4.16.3. Auto-Indent

Wing auto-indents code as you create new lines with **Return**, by adding leading white space appropriate for the context. Enough white space is inserted to match the indentation level of the previous line, possibly adding or removing a level of indentation if a block has been started (with **if**, **for**, and others) or ended (with **return**).

Some of the auto-editing operations also result in auto-indentation.

Disabling Auto-Indent

Auto-indent can be disabled with the **Editor > Indentation > Auto-indent** preference. When disabled, the Tab key may be used to insert indentation, depending on its configuration.

Auto-Indent After Paste

Wing also auto-indents code when pasting multiple lines of Python. 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 the **Source > Indentation** menu items. Auto-indent during **Paste** can be disabled with the **Edit > Clipboard > Adjust Indent After Paste** preference.

Wing also converts indentation style during **Paste** to match the target file. This can be disabled with the **Edit > Clipboard > Convert Indent Style On Paste** preference.

4.16.4. The Tab Key

The action of the tab key depends on the **Editor > Keyboard > Personality** preference, the file type being edited, and the position within the file.

To insert a real tab character, press **Ctrl-T**.

Tab Key Action

The behavior of the tab key can be altered with the **User Interface > Keyboard > Tab Key Action** preference, which provides the following options:

Default for Personality selects from the other tab key actions below, according to the current keyboard personality and file type. In all non-Python files, the default is **Move to Next Tab Stop**. In Python files, the defaults are as follows:

> **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

> **Eclipse**: Emulates Eclipse

> **XCode**: Smart Tab

> **MATLAB**: Insert Tab Character

Indent to Match 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 enters indentation so that the caret reaches the next tab stop.

Indent Region increases the indentation of the current line or selected lines by one level.

Insert Tab Character inserts a **Tab** character **chr(9)** into the file.

Smart Tab is equivalent to **Move to Next Tab Stop** in non-Python files, and implements the following behavior in Python files:

> **(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 other valid positions.

> **(2)** When the caret is at the end of a non-empty line and there is no selection, one indent level is inserted. The **User Interface > Keyboard > Smart Tab End of Line Indents** preference alters the type of indentation used in this case, or disables this aspect of the **Smart Tab** feature.

4.16.5. Adjusting Indentation

For cases where the **Tab** key cannot be used to adjust indentation of a line or selected lines, the following commands are available in the **Indentation** portion of the **Source** menu:

> **Indent** and **Outdent** increase or decrease the level of indentation of selected blocks of text. All lines that are included in the current text selection are moved, even if the entire line isn't selected.

> **Indent Lines to Match** adjusts the indentation of the current line or selected lines so that the first line is positioned correctly under preceding code.

4.16.6. Indentation Tool

The **Indentation** tool, accessible from the **Tools** menu, can be used to inspect and change indentation style and size in the current source file.

Indentation Statistics

The top of this tool shows indentation statistics for the current file. This includes the following information:

> **Status** indicates what indentation type is being used for the file.

> **Counts** shows the number of indentations found in the file and how many of those are tab-only, space-only, or tab+space. For example, in a file with 233 space-only indentations and 3 tab-only indentations this would display **236 (3t/233s/0t+s)**. If the file contains indentation errors or warnings, these can be traversed with the right/left arrow buttons shown to the right of the counts. Pressing these buttons jumps to the problem in the editor and hovering the mouse over the indicated code will show details of the error or warning.

> **Tab Size** shows the effective size of the tab character for this file and the origin of this value, which may be preferences or the contents of the file, depending on indentation preferences, file type, and file contents,

> **Indent Size** shows the indent size being used for this file, along with the origin of this value.

Converting Indentation

To convert the indentation type and/or size in the current file, select the type of indentation to use in the **Conversions** section at the bottom of the tool. When converting to **Spaces Only** or **Mixed Tabs & Spaces** the **Indent Size** shown in the **Statistics** area can be changed to select the desired indent size.

The action that will be performed is explained in the area below the conversion type tabs. Press **Convert** to complete the operation in the editor.

Once conversion is complete, the **Indentation** tool updates to display the new status of the file. In Wing Pro, the conversion can be reviewed with **Compare Buffer with Disk** from the **Difference/Merge** toolbar icon.

Save the editor to make the conversion permanent, or use use **Undo** from the **Edit** menu while the editor has focus to discard the conversion.

4.17. Keyboard Macros

Start Macro Record in the **Edit** menu starts the definition of a new keyboard macro. Once macro recording is started, any keystroke or editor command is recorded as part of that macro, until macro recording is stopped with **Stop Macro Record** in the **Edit** menu. Most commands may be included in macros, as well as all character insertions and deletions.

Using the operations under **Mini-search** in the **Edit** menu combined with cursor movements and edits allows for the creation of macros that can be applied repeatedly to code with **Execute Macro** in the **Edit** menu.

Keyboard macros are also supported by the native bindings emulated by keyboard personalities like **Emacs**, **VI/Vim**, **Visual Studio**, and **Brief**.

4.18. Auto-Reloading Changed Files

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

The 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 with the the **Files > Reloading > Reload when Unchanged** and **Files > Reloading > Reload when Changed** preferences

By default, reloading will close files that disappeared on disk. This is the recommended behavior when a revision control system is in use, because updates or branch switches that occur while Wing is running may remove open files. However, this behavior can be overridden with the **Files > Reloading > Reloading Deleted Disk Files** preference. Using **Prompt for Action** instead reduces the chances of entirely losing a file if the file is accidentally deleted on disk.

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 5 seconds. This frequency can be changed with the **Files > Reloading > External Check Freq** preference.

Before reloading a file with changed modification time, Wing checks the contents of the file and avoids reloading it into the editor when its contents remains unchanged. This check is skipped for files larger than 5MB and it may be disabled entirely with the **Files > Reloading > Check Hash Before Reloading** preference. This may be needed when working with a slow network disk, where the process of checking the contents of files slows down Wing more than reloading unchanged files.

4.19. Auto-Save

Wing auto-saves files to disk every few seconds so they can be restored if the IDE is killed from the outside or crashes. The auto-save files are placed in a subdirectory of your Cache Directory.

Wing checks this directory at startup and will offer to restore any unsaved changes. The files you select to restore will be opened into Wing as edited files.

In Wing Pro you can compare the restored files to disk using **Compare Buffer with Disk** item in the **Difference/Merge** toolbar item or **Source > Difference/Merge** menu area.

To keep the restored unsaved changes, save the file to disk.

To discard the unsaved changes, use **Revert to Disk** in the **File** menu.

4.20. File Sets

File sets are named groups of files that can be opened together or searched from the **Search in Files** tool in the **Tools** menu. File sets are created in several ways:

- Open the desired files and use **Name Set of Open Files** in the **Files > File Sets** menu.
- Select the desired files in the **Project** or **Open Files** tools in the **Tools** menu. Then right-click on the tool and select **Name Selected File Set**, or use **Name Set of Selected Files** in the **Files > File Sets** menu.
- Search in the **Search in Files** tool in the **Tools** menu and when the search is complete use **Name Result File Set** in the tool's **Options** menu.

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.

Managing File Sets

To view or edit the defined file sets, use **Manage File Sets** in the **File > File Sets** menu. Right-click to access all 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 binding that will open all the files in the file set into the editor. This is done in the **Manage File Sets** dialog from the **Files > File Sets** menu, by selecting the file set, right clicking, and choosing **Set Key Binding**.

To enter a key binding, just press the desired binding while focus is in the **Key binding** field. Bindings can consist of multiple parts, such as **Ctrl-H B**. Pressing multiple keys will create a key binding sequence, unless too much time elapses between the key presses. To reset the value to blank (no key binding), select all text and press **Backspace** or **Delete**.

Shared File Sets

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

4.21. Other Editor Features

Show Line Numbers

To show and hide line numbers on the editor, use the **Show Line Numbers** and **Hide Line Numbers** items in the **Edit** menu.

Block Commenting

Use **Toggle Block Comment** in the **Source** menu to comment out the selected lines of code in the current editor. Selecting the command a second time will return the lines to their former uncommented state.

For Python files, the type of commenting used with this feature is configured with the **Editor > Block Comment Style** preference. Indented block commenting styles tend to work better when editing code around commented out lines.

Line Editing

The **Line Editing** sub-menu of the **Source** menu provides some commands for quickly operating on lines of code:

New Line Above creates a new blank line above the current line, auto-indents, and places the caret at the start of the new line.

New Line Below works the same way but places the new line below the current line.

Duplicate Lines Above duplicates the current lines or lines above the current selection. The caret or selection is left unchanged.

Duplicate Lines Below works the same way but the lines are placed below the current selection.

Move Lines Up moves the current line or lines upward one line.

Move Lines Down moves the current line or lines downward one line.

Delete Lines deletes all of the current line or lines, even if the selection does not span whole lines.

Swap Lines swaps the current line, or the line at the start of the selection, and the next line.

Enclose

The **Enclose** sub-menu of the **Source** menu provides commands to enclose the current selection with **()**, **[]**, **{}**, **"**, **""**, or **<>**. If there is no selection, the operation is applied to the text between the caret and the end of the line.

Changing Case

The **Change Case** sub-menu of the **Source** menu provides commands to convert the case of the current selection to **UPPER CASE**, **lower case**, or **Title Case**.

Toggle Symbol Case in the same menu converts the current symbol between **my_symbol_name**, **mySymbolName**, and **MySymbolName** form. To convert all occurrences of a symbol, use the items in the **Refactor** menu instead.

Zooming In and Out

The editor font size can be increased and decreased temporarily from the **Zoom** sub-menu of the **Edit** menu.

If the **Editor > Enable Font Size Zooming** preference is enabled, zooming the editor can also be accomplished by holding down the **Ctrl** key (or **Command** on OS X) while operating the mouse wheel or track pad.

Reset Zoom in the **Edit > Zoom** menu returns the font size to the original.

Brace Matching

Wing highlights 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 **Match Braces** in the **Source** menu.

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

Zip and Egg Support

Source files that are stored in **.zip** or **.egg** files may be loaded into the editor as read-only files, during stepping in the debugger, for goto-definition, and as otherwise needed. However Wing is unable to write to a file within a **.zip** or **.egg** file.

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.

Search and Replace

Wing provides a number of tools for search and replace in your source code, for quick one-off searches from the toolbar, keyboard-driven search and replace, and single and multi-file search and replace.

5.1. Toolbar Quick Search

The search area of the toolbar can be used for simple searching in the current file. This scrolls as you type to display the next match found after the current caret position or selection. Press **Enter** to search for each subsequent match. The search wraps when it reaches the end of the file.

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

When a match is visited in the editor, Wing highlights it briefly with a callout, as configured from the **Editor > Callouts** preferences group.

5.2. Keyboard-Driven Search and Replace

Keyboard-driven search and replace are available in the **Mini-search** sub-menu of the **Edit** menu. These are normally initiated with the key bindings shown in the menu and can be controlled entirely from the keyboard. All interaction with the mini-search manager occurs in the status area at the bottom of the IDE window.

For keyboard personalities like **Emacs** and **VI/Vim**, Wing will emulate the appropriate bindings for that editor.

Forward and **Backward** display an initially blank search area at the bottom of the IDE window to search in the current source editor, starting from the current position. The search takes place as you type and can be aborted with **Esc** or **Ctrl-G**, which restore the original selection and scroll position.

Searching is case-insensitive unless you enter a capital letter as part of your search string.

To move through matches in the editor, press the key binding for the command repeatedly. The search direction can be changed by using the key binding for the other search direction.

When search is first initiated, pressing the key binding a second time enters the most recent search string. When the top or bottom of the file is reached, press the key binding again to cause the search to wrap.

While the mini-search area is visible, **Ctrl-W** adds the current word in the editor to the search string. Pressing **Ctrl-W** repeatedly adds subsequent words.

Selection Forward and **Selection Backward** start mini-search with the current selection in the editor.

Regex Forward and **Regex Backward** start mini-search using the search string as a regular expression.

Query/Replace and **Query/Replace Regex** prompt for **Search** and **Replace** strings in the status area at the bottom of the IDE window. **Tab** moves between the fields and **Enter** starts the search

from the current caret position in the editor. For each match, press **y** to replace or **n** to move on to the next match without replacing. The interaction can be canceled 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 at the end of the file, without wrapping.

Replace String and **Replace Regex** work like **Query/Replace** but immediately replace all matches without prompting.

When a match is visited in the editor, Wing highlights it briefly with a callout, as configured from the **Editor > Callouts** preferences group.

5.3. Search Tool

The **Search** tool in the **Tools** menu can be used to search and replace within the current editor.

Searches may be initiated from the **Search and Replace** sub-menu of the **Edit** menu, using **Search**, **Replace**, **Search for Selection Forward**, and **Search for Selection Backward**. The **Replace** field will be hidden unless a replace operation was started. It can also be shown from the **Options** menu at the top right of the tool.

The popups to the right of the **Search** and **Replace** fields, contain a history of previously used strings. Right-click on the fields to insert special characters.

To search only part of a file, select the desired range in the editor and check **In Selection**.

When a match is visited in the editor, Wing highlights it briefly with a callout, as configured from the **Editor > Callouts** preferences group.

Search Type

The type of search is selected from the **Options** menu:

Text Search chooses plain text search, without wildcard or regex matching.

Wildcard Search uses wildcard style searching. See Wildcard Search Syntax for details.

Regex Search uses regular expression style searching. See Python's Regular Expression Syntax documentation for details. When **Regex Search** is selected, a popup menu **Regex Flags** appears to the left of the **Options** menu. These are the same flags passed to Python's **re.compile()**. For regex searching, the replace string can reference regex match groups with **\1**, **\2**, etc, as in the Python **re.sub()** call.

For each of these, the checkboxes in the tool provide some additional options:

Case Sensitive shows only exact matches of upper and lower case letters in the search string.

Whole Words requires that matches are surrounded by white space (spaces, tabs, or line ends) or punctuation other than _ (underscores).

Search Options

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

Show Replace controls whether the **Replace** field is visible in the tool.

Wrap Search allows wrapping when the search reaches the top or bottom of a file.

Incremental immediately starts or restarts searching as you type or alter search options. When unchecked, use the **Previous** and **Next** search buttons to initiate searching.

Find After Replace automatically finds the next search match after each replace operation.

Special Characters

The right-click context menu on the **Search** and **Replace** fields provide some options for search and replace strings to include special characters:

Insert Newline inserts new line (**\r\n** on Windows and **\n** on other OSes)

Insert Line Feed inserts a line feed character (**\n**)

Insert Carriage Return inserts a carriage return character (**\r**)

Insert Tab inserts a tab character (**\t**)

Interpret Backslash Characters toggles whether special characters like **\n**, **\r**, **\t** and others are interpreted as a backslash followed by a letter or as the character that they represent (line feed, carriage return, tab, etc). The supported characters are all those that Python supports in its representation of strings.

5.4. Search in Files Tool

The **Search in Files** tool in the **Tools** menu searches within sets of files and displays a list of all matches found.

The files to search are selected with **Look in** and **Filter**. **Look in** specifies the set of files to search, which may be the current editor, a single selected file, all open files, all project files, a named File Set, a selected directory on disk, or all of Wing's documentation. **Filter** can be used to select a subset of the files specified by **Look in**, using a File Filter or by typing a wild card expression containing ***** and/or **?**. For example, **Look in** set to **Project Files** and **Filter** set to **Python Files** will restrict searching to only Python files that appear in the **Project** tool. **Look in** set to ***.mako** would search only file sending in **.mako**. If the **Filter** is neither a valid **File Filter** name nor a valid wild card expression then all the files selected by **Look in** are searched.

Searches may be initiated using **Search in Files** in the **Search and Replace** sub-menu of the **Edit** menu. The **Replace** field will be hidden unless **Replace in Files** was used. It can also be shown from the **Options** menu at the top right of the tool.

The popups to the right of the **Search** and **Replace** fields contain a history of previously used strings. Right-click on the fields to insert special characters.

Once a search is started, matches can be selected from the result list and shown in the editor or documentation viewer, even before the entire search completes. The result list is updated automatically as files are edited, added, or removed, in order to include any new matches or remove any old ones.

When a match is visited in the editor, Wing highlights it briefly with a callout, as configured from the **Editor > Callouts** preferences group.

Search Type

The type of search is selected from the **Options** menu in the top right of the tool:

Text Search chooses plain text search, without wildcard or regex matching.

Wildcard Search uses wildcard style searching. See Wildcard Search Syntax for details.

Regex Search uses regular expression style searching. See Python's Regular Expression Syntax documentation for details. When **Regex Search** is selected, a popup menu **Regex Flags** appears to the left of the **Options** menu. These are the same flags passed to Python's **re.compile()**. For regex searching, the replace string can reference regex match groups with **\1**, **\2**, etc, as in the Python **re.sub()** call.

Case Sensitive shows only exact matches of upper and lower case letters in the search string.

Whole Words requires that matches are surrounded by white space (spaces, tabs, or line ends) or punctuation other than _ (underscores).

Options

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

Show Replace controls whether the **Replace** field is visible in the tool.

Show Search Type in Tool moves the selection of search type out of the **Options** menu and to the surface of the tool.

Find After Replace automatically finds the next search match after each replace operation.

Replace Operates On Disk replaces text in un-opened files directly on disk rather than in an editor. This should be used with caution since changes cannot be undone except by reverting using a version control system or restoring from a backup.

Recursive Directory Search also searches all sub-directories when searching a directory on disk.

Omit Binary Files omits any file that appears to contain binary data.

Auto-restart Searches restarts searching immediately if it is interrupted because a search option or the set of files being searched has changed.

Open First Match automatically opens the first batch search match found when searching starts.

Show Line Numbers includes line numbers in the result area.

Result File Name selects the format of the file names shown in the batch result area.

Copy Result to Clipboard places a copy of all the search results on the clipboard.

Name Result File Set creates a File Set containing all the files listed in the current result list.

Special Characters

The right-click context menu on the **Search** and **Replace** fields provide some options for search and replace strings to include special characters:

Insert Newline inserts new line (**\r\n** on Windows and **\n** on other OSes)

Insert Line Feed inserts a line feed character (**\n**)

Insert Carriage Return inserts a carriage return character (**\r**)

Insert Tab inserts a tab character (**\t**)

Interpret Backslash Characters toggles whether special characters like **\n**, **\r**, **\t** and others are interpreted as a backslash followed by a letter or as the character that they represent (line feed, carriage return, tab, etc). The supported characters are all those that Python supports in its representation of strings.

5.5. Find Points of Use

Wing Pro can find all points of use of a symbol 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 the editor's right-click context menu, or **Alt-click** on a symbol.

Find Points of Use searches all files in your project. To limit your search to only the current file, use **Find Points of Use in Current File** instead.

The results are shown in the **Uses** tool. Clicking on a match will show it in the editor, highlighting it briefly with a callout, as configured from the **Editor > Callouts** preferences group.

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

Result Display

Wing tries to show only symbols that are actually the same symbol, and not also other like-named symbols. However, 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 likelihood 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 from the **Options** menu on a per-search basis.

Finding Imported Symbols

When finding a symbol from an import statement, Wing defaults to finding where the imported module, class, function, or attribute is used with the same name in all files searched. To only find the symbol created by the import statement used to start the search, uncheck **Find Imported Items Everywhere** on the **Options** menu.

Improving Quality of Results

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** or place type hints so that the source analysis engine can determine the type of more symbols. See Helping Wing Analyze Code for details.

5.6. Wildcard Search Syntax

The following syntax is used for wild card searches in Wing's search tools:

* matches 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 *****.

? matches 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 **]** indicate sets of characters to match. 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**, either lower case or uppercase. Case specifications in character ranges will be ignored unless the **Case Sensitive** option is turned on.

Code Warnings and Quality Inspection

Wing Pro warns about possible problems with Python code, by underlining them in the editor and listing them in the **Code Warnings** tool. A warning indicates something might be wrong in the code, for example a syntax error, indentation problem, use of an undefined variable, an import that cannot be resolved, or a variable that is set but never used.

New code is checked as you work, although Wing will wait until you have finished typing so that it doesn't warn about code that is still being entered.

Wing's builtin warnings may be supplemented with warnings found with external code quality checkers like **Pylint**, **mypy**, and **pep8**.

Since all code checkers have only a limited understanding of what happens when code is actually run, they may show incorrect warnings. Wing allows you to disable specific warnings either for a single case, for an entire file, or for all files.

6.1. Code Warnings Tool

The **Code Warnings** tool in the **Tools** menu lists all the warnings found on the current editor. Clicking on warnings or pressing the **Enter** key in the list navigates to that warning in the editor, highlighting it briefly with a callout, as configured from the **Editor > Callouts** preferences group.

Disabling Warnings

The **Code Warnings** tool is also used to selectively disable warnings.

Individual warnings may be disabled by clicking on the red **X** that appears while moving the mouse cursor over warnings in the **Code Warnings** tool, or by selecting an item and pressing the **Delete** key. When this is done, Wing disables most individual warnings only for the scope it appears in. However, undefined attribute warnings are always disabled in all files.

The right-click context menu on the **Code Warnings** tool may be used to specify how widely to disable a warning, either only one specific case, all warnings of that type in the file, or all warnings of that type in all files.

For external checkers, warnings disabled by clicking the red **X** or pressing **Delete** are hidden globally by type. Note, however, that Wing does this without altering the external checker's regular configuration file. Editing the configuration for the external checker directly, as documented by the external checker, is another way to ignore some of its errors and warnings.

Configuration: Disabled Warnings

When a warning is disabled, Wing adds a rule to the **Configuration: Disabled Warnings** page in the drop-down menu at the top of the **Code Warnings** tool.

Rules are organized into those defined for the current file and those defined for any file. Configuration rules may be dragged between these two groups. Rules may be deleted by clicking on the red **X** that appears while moving the mouse cursor over the items, or by selecting them and pressing the **Delete** key.

Rules may be edited from the right-click context menu, in order to disable a broader or narrower set of warnings. For example, a rule to disable a specific undefined attribute warning can be changed to disable all undefined attribute warnings for the class by changing the **Attribute Name** field from **.attribute** to **.***.

Configuration: Defaults

The types of code warnings that Wing shows can be configured from the **Configuration: Defaults** page in the drop-down menu at the top of the **Code Warnings** tool. The warnings types Wing supports are documented in Warning Types and some of the warning types offer configuration options to control which variants of that type of warning will be shown.

Sharing Configurations

The current code warnings configuration may be exported to the user settings area, or to a selected file from the **Options** menu in the **Code Warnings** tool. Projects may then share the configuration through the **Use Configuration From** item in the **Code Warnings** tool's **Options** menu.

When the code warnings configuration is stored in user settings, it is written to a file named **code-warnings.conf** in the User Settings Directory.

When the code warnings configuration is stored to another file, it may be checked into revision control along with the **.wpr** file. The choice of which external configuration file is used is stored in the **.wpr** file so all users of that project will use the same code warnings configuration. Wing will be able to find the shared configuration as long as the relative path between the project and the configuration file remains the same.

Clearing the Configuration

The code warnings configuration may be reset to blank with the **Clear Configuration** item in the **Code Warnings** tool's **Options** menu.

6.2. Warnings on the Editor

The ⚠ code warnings icon appears in the top right of any editor that has some code warnings. This can be used to jump to each warning, force immediate update of the warnings in the file, disable all warnings in the file, or bring up the **Code Warnings** tool.

When code warnings are displayed on the editor, hovering the mouse cursor over the indicator will display details for that warning in a tooltip. The tooltip includes a red **X** icon that can be pressed to disable that warning in the same way as disabling it from the **Code Warnings** tool.

The way in which code warnings appear on the editor may be changed with **Indicator Style**, **Error Color**, and **Warning Color** in the **Editor > Code Warnings** preferences group.

6.3. Warnings Types

Wing's internal code checker supports following types of code warnings. Each of these may be configured from the **Configuration: Defaults** page from the drop-down menu at the top of the **Code Warnings** tool.

General

Import Not Found is shown when a module or package cannot be found on the configured Python Path. This may indicate that you may need to modify the **Python Path** in **Project Properties**, so that Wing can find your modules. In cases where this is not feasible, or if code is overriding the import, warnings of this type may be disabled instead.

Indent warnings are shown when an indent is not consistent in size or content (tabs vs. spaces) with indents found elsewhere in the file, or when an indent does not match the logical structure of the code. For example, the line after **if** and **for** must be indented, while the line after **return** or **raise** should be outdented. Code with inconsistent indent size or content may still be correct, and sometimes warnings of this type should be disabled.

Undefined Symbols

Undefined Name warnings are shown when a variable is used without the variable ever being set, or when a function is used without defining the function. This warning usually indicates broken code that should be fixed, and warnings of this type should be disabled only in rare cases.

Undefined Attribute warnings are shown when a class or instance attribute name appears to be undefined. This occurs when the attribute is not in the list of attributes that Wing has found for the type of symbol before the dot. This list is the same as the one that used for autocompletion on the object. Warnings of this type should be disabled in cases where Wing doesn't identify the object correctly or the list of attributes is incomplete. When disabled, an undefined attribute warning for the same attribute name and object type are ignored across all files.

Unused Symbols

Import Not Used warnings are shown when a name that is imported is not used anywhere in the file that it has been imported into. Warnings of this type should be disabled in cases where the name is used in another file, as an attribute of the module.

Variable Not Used warnings are shown when a variable is set but never used in any other code. When warnings of this type are enabled, additional configuration is possible with the **Configure** button, to control some of the common cases where this warning is unwanted. These cases include:

(1) By default, Wing does not warn about top level global variables in a file because they may be used as module attributes in another file. However, Wing still warns if **__all__** is set in the file and the unused global is not included in it.

(2) By default, Wing does not warn about unused variables if they are defined by unpacking a tuple, such as in **a, b = (1, 2)**. However, Wing still does warn about unpacked variables if *all* of the variables unpacked together are unused.

(3) By default, Wing does not warn about unused variables with names starting with **unused** or **dummy** since these are usually intentionally unused values. Additional regular expressions for identifying intentionally unused variables may be set in the configuration dialog.

Other unused variables are always ignored by Wing, including: (a) variables or methods set in a class scope, because they may be used as either class or instance attributes, and (b) loop variables such **idx** in **for idx in range(5)**.

Argument Not Used warnings are shown when a function or method argument is defined in the **def** statement but never used. This warning type is disabled by default because arguments often need to be included to match a desired standard signature. It may be enabled for code bases where this is not a frequent issue.

6.4. Advanced Configuration

The **Advanced Configuration** button at the bottom of the **Configuration: Defaults** page in the drop-down menu at the top of the **Code Warnings** tool may be used to control several other options for Wing's code warnings facility:

Show Warnings in the Standard Library controls whether Wing shows any warnings in files found in Python's standard library. This is disabled by default since most users are not in a position to make changes to this code.

Show Warnings in site-packages controls whether Wing shows any warnings in files that are in the **site-packages** directory in the Python installation. This is the location that third party modules are installed into Python by **pip** and other package managers. This is also disabled by default.

Allow Comments to Disable Warnings controls whether Wing will look for comments in code to indicate that all warnings of a particular type should be disabled in the scope in which the comment is found. This is enabled by default and uses a set of comment regular expressions that match the informal **pylint disable** standard developed by **Pylint** and a similar **wing disable** standard for Wing's internal code checker. The set of regular expressions may be edited and extended through the configuration dialog. Each expression may disable on class of code warnings, including all instances of that type of warning.

6.5. External Code Quality Checkers

Errors and warnings found by external checkers like **mypy**, **pep8** and **pylint** may be interleaved with those found by Wing. Wing will filter the warnings through its list of rules to disable warnings. This can be used to quickly disable warnings that are stylistic in nature and not real problems in code.

To enable any external checker, check the **Enable External Checkers** option at the bottom of the **Configuration: Defaults** page in the **Code Warnings** tool. Then press the **Configure** button to select which checkers to enable and when to run them. External checkers may be run when a file is opened, after it is saved to disk, or both. Checkers will also be re-run if warnings are updated manually from the code warnings menu in the editor or the **Options** menu in the **Code Warnings** tool.

The command line used to run the checker is configured under its tab in the **Configure** dialog. By default, Wing runs the **Python Executable** configured in **Project Properties** with the **-m** argument to load the checker. This means that the checker must be installed into the selected Python, usually with **pip** or **conda** if using Anaconda Python.

Note that some checkers take a long time to run on even moderately sized source files and may consume significant amounts of CPU time. To prevent checks from consuming too many resources, Wing will skip checks on any file above the threshold set in the **Maximum File Size** option in the

external checker's configuration. When a file is skipped, a message will appear briefly in the status area at the bottom of the IDE window.

The configuration page for **pylint** includes also options for enabling or disabling warnings based on the priority assigned by Pylint (errors, warnings, and informational messages).

Once external checkers have been configured, Wing runs them, parses the output, and merges its warnings into the **Code Warnings** tool and editor's code warnings indicators. To view the raw output of the checkers that Wing is running, select **Show Console** from the **Options** menu in the **Code Warnings** tool.

Refactoring

Wing Pro supports refactoring, which is the process of modifying code to improve its structure and organization without changing its behavior. These very high-level editing operations are informed by Wing's understanding of Python source code. 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.

7.1. 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's right-click context menu. 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 **Enter** or click on the **Rename Checked** button.

Each match found 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.

7.2. 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's right-click context menu. 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 **Enter** or click on the **Move and Update Checked** button.

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

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

7.3. 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 extracted in the editor and then select **Extract Function/Method** from the **Refactor** menu or from the **Refactor** sub-menu of the editor's right-click context menu. 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 **Enter** or click on 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.

7.4. Introduce Variable

The **Introduce Variable** operation adds a variable that is initialized to the value of an existing expression and then replaces all occurrences of that expression with the new variable. To start an introduce variable operation, select an expression in the editor and choose **Introduce Variable** from the **Refactor** menu or from the **Refactor** sub-menu of the editor's right-click context menu. 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 **Enter** or click on the **Introduce Variable** button.

The introduced variable name may include a dot. For example, 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 introduce variable operation.

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

7.5. Symbol to *

Several **Symbol To *** refactoring operations are given to easily convert the name of a symbol between **UpperCamelCase**, **lowerCamelCase**, **under_scored_name**, and **ALL_CAPS_UNDER_SCORED_NAME** naming styles. These work the same way as **Rename Symbol** but prefill the new symbol name field with the selected style of name.

Difference and Merge

Wing Pro provides single and multi-file difference and merge capabilities that can be used to compare files or directories on disk and to manage differences to an Integrated Version Control system.

To initiate a session, click on the **Diff/Merge** toolbar item or use the **Difference and Merge** sub-menu of the **Source** menu. You will be prompted for any file or directory names in a dialog or, for some keyboard personalities, 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 active **Diff/Merge** 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 **Difference and Merge** sub-menu of the **Source** 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.

Session Types

The following types of **Diff/Merge** sessions are available:

Compare Files compares two selected files on disk.

Compare Directories compares two selected directories on disk. The **Diff/Merge** tool, which will be shown while the 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 compares the two visible editors. This is only available when two or more editor splits are shown and two different files are open in them. If three or more splits are shown, the files in the last two splits are compared.

Compare Buffer with Disk compares the current unsaved editor and its disk file. This is only available when the current file has unsaved edits.

Compare Recent provides a sub-menu for quick access to recently performed comparisons.

Compare to Repository can be used to compare the working copy of a file with the corresponding repository revision. This is only available if the file is checked into one of the version control systems that Wing Pro supports.

Options

The **Difference and Merge** sub-menu of the **Source** menu contains two items that control the behavior of **Diff/Merge** sessions:

Lock Scrolling keeps the scrolling position of the two files in the **Diff/Merge** session synchronized.

Ignore Whitespace ignores changes that consist solely of white space (space, tab, line feed, or carriage return characters).

These are also available in preferences, along with the following:

The **Editor > Diff/Merge > Empty Session Warning** preference chooses whether a warning should be shown if some changes were ignored because of the **Ignore Whitespace** setting.

The **Editor > Diff/Merge > Orientation** preference selects between side by side or top/bottom orientation of the two files shown during a **Diff/Merge** session.

The **Editor > Diff/Merge > Diff Color** and **Editor > Diff/Merge > Merged Diff Color** preference selects the color used in the **Diff/Merge** highlights on the editor.

Source Code Browser

The **Source Browser** in Wing Pro and Wing Personal provides an index into your source code, from either a module-oriented or class-oriented viewpoint.

9.1. Display Choices

The **Source Browser** provides three ways in which to browse your source code. These are selected from the menu at the top left of the tool:

Browse Project Modules displays the structure of all directories, packages, and modules in your Project, and their contents.

Browse Project Classes shows a list of all classes found in the project. Methods and attributes are shown within each class, along with any derived classes. Right-click on a class to navigate to super classes.

Browse Current Module restricts 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 acts as an index into the current editor file.

9.2. Symbol Types

The following types of items may be displayed in the **Source Browser**, each with its own icon:

Packages, which are directories that contain a file named **__init__.py**. See the Python documentation for additional information on packages.

Directories that do not contain an **__init__.py** file.

Modules defined by Python files.

C: Classes found anywhere in Python source

M: Methods defined within classes

● **Attributes** defined in a class or instance

F: Functions defined at the top-level of a module or within another function or method

◆ **Variables** defined at the top-level of a module or within a function, class, or method

Symbols may be annotated to indicate their origin:

← Symbols that were imported from another module are annotated with a leftward pointing arrow.

↑ Symbols inherited from a superclass are annotated with an upward pointing arrow.

The **Source Browser** does not include function or method arguments, but these may be displayed in the **Source Assistant**, along with other information for the currently selected item in the **Source Browser**.

9.3. Display Filters

The display can be filtered from the **Options** menu according to Symbol Type and Origin, and also according to the symbol's intended scope of use, which is defined as follows:

Public symbols are accessible to any user of a module or instance. These are names that have no leading underscores, such as **Print** or **kMaxListLength**.

Semi-Private symbols are intended for use only within related modules or from sub-classes or closely related classes. These are names that have one leading underscore, such as **_NotifyError** or **_gMaxCount**. Python doesn't enforce usage of these symbols, except to omit them in **from mod import ***. However, they are helpful in writing clean, well-structured code and are recommended in PEP 8.

Private symbols are intended to be private to a module or class. These are names that have two leading underscores, such as **__ConstructNameList** or **__id_seed**. Python omits these in **from mod import ***. When used in classes, they cannot be accessed from outside of the methods of the class where they are defined. See PEP 8 for details.

9.4. Sorting the Display

The symbols within a module or class can be sorted from the **Options** menu:

Sort Alphabetically displays all items in alphabetic order, regardless of type.

Sort by Type sorts first by symbol type, and then alphabetically.

Sort in File Order sorts the contents of each scope in the same order that the symbols are defined in the source file.

9.5. Navigating the Views

Double-clicking on an item in the **Source Browser** navigates to that symbol in the editor.

Files visited from the **Source Browser** are opened in transient mode and may automatically close, if not edited. See Transient, Sticky, and Locked Editors for details.

The option **Follow Selection** in the **Options** menu causes the browser to open files whenever the currently selected item changes.

Right-clicking on classes shows a popup menu that includes items for navigating to super classes.

Keyboard Navigation

Once it has the focus, the **Source Browser** is navigable from the keyboard, using the arrow keys, page up and page down, and home/end. Press the right arrow key on a parent to expand it, and the left arrow key to collapse it. Pressing **Enter** or **Return** will open the current item into the editor.

Callouts

Source Code Browser

When a symbol is visited in the editor, Wing highlights it briefly with a callout, as configured from the **Editor > Callouts** preferences group.

Integrated Python Shell

The integrated **Python Shell** is used to execute or debug commands and expressions interactively, in a way that is tightly integrated with Wing's editor, code inspection, and debugger features.

The **Python Shell**'s auto-completer uses introspection of the runtime environment as a powerful way to find and inspect functionality and craft new code interactively. The Source Assistant in Wing Pro and Wing Personal displays documentation, call signature, and other information about symbols as you work in the **Python Shell**

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

Evaluating Code from the Editor

There are several ways to evaluate code from an editor within the **Python Shell**:

> **Copy and Paste** and **Drag and Drop** adjust leading indentation and execute the code.

> **Evaluate File in Python Shell** in the **Source** menu restarts the **Python Shell** and then evaluates the top level of the current file. Restarting can be disabled by unchecking **Auto-restart When Evaluate File** in the **Options** menu at the top right of the tool. This operation sets the value of **sys.argv** to match the value that would be used if the file were debugged. If a launch configuration has been selected in the Python Shell's **Options** menu then its run arguments are used instead.

> **Evaluate Selection in Python Shell** in the **Source** menu evaluates the current selection in the shell. This is also available in the editor's right-click context menu.

> **Set an Active Range** from the editor into the **Python Shell** so it can be executed or debugged repeatedly during editing. See Active Ranges in the Python Shell for details.

> **The Options menu** in the **Python Shell** tool also contains items for evaluating the current file or selection

To clear the shell's state at any time, use **Restart Shell** in the **Options** menu.

Debugging

Code entered into the **Python Shell** may be executed with or without debug. When debugging is enabled, execution will reach breakpoints, allow stepping through code, and support inspection of runtime state. See Debugging Code in the Python Shell for details.

In Wing Pro, the Debug Console can be used to interact in a similar way with the current frame of a debug process.

Command History

The **Up** and **Down** arrow keys traverse the history of the code you have entered and the **Return** key executes the code if it is complete, or prompts for another line if it is not. If **Filter History by Entered Prefix** in the **Options``menu**

is checked then any text typed before pressing ``Up will be used to filter the history items that are traversed.

Code recalled from history can be edited within the **Python Shell**. Use **Ctrl-Up** and **Ctrl-Down** to move the caret up and down and **Ctrl-Return** to insert a new line at the caret position.

To save the contents of the shell, use **Save a Copy** in the **Options** menu or the tool's right-click context menu. The context menu also provides items for copying text from the shell, with or without prompts.

10.1. Python Shell Environment

Code typed, pasted, dropped, or otherwise entered into the **Python Shell** executes in a separate Python process that is independent of the IDE and functions without regard to the state of any running debug process.

The version of Python used in the **Python Shell**, and the environment it runs with, including initial working directory, is configured in Project Properties from the **Project** menu, or by selecting a particular Launch Configuration from **Use Environemnt** in the **Options** menu.

To preload some code into the Python Shell when it is started, you can set the **PYTHONSTARTUP** environment variable to the full path of a Python file. Or, set **PYTHONSTARTUP_CODE** to execute a line of Python code, optionally with multiple statements separated by **;**

10.2. Active Ranges in the Python Shell

Code in an editor can be marked as the active range for the **Python Shell**, in order to make it easier to reevaluate after it is edited. This is done by selecting a range of lines in the editor and pressing the **Set Active Range** icon at the top right of the **Python Shell**.

Once a range is set, additional icons appear to execute or debug the active range, jump to the active range in the editor, or clear the active range.

The active range is marked in the editor and will adjust its position and extent as code is added or deleted.

10.3. Debugging Code in the Python Shell

Code executed in the **Python Shell** can be run with or without debug. This is controlled by clicking on the bug icon in the upper right of the tool, or using **Enable Debugging** in the **Options** menu.

When debugging is enabled, a breakpoint margin appears at the left of the **Python Shell** tool, and breakpoints can be set, as in editors. This works for code previously typed, dragged, or pasted into the shell. Breakpoints set in editors are also reached, if that code is executed. Wing copies breakpoints from a source file and stops in the **Python Shell** itself when **Evaluate Selection** is used on a short enough range of code. However, when using active ranges, or when evaluating a long selection or a whole file, Wing instead stops at breakpoints set within the code editor, since in those cases the code is not visible in the shell.

Note that the debugger only appears active when code is actually running, and not when waiting at the **Python Shell** prompt.

Whenever code is being debugged from a shell prompt, **Stop Debugging** and **Start/Continue** in the **Debug** menu, and their keyboard and toolbar equivalents, will return to the prompt in the shell. Both will continue executing code to complete the invocation from the prompt but **Stop Debugging** will do so with debug temporarily disabled. The fact that code is not preemptively interrupted is a limitation stemming from the way Python is implemented. In cases where this is a problem, the **Python Shell** can be restarted instead.

Recursive Debugging

In Wing Pro, to interact recursively with code debugged from the **Python Shell**, use the **Debug Console** or turn on **Enable Recursive Prompt** in the **Options** menu. The latter presents a new prompt in the **Python Shell** whenever the debugger is paused or at a breakpoint, even if that shell is already in the process of executing code.

Debugging Threaded Code

Threads are treated differently in the **Python Shell** and Wing Pro's **Debug Console** depending on whether or not debug is enabled and/or whether the shell is at the prompt, as follows:

In the **Python Shell**, when debugging is disabled, threads are run continuously in the background without debug and whether or not the shell is at a prompt. When debugging is enabled in the **Python Shell** it will also debug threads. However, it will allow threads to run only while code is being executed from the shell and the **Python Shell** is not at the prompt. This matches the behavior of the debugger when it is running stand-alone files, where it halts all threads if any thread is halted. When the **Python Shell** is debugged, Wing treats execution of code from the shell prompt as continuing the debugger until the prompt is reached again. Thus it also allows other threads to run during this time.

In the **Debug Console**, when debugging is disabled in its **Options** menu, threads are debugged but are halted whenever the main thread is halted in the debugger. Threads are not run even while executing code from the prompt in the **Debug Console** so that data in all threads can be inspected without any unexpected change in runtime state caused by running of a thread. Threads will only continue running when the main debug program is continued. This is true whether or not the debug program was started from a file, or from within the **Python Shell**. As in the **Python Shell**, when debugging is enabled in the **Debug Console** child threads will also be allowed to run whenever code is being executed recursively and the **Debug Console** is not at the prompt. Threads are still halted whenever the **Debug Console** is at the prompt

These subtle but necessary differences in threading behavior may affect how threaded code performs within the **Python Shell** and Wing Pro's **Debug Console**. Currently there are no options for selecting other behaviors, such as always letting threads run even when at the prompt, or never letting threads run even when executing code from the prompt. If you run into a situation where one of these options is needed, please send details of your use case to support@wingware.com.

10.4. Python Shell Options

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

Enable Debugging controls whether code run in the **Python Shell** will be debugged.

Enable Recursive Prompt in Wing Pro can be used to cause the **Python Shell** to present a new prompt when debugging, even if the previous prompt invocation has not completed because the debugger is paused or at a breakpoint or exception. Execution returns to the previous prompt when the debug process is continued.

Enable Auto-completion controls whether Wing will show the auto-completer in the **Python Shell**.

Wrap Lines causes the shell to wrap long output lines in the display.

Pretty Print causes Wing to use Python's **pprint** module to format output.

Filter History by Entered Prefix causes up/down arrow key traversal of history to match only items that start with the string between the prompt and the caret. If no string was typed before pressing the up arrow then all history items are traversed.

Evaluate Only 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.

Auto-restart when Switch Projects causes Wing to automatically restart the shell after switching projects, so that the shell environment will match the project's configuration.

Prompt to Confirm Restart controls whether Wing will prompt before restarting the **Python Shell**.

Use Environment in Wing Pro and Wing Personal selects the runtime environment, including initial working directory, for the **Python Shell**. This may be Project Properties or a selected Launch Configuration. When this is changed, the shell must be restarted from its **Options menu** before a newly selected environment takes effect.

Edit Environment in Wing Pro and Wing Personal edits the runtime environment selected with **Use Environment**. This highlights the initial working directory property, but all of the properties may be changed. The shell must be restarted from its **Options** menu before the edited environment takes effect.

Prompt on Stale Environment controls whether Wing will display a dialog indicating that the **Python Shell** is no longer using a Python environment that matches the configured environment.

OS Commands Tool

The **OS Commands** tool in Wing Pro and Wing Personal executes and interacts with external commands provided by the OS or by other software. It can be used to execute Python code outside of the debugger, run build commands, integrate external tools into Wing, start code that is debugged using **wingdbstub**, and so forth.

Adding and Editing Commands

There are three types of OS Commands:

(1) *Command Lines* are executed in the environment configured in the OS Command itself.

(2) *Python Files* are executed in the environment configured in their File Properties.

(3) *Named Entry Points* are executed in the environment configured by the selected Named Entry Point.

Commands can be added, edited, and deleted with the icons in the **OS Commands** tool and from its **Options** menu.

Additionally, whenever a file is executed outside of the debugger, or when a build command is configured in Project Properties or Launch Configurations, these are added automatically to the **OS Commands** tool.

For details on setting up a new command, see OS Command Properties.

Executing Commands

Commands can be executed and terminated from icons in the **OS Command** tool and from its **Options** menu.

The bottom portion of the **OS Commands** tool contains 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 running processes, or add multiple instances of the **OS Commands** tool to view them at the same time.

The console provides a right-click context menu for controlling the process, copy/pasting, and clearing or saving a copy of the output to a file.

Start Terminal

On Linux and OS X, or when working with a project that points to a remote host, Wing offers **Start Terminal** in the **OS Commands** tool menus and the **Tools** menu in the menu bar. This configures and starts a new *Command Line* style OS Command that runs a **bash** terminal.

For projects that use a virtualenv Python, the terminal will be started after running **activate**. This is set in the **Command Line** property, which can be set back to **bash -norc** to avoid activating the virtualenv.

To set up a terminal that runs a different shell, add a *Command Line* style OS Command with **Command Line** set to your shell executable (for example, Wing's default terminal configuration uses **bash -norc**) and then enable the **Use pseudo-TTY** and **Line mode** options.

Note that Wing's **OS Commands** tool does not fully emulate a TTY, so the tab key, color, and cursor movement are not supported. As a result of this, Wing sets **TERM=dumb** in the **Environment** in the OS Command configuration for terminals.

Options

The **Options** menu includes items for restarting a command and clearing the execution console, and also the following options:

- **Auto-Clear Consoles** controls whether the console is automatically cleared each time a command is started or restarted.
- **Python Prompt After Execution** controls whether *Python File* style commands drop into the Python prompt after the file is executed, rather than exiting the process.
- **Wrap Long Lines** controls whether long lines are shown on a single line or wrapped to the width of the **OS Commands** tool.

Toolbox

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

11.1. OS Command Properties

The runtime environment for commands added to the **OS Commands** tool is configured in the dialog shown when the item is added or edited.

Shared Properties

All OS Command types share the following configurable properties:

Title is the display title to use for the command. If not set, the command line or file name is shown instead.

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

Key Binding assigns a key binding to execute the command. To enter a binding, just press the desired binding while focus is in the **Key Binding** field. Bindings can consist of multiple parts, such as **Ctrl-H B**. Pressing multiple keys will create a key binding sequence, unless too much time elapses between the key presses. To reset the value to blank (no key binding), select all text and press **Backspace** or **Delete**.

Raise OS Commands When Executed 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 automatically saves any unsaved changes in open files before the command is executed, even if the **Files > Auto-Save Files Before Debug or Execute** preference is disabled.

Use Pseudo-TTY (on Linux and OS X) 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. This option is omitted for OS Commands being executed on Windows.

Line Mode (on Linux and OS X) can be disabled to 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. This option is omitted for OS Commands executed on Windows, and all I/O is performed line by line.

Shared stores the OS Command in the Settings Directory so that it appears in all projects.

Additional Properties for Command Lines

The **Environment** tab provided for *Command Line* style OS Commands allows specifying the **Initial Directory**, **Python Path**, and **Environment**, which act the same as the corresponding values configurable in Project Properties.

Hostname (only in Wing Pro) is used with *Command Line* style OS Commands to select the remote host where the command should be executed. For *Python File* and *Named Entry Point* style OS Commands, the hostname on which the command will execute is determined by the location of the Python file. See Remote Hosts for details.

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.

Additional Properties for Python Files

For *Python File* style OS Commands, **Python Prompt after Execution** in the **Options** menu specifies that the Python interpreter should be left active and at a prompt after the file is executed.

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.

11.2. Sharing Projects with OS Commands

By default OS Commands are stored in the ***.wpr** branch of the project file, which in Wing Pro may be checked into a revision control system or otherwise shared with other users and hosts. If the project will be used on different OSes or differently configured systems, some extra work may be needed to configure the same OS Commands to work properly on each host.

Using Environment in Configuration

The best option to manage OS Commands shared across different environments is to use environment variable references in the OS Command's properties. Environment variables used in OS Commands can be defined differently by each user of the project in Project Properties. Because these are stored in the per-user ***.wpu** branch of the project file (and not the shared ***.wpr**) the values can differ for each host on which a project is used.

For example, instead of specifying **bash -norc** for a *Command Line* style command, the environment variable **USERSHELL** could be set in the **Environment** in Project Properties to the user's preferred shell, and then the OS Command could reference that value with **${USERSHELL}**.

Environment variables can also be defined for directories used as the **Initial Directory**, in the **Python Path**, or for any other value needed for any of the other properties of an OS Command.

In addition to referencing user-defined environment variables, OS Commands may reference any of the special environment variables listed in Environment Variable Expansion.

Storing OS Commands Locally

Another option to keep some OS Commands out of the shared ***.wpr** branch of the project is to mark them as **Shared** in their configuration. This causes them to be stored in the User Settings directory and not the project file. Thus they will be omitted from the ***.wpr** that is commited to revision control.

Storing OS Commands in the Per-User Project File

In Wing Pro, it is also possible to reconfigure a project to cause all the OS Commands in the project to be stored in the user-specific ***.wpu** branch of the project file. This is done by removing **console.toolbox** from the **proj.shared-attribute-names** property in the ***.wpr** file, as described in more detail in the section "Changing Which Properties are Shared" in Sharing Projects.

Unit Testing

Wing Pro's **Testing** tool provides a convenient way to run and debug unit tests written using the standard library's **unittest** and **doctest** modules, **pytest**, **nose**, and the Django testing framework.

Adding Tests

Tests are added from the **Testing** menu, with **Add Single File** and **Add Current File** to add individual files, or with **Add Files from Project** to apply a filter to the set of all files in the project. For details, see Project Test Files.

The testing framework used to run files is selected with **Default Test Framework** under the **Testing** tab of Project Properties or with **Test Framework** under the **Testing** tab of File Properties for individual test files.

Running Tests

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

While tests are running, the **Testing** tool updates to indicate the status of the run. After the tests have finished running, the status icon for each test will change to indicate the result of the run:

✔ indicates the test passed

✖ indicates the test failed

↪ indicates the test was skipped

? indicates the test was not run or did not complete

Viewing Test Results

Individual tests may be expanded to show output generated by the test or any exception that occurred. Exceptions, including any PEP 3134 chained exceptions, may be expanded to display tracebacks.

Collapse All Tests and **Expand All Failed Tests** in the right-click context menu in the **Testing** tool can be used to quickly hide all test details, or show details only for failed tests.

Double-click on any test or use **Goto Source** in the right-click context menu in the **Testing** tool to display the source code for the test in the editor.

To focus on a subset of the test files, enter a fragment matching those test file names into the **File Filter** field in the **Testing** tool. Restore the field to blank redisplays the entire lists of tests.

Output shown for tests may optionally be wrapped to fit the display by checking the **Wrap Output Lines** item in the right-click context menu on the **Testing** tool.

12.1. Project Test Files

A subset of all the files in the project may be added to the **Testing** tool by specifying one or more **Test file patterns** under the **Testing** tab of Project Properties. This can be initiated with **Add Files from Project** in the **Testing** menu.

Any project file that matches a test file pattern is considered to be a test file, and will be displayed in the **Testing** tool. The list will update automatially as project files are added and removed or the contents of project directories changes.

Test file patterns can be applied to the full path of the test file. For example, the wildcard pattern **internals*/*/test_*.py** would match files named **test_*.py** in any directory below a directory with a name starting with **internals**. A similar approach works with regular expression style patterns. For details on the syntax for wildcards, see Wildcard Search Syntax. For details on the syntax for regular expressions, see Regular Expression Syntax in the Python documentation.

12.2. Running and Debugging Tests

Tests can be run and debugged from the **Testing** menu, in the following ways:

- **Run All Tests** runs all the unit tests listed in the **Testing** tool.
- **Run Tests in Current File** runs all the tests found in the current editor.
- **Run Tests at Cursor** runs the test or tests at the caret or selection in the current editor
- **Run Failed Tests** reruns all the tests marked as failed in the **Testing** tool.
- **Run Tests Again** reruns all the tests that were run the last time tests were run.

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

Tests are run in the order they are shown in the **Testing** tool.

To stop running tests, press **Abort Tests** in the **Testing** tool or select **Abort Running Tests** from the **Testing** menu.

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

Debugging

For each of the 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.

When tests are debugged, output goes to the **Debug I/O** tool and the contents of the **Testing** tool are not updated with the results of the test.

Unexpected Exceptions

Some testing frameworks such as **pytest** may stop at internal exceptions that should be ignored by clicking on **Ignore this exception location** in the **Exceptions** tool. This occurs when the testing framework raises and then handles **AssertionError** in order to probe the capabilities of the running Python. By default, Wing will always stop on assertions, even if they are handled, because in most cases a failing assertion indicates a bug in code. Once ignored, Wing won't stop on these internal exceptions again and debugging can proceed as usual.

Execution Options

There are several options available for how Wing runs unit tests.

Process Model

When multiple test files are run at once, they may be run in a separate process for each file (the default), or all test files in one directory may be run in a single process. This is selected with **Process Model** under the **Testing** tab of Project Properties.

In the **Per-Module** model, Wing is running the equivalent of the following command line:

```
cd /path/to/files
python -m unittest one.py
python -m unittest two.py
```

In the **Per-Package model**, Wing is instead running the equivalent of:

```
cd /path/to/files
python -m unittest one.py two.py
```

In both cases all tests should be run, but two processes are used in the first case and only one in the second case. Which model you choose depends on the requirements of your test suite.

Running Tests Concurrently

Two or more test processes may be run in parallel by increasing the **Number of Processes** under the **Testing** tab of Project Properties. This can increase performance on systems with multiple CPU cores, but may introduce problems if the tests do not handle concurrency well.

Running Test Packages

When test files that are located in a package (a directory that contains **__init__.py**), they may be loaded either as package modules, or as top-level modules. Each testing framework defines a default behavior for this case, but this can be overridden using **Run as Package Modules** under the **Testing** tab of Project Properties.

When files are loaded individually as package modules, Wing is running the equivalent of:

```
python -m unittest package.module
```

When files are loaded as a top-level package, Wing is running the equivalent of:

```
python -m unittest module
```

12.3. Running unittest Tests from the Command Line

Wing's test runner for the **unittest** testing framework can be run from the command line, in order to store results in an XML file that can be loaded into Wing later using **Load Test Results** in the **Testing** menu. The test runner is **src/testing/runners/run_unittest_xml.py** within the **Install Directory** listed in Wing's **About** box. It should be started as in the following example:

```
/path/to/python /path/to/wing/src/testing/runners/run_unittests_xml.py [options] -q testModule.className.testName
```

Replace **/path/to/python** with the Python that should be used to run the tests, **/path/to/wing** with the installation directory for Wing, **[options]** with any of the command line options listed below, and **testModule.className.testName** with the real test specification.

In the test specification, **testModule** is the module name (without **.py**), **className** is the test class name, and **testName** is the name of the test method to run. To run all tests in a class, omit **testName**. To run all tests in a module, also omit **className**.

Command Line Options

--directory=<dirname> runs the tests in the given directory. When omitted, the tests are run in the current directory, inherited from the command line.

--output-file=<filename> writes results to the selected file. When omitted, results are written to **stdout**.

--append-to-file appends results to the file selected with the **--output-file** option, rather than truncating the file.

--one-module-per-process runs 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> runs tests in each filename matching the given pattern. This option may be repeated multiple times with different wildcards. This option also turns on the **--one-module-per-process** option.

Note: Only the **unittest** test runner supports running from the command line. The other test runners cannot be used in this way.

Debugger

Wing Pro's debugger provides a powerful toolset for rapidly locating and fixing bugs in single and multi-threaded Python code, and in a single or multi-processing environment. The debugger 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.

There are a number of ways to use Wing's debugger. Which you use depends on where your code is running, and how it is invoked:

Local Stand-Alone Code -- Wing can debug stand-alone scripts and applications that run on your local machine and that are launched on demand from within Wing. For details, see Debugger Quick-Start.

Remote Stand-Alone Code -- Wing Pro can debug stand-alone code remotely in the same way as it debugs locally running code. This is done by configuring a remote host to which Wing will connect via SSH tunnel. For details on this, see Remote Hosts.

Local Embedded or Externally Launched Code -- Wing can also debug code that runs within a web server, in an embedded Python instance (for example as a script that controls a larger application), and other code that cannot be directly launched from the IDE. For details, see Debugging Externally Launched Code.

Remote Embedded or Externally Launched Code -- Finally, Wing Pro can debug externally launched or embedded code that is running on another host. For details, see Debugging Externally Launched Remote Code.

Because the debugger core is written in optimized C, debug overhead is relatively low. However, you should expect your programs to run 25-50% slower within the debugger in most code. Overhead is proportional to number of Python byte codes executed, so code that does a lot of work in Python and very little in support libraries will incur more overhead.

13.1. Debugger Quick Start

Wing can be used to debug all sorts of Python code, including applications written with wxPython, Tkinter, PyQt, PyGObject, matplotlib, Jupyter and pygame. Wing can also debug code written for frameworks like Django, Flask, Pyramid, mod_wsgi, Plone and Twisted, and code running in an embedded Python interpreter in the context of a larger application such as Blender, Maya, Nuke, and Source Filmmaker.

Note: This section describes how to get started with Wing's debugger when you are working with locally stored code that can be launched from the IDE. If you need to launch code from outside of the IDE or on a remote host, see the options given in Debugger.

Before debugging, you will need to install Python on your system if you have not already done so. Python is available from python.org, or you can use a distribution like Anaconda to also install many third party libraries.

To start debugging some Python code, open up the file in the editor and then 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. For details see Starting Debug.

To set breakpoints, just click on the left-most margin next to the source code in the editor. In Wing Pro, 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. For details, see Setting Breakpoints.

Use the Debug I/O tool to view your program's output, or to enter values for input to the program you are debugging. 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 need to specify a **Python Executable**, **Python Path** or other environment using **Project Properties** in the **Project** menu. Setting the **Python Executable** is only necessary if Wing cannot find Python on your system or if you have more than one version of Python installed. Command line arguments to use when debugging a file may be set in File Properties for the file. See Debug Environment for more options.

There are many other capabilities available in the debugger, as described in the rest of this chapter and Advanced Debugging Topics.

13.2. Debug Environment

The Python executable that should be used for debugging, and environment like **Python Path** and starting directory, are specified in Project Properties in the **Project** menu.

Per-File Environment

In cases where different debug environments are needed for different files, use File Properties for each file to specify a Launch Configuration to use with that file.

If different debug environments are needed for different launches of the same file, set up a Named Entry Point instead.

Command Line Arguments

Command line arguments to use when debugging a file can be set using **Debug Environment** in the **Debug** menu, under the **Debug** tab of File Properties for the file, or by defining a Named Entry Point.

Unit Testing Environment

The environment to use for files when they are debugged as unit tests by the Testing tool is instead set under the **Testing** tab of Project Properties or File Properties.

13.3. Named Entry Points

Named entry points define additional debug/execute entry points into Python code, by pairing a Python file or a named module with the desired execution environment. Named entry points can be debugged or executed from the **Debug Named Entry Point** and **Execute Named Entry Point** sub-menus of the **Debug** menu.

Named Entry Points in the **Debug** menu displays the **named entry point manager**. Use the toolbar in the dialog or 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.

Right-click on a named entry point in the list and select **Set Debug Key Binding** or **Set Execute Key Binding** to assign a key binding that will debug or execute that named entry point.

Named Entry Point Fields

Each named entry point defines the following fields to launch either a Python file or a named module using **python -m**.

Python File is the file to launch by invoking Python with its filename.

Named Module is the module to launch with **python -m** and the module named.

Environment specifies the environment to use when launching the file. This can either be the project-defined environment from Project Properties with the specified run arguments, or it can be a selected launch configuration. For Python files selected by filename rather than module name, the file may also be launched using **python -m** and the specified run arguments.

Show this dialog before each run displays the named entry point properties dialog before debugging or executing it.

13.4. Specifying Main Entry Point

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

When a main entry point is defined, it is used whenever you start the debugger, except if a specific file is debugged, for example with **Debug Current File** in the **Debug** menu.

The path to the main entry point, if one is set, is highlighted in bold text in the **Project** tool.

The main entry point defined for a project is also used by source code analysis to determine the Python path to use for analysis. As a result, changing this value will cause partial reanalysis of all source files. See Source Code Analysis for details.

13.5. 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 Pro, 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 caret or selection is located).

Breakpoint Types

In Wing Pro, the following types of breakpoints are available:

Regular breakpoints will always cause the debugger to stop on a given line of code, whenever that code is reached.

Conditional breakpoints contain an expression that is evaluated each time the breakpoint is reached. The debugger will stop only if the condition 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 breakpoints are removed automatically after the first time they are encountered. No record of the breakpoint is retained for future debug sessions.

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 in the **Debug** menu, in the breakpoint margin's right-click context menu, and from the **Breakpoints** tool:

Ignore Count ignores a breakpoint a given number of times. The debugger will only stop if it is reached more often than that. 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 can be used to temporarily disable and subsequently re-enable breakpoints. 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 is checked if the breakpoint is enabled.

Location gives the file and line number where the breakpoint is set.

Condition lists the conditional that must be true for the breakpoint to cause the debug process to stop. This is blank if the breakpoint is not conditional.

Temporary is checked if the breakpoint is a temporary one-time breakpoint.

Ignores indicates the number of times the breakpoint should be ignored before it causes the debugger to stop.

Ignores Left shows the number of ignores left for the breakpoint, for the current debug process.

Hits shows the number of times the breakpoint has been reached in the current debug process, if any.

Most of these values can be edited by clicking on the list. To delete the selected breakpoints, press the **Delete** key.

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 right-click context menu. Additional editing 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.

13.6. Starting Debug

The following items in the **Debug** menu, or their key bindings, can be used to start debugging:

- **Start / Continue** runs the main entry point, if one has been set as described in Specifying Main Entry Point, or otherwise the file open in the current editor. Execution stops at the first breakpoint or exception, or upon program completion.
- **Step Into** starts a debug session that stops at the first line of code.
- **Debug Current File** runs the file from the current editor. This will stop on the first breakpoint or exception, or upon program completion.
- **Run to Cursor** starts or continues debugging until it reaches the line selected in the current editor, until a breakpoint or exception is encountered, or until program completion.
- **Debug Recent** can be used to rerun a recent debug session. This will stop on the first breakpoint or exception, or upon program completion.

Other ways to start debug include:

- **Debug Selected** in the right-click context menu on the **Project** tool runs the selected file.
- In Wing Pro, Named Entry Points can be used from the **Debug** menu, to debug or execute files in a particular environment.
- Code may also be debugged from the **Python Shell** tool by clicking on the bug icon in the top right of the tool and entering some code or using the **Evaluate** options in the **Source** menu. See Debugging Code in the Python Shell for details.
- In Wing Pro and Wing Personal, debug may also be initiated from outside of Wing. See Debugging Externally Launched Code for details.

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 that when debugging code from the **Python Shell** the debugger only appears active if code is actually running and the shell is not at the prompt.

Custom Python Compilations

Wing's debugger contains an extension module that uses the cross-Python API to support multiple versions of Python with a single compilation of the module. This should cover most custom compilations of Python. However, if you need to support a new OS or device, or an unusual compile configuration, you may need to recompile the debugger core to match your compilation of Python. This is possible for Wing Pro users, through access to the source code under NDA. Please contact us for details.

13.7. Debugger Status

The debugger status indicator in the lower left of editor windows is used to display the state of the debugger. 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.

These colors may vary with customization of the user interface. Hover the mouse over the bug icon to display expanded debugger status information in a tool tip.

The status of the debugger is also reflected in the toolbar, which adds items while a debug process is active.

13.8. Flow Control

Once the debugger is running, the following commands are available to control further execution of the debug process from Wing.

Stepping Through Code

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**. When used on an import, this will skip Python code executed in **importlib** and instead will step directly into the imported module. This behavior can be disabled with the **Debugger > Advanced > Step Past importlib Frames** preference.

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**.

Pausing and Terminating Debug

At any time, a freely running debug process 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 process, as long as some Python code is being executed.

At any time during a debug session, the **Stop Debugging** menu item or toolbar item can be used to force termination of the debug process. This option is disabled if the current process was launched outside of Wing. It may be enabled for all local processes by using the **Debugger > Listening > Kill Externally Launched Processes** preference.

Move Program Counter

Move Program Counter Here in the editor's right-click context menu moves the current position of the execution counter within the innermost stack frame to any other valid position within the same scope. Stepping or execution will continue with the selected line.

Because of how Python is implemented, this feature works only in the innermost stack frame and it does not work when the debugger is stopped on an exception.

13.9. Viewing the Stack

Whenever the debug program is paused at a breakpoint, at an exception, or during 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. If there are PEP 3134 chained exceptions, these are listed in the order that they occurred, above the final exception.

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 stack icons in the toolbar, the toolbar stack popup menu, and the stack selector popup menus at the top of 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 shown in the editor.

In Wing Pro, the current stack frame is also used to control evaluation context in the Debug Console and Watch tools.

To change the type of stack display, right-click on the **Call Stack** tool.

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

13.10. Viewing Debug Data

Wing Pro's debugger provides many ways to inspect your debug program's data:

1. The **Stack Data** tool displays values in locals and globals for the currently selected stack frame. The display includes an expandable tree of values, and array and textual views for individual values. See Stack Data View for details.
2. The **Modules** tool supports the same type of inspection for values in all loaded modules (as determined by **sys.modules**).
3. The **Watch** tool can watch specific values from either of the above views. Right-click on values to watch them by symbolic name or object reference. See Watching Values for details.
4. The **Watch** tool can also watch expressions typed into the tool.
5. Hovering the mouse cursor over a symbol in the editor displays the value of that symbol in a tooltip, if it is in defined in the current stack frame. See Viewing Data on the Editor for details.
6. Holding down **Shift-Space** shows tooltips containing the values of all visible names on the editor. See Viewing Data on the Editor for details.
7. The **Debug Console** can be used to interact with the current stack frame of the debug process, in order to inspect data with arbitrary Python code. See Interactive Debug Console for details.

Debug 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.

13.10.1. Stack Data Tool

The **Stack Data** tool can be used to view debug data for locals and globals. It contains a process, thread, and stack frame selection area, an expandable tree area for viewing data, and a details area for inspecting individual values as an array or in textual form.

Process, Thread, and Stack Frame Selector

The top part of the tool contains popup menus for selecting the current debug process, thread, and stack frame to focus on. The process selector is omitted in Wing 101 and Wing Personal, which do not support multi-process debugging. The thread selector is hidden unless there is more than one thread in the debug process.

This area also contains the Stack Data Options Menu.

Value Display

The value display area is shown below the stack selector area, and will contain the values for the currently selected process, thread, and stack frame. Each value or part of a value is shown as one line in the tree.

Simple values, such as strings and numbers, and values with a short string representation, are displayed in the **Value** column of the tree. Strings are always contained in **""** (double quotes). Any value outside of quotes is the **repr** of an instance, a number, or a Python constant such as **None** or **False**. Integers can be displayed as decimal, hexadecimal, or octal, as controlled by the **Debugger > Data Display > Integer Display Mode** preference.

Complex values, such as instances, lists, and dictionaries, will be shown in a short form containing type and (optionally) the memory address, for example **<dict 0x80ce388>**. These can expanded by clicking on the expansion indicator in the **Variable** column. The memory address uniquely identifies the instance. If you see the same address in two places, you are looking at two object references to the same instance. Memory addresses may be hidden by toggling **Show Memory Addresses** in the tool's **Options** menu.

If a complex value is short enough to be displayed in its entirety, the **<type address>** form is replaced with its value, for example **{'a': 'b'}** for a small dictionary. These values can still be expanded from the **Variable** column. The size threshold used for this is set with the **Debugger > Line Threshold** preference. If you want all values to be shown uniformly, set this preference to **0**.

Expanding Values

When a complex value is expanded, the position or name of each sub-value will be displayed in the **Variable** column, and the value of each sub-value (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 a value, the debugger will continue to present that entry expanded, even after you step further or restart the debug session. Expansion state is saved and reused in later debug sessions, until you quit Wing.

Selected values can be viewed as an array or text by right-clicking on the item and choosing **Show Value as Array** or **Show Value as Text**. The content of the detail area is updated when other items in the **Stack Data** tool are selected. See Array and Textual Data Views for details.

Data Handling Errors

Wing may fail to show some data values because they are too large or can't be inspected safely. These are indicated in the form **<huge type 0x803ca872>** or **<opaque 0x80ce784>** in the **Stack Data** display and cannot be expanded further.

Some values that are too large for display in the **Stack Data** tool may still be viewed as arrays by right-clicking on the value and selecting **Show Value as Array**. Arrays are loaded incrementally according to what is visible on screen, and thus are less subject to size thresholds.

In Wing Pro you may also be able to use the Debug Console to access large or opaque values (for example try typing **dir(varname)**), or enter expressions into the Watch tool.

For details, see Problems Handling Values.

13.10.1.1. Array, Data Frame, and Textual Data Views

The value details area of the **Stack Data** tool can display selected values as an array or in textual form. The details view area is shown and hidden with **Show Value Detail** in the **Stack Data** tool's **Options** menu. The position of the details view can be changed by checking or unchecking the **Show Detail to Side** item in the **Options** menu.

Array View

Values like Pandas DataFrames, numpy ndarrays, xarray.DataArrays, sqlite3 result sets, and Python lists, tuples, and dicts can be viewed as an array by right-clicking on the item and selecting **Show Value as Array**. The array viewer loads slices of data as needed for display, rather than loading the whole data value at once.

A filter area is provided for searching the data. Only rows that match the filters will be shown. The filters are applied on the server side, to limit the amount of data examined and transferred to the IDE.

Each filter can be a string to search for in any data column, or may specify the column to search in the form **colspec:text**. For example, **0:msg** searches for the string **msg** only in column zero. If column labels are shown, as they are for sqlite3 results and some numpy and Pandas data, the column label can be used instead of the column number. For example, **name:oli** will search the **name** column for the string **oli**.

If multiple space-separated filters are entered, they must all match a row for that row to be displayed.

Filtering options are accessed by clicking on the drop down arrow to the right of the filter enter area:

- **Case Sensitive** can be checked for case-sensitive searching, for both the search string and any column specifiers.
- **Text Search**, **Wildcard Search**, and **Regex Search** select the type of matching to use.
- **Search All Columns** and **Search Visible Columns** select whether your filters are applied only to the visible range of columns, or to all columns. The default is to filter only on visible columns since filtering on all columns can be very slow in large arrays.

The array view can also display array-like instances that implement **__len__** and **__getitem__** and dict-like instances that implement **keys** and **__getitem__** if the **Debugger > Introspection > Allow Calls in Introspection** preference is enabled. This should be used with caution because it invokes these user-defined methods in a way that may be untested, possibly leading to unexpected changes in runtime state, hanging, threading deadlocks, or crashing.

Textual View

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 details area by right-clicking on a value and selecting **Show Value as Text**.

This can be useful in some other cases as well, where the textual representation of a value is easier to read than the tree or array view.

13.10.1.2. Stack Data Options Menu

The **Stack Data** tool's **Options** menu contains the following display options:

Show/Hide Value Detail toggle display of the array or textual value detail area.

Show Detail to Side show the array or textual value detail area to the right of the main display, instead of below it.

Show _name Protected Variables shows or hides symbols with names starting with a single underscore (protected members).

Show __name Private Variables shows or hides symbols with names starting with double underscore (private members).

Show __name__ Special Variables shows or hides symbols with names starting and ending with double underscore (special members).

Show Integers as Decimal shows all integers in decimal (base 10) form.

Show Integers as Hex shows all integers in hexadecimal (base 16) form.

Show Integers as Octal shows all integers in octal (base 8) form.

Show Memory Addresses shows or hides memory addresses for instances.

Resolve Properties enables or disables displaying properties in **Stack Data**. This should be used with caution. See Advanced Data Display for details.

13.10.1.3. Stack Data Context Menu

Right-clicking on the **Stack Data** tool displays a popup menu with options for navigating data:

Show Value as Array show the selected value as an array in the value details area.

Show Value as Text shows the selected value as text in the value details area.

Hide Value Detail hides the value details area shown with the above menu items.

Expand More increases the expansion of the selected complex data value by one additional level. If many values are expanded, you may experience a delay before the operation completes.

Collapse More decreases the expansion of the selected complex data value by one level.

Watch by ... in Wing Pro adds a value to the **Watch** tool, to track it over time as described in Watching Values.

Force Reload -- This forces Wing 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.

13.10.1.4. Filtering Value Display

Values shown in the **Stack Data** tool that are not of interest during debugging may be omitted by type or name (for variables and dictionary keys) by setting the **Debugger > Data Filters > Omit Types** and **Debugger > Data Filters > Omit Names** preferences. By default, these omit display of classes, functions, methods, and some other types.

For **Omit Types**, if type(value).__name__ is found in the list then it is omitted from the display.

For **Omit Names**, if the variable name or dictionary key is found in the list then it is omitted from the display.

The **Debugger > Data Filters > Do Not Expand** preference can be used to tell the debugger to avoid all attempts at probing certain values, based on their data type. This is useful to avoid inspection of values that cause problems or crashing when handled by the debugger. For example, values defined in buggy extension modules may cause crashing of the debug process if 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.

13.10.1.5. Advanced Data Display

Wing handles debug data conservatively to avoid invoking code that might cause unexpected changes in debug program state, hanging, crashing, thread deadlocks, and other problems that can occur if the debugger exercises code in a way that it was not designed to handle. Some advanced options are provided on the **Debugger > Introspection** preferences page, to allow Wing to inspect data more deeply:

- **Resolve Properties** enables calling **fget()** on properties so that properties can be shown in the **Stack Data** tool. This is off by default since calling property methods may changed program state unexpectedly, cause threading deadlocks, and bring out bugs in properties code not seen during regular execution.
- **Allow Calls in Data Inspection** enables calling user-defined **__len__**, **__getitem__**, **__call__** and similar special methods during data inspection. By default, Wing only calls these if implemented in C code, as for Python's standard data structures.
- **Call Python __repr__ Methods** enables calling __repr__ even if it is implemented in Python. This is enabled by default, since it is usually safe, but may be disabled for cases where these calls cause problems. Known cases where this option must be disabled include SQL database implementations that include all of very large query results in the **repr**.
- **Inspect Base Classes** controls whether Wing will try to inspect base classes for class attributes. This is enabled by default, since it is usually safe, but may be disabled for cases where it causes problems. Known cases where this option must be disabled include **openerp** and **odoo**, since they crash on inspection of some base classes.

When any of these options cause errors in the debugger, Wing will try to continue inspection of other data values whenever possible and mark the offending values with **<error handling value>**. However, if the inspection causes the debug process to crash or deadlock, Wing will fail to identify which value caused the problem, and the debug session will end.

If you are having problems with the debug process crashing unexpectedly while paused in Wing's debugger, try disabling all of the above options and then reenabling those that you need one at a time.

More information can be obtained about failures caused by these options by enabling additional debugger logging with the **Debugger > Diagnostics > Debug Internals Log File** preference.

13.10.2. Viewing Data on the Editor

Wing can show debug data values in tooltips over the editor in one of two ways.

Hovering Over the Editor

Hovering the mouse over a symbol in the editor will show a tooltip with its value, if one is available in the current stack frame. If a selection is made, hovering will show the value of the entire selection.

By default, Wing only shows values for selected symbols and not for all selected expressions. To show the value of any expression, set the **Debugger > Hover Over Selection** preference to **All (Use with Caution!)**. As the name suggests, changing this preference can result in the unintended evaluation of expressions that change the debug program state or that invoke arbitrary functionality in the debug process.

Showing All Available Values

In Wing Pro, holding down **Shift-Space** will show the values of all visible symbols on the editor. The values are shown only once for each symbol, usually on the first occurrence of the symbol, and will be hidden as soon as the key binding is released.

For simple variable names (such as **myvar**), this will show the already-obtained value from locals and globals in the current stack frame. For dotted names (such as **self.myvar**), this will evaluate the value on demand, also in the current debug stack frame.

If Wing can't fit the value tips into the code, it will move them out of the way and point each to its value. Color coding is used to make it easier to distinguish nearby values.

13.10.3. Watching Values

Wing can watch debug data values, using a variety of techniques for tracking the value over time. Watching a value is initiated by right-clicking on a value in the **Stack Data**, **Modules**, or **Watch** tool and selecting one of the following ways to watch the value in the **Watch** tool:

> **Watch by Symbolic Path** uses 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 **myvar** and watch **myvar['foo']**, the watched symbolic path is **myvar.foo**. This can be applied to **myvar** whether it's a dictionary or an instance with attribute **foo**. The **Watch** tool continues to show any value for that slot of **myvar**, even if you delete **myvar** and recreate it, change its type, or move to another stack frame with a variable of the same name. In other words, the value is tracked

only by reevaluation of the symbolic path **myvar.foo** and is independent of the life of any particular object instance.

Watch by Direct Object Reference watches the selected value using its object reference. If you use this method to watch **myvar**, it tracks the contents of that instance, even if the symbol **myvar** goes out of scope or is reassigned a new value. The **Watch** tool continues to show the contents of the instance as long as it exists, until there are no more references to it in the debug process. In other words, the symbolic path to the value that was originally watched is irrelevant and only instance identity is used to track the value. This is useful for watching a particular instance as you step in the debugger, even if references to that instance go out of scope. Because it is meaningless to track immutable types like **None** this way, this option is disabled or enabled according to the value you select to watch.

Watch by Parent Reference and Slot combines the above methods by using the object reference to the parent of the selected data value and a symbolic representation of the slot within the parent in order to determine where to look for the watched value. For example, watching **myinstance.attrib** will store the object reference to the instance referenced by **myinstance** and the symbolic name **attrib**. The **Watch** tool displays the attribute **attrib** in the referenced object instance, as long as there are still references to that instance in the debug process. This means that reassignment of **myinstance** to another value does not alter what is displayed in the **Watch** tool. Only reassignment of the selected instance slot changes what is displayed.

Watch by Module Slot looks up a module by name in **sys.modules** and references the value within that module by symbolic path. Any change in the value, even across module reloads, is reflected in the **Watch** tool. This option is only available when clicking on values within a module, such as **sys.path** or **os.environ**.

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 an instance 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 because those do not survive the debug process.

As in the **Stack Data** tool, values in the **Watch** tool can be displayed as an array or in textual form in the value details area. This is done by right-clicking on a value and selecting **Show Value as Array** or **Show Value as Text**.

13.10.4. Evaluating Expressions

The **Watch** tool can also be used to view the value of arbitrary expressions in the context of the current debug stack frame. These may be entered by clicking on any cell in the **Watch** tool's tree and editing or entering the desired expression in the **Variable** column. Press **Enter** to complete the edit.

Since expressions are evaluated in the context of the current debug stack frame, this feature is available only if there is a paused debug process. For the same reason, the value of expressions may change as you move up and down the stack.

Some caution is required to avoid undesired side-effects in the debug process. In cases where evaluating an expression results in changing the value of local or global variables, your debug process will continue in that changed context and the updated values will be shown in Wing's debugger tools.

Only expressions that evaluate to a value may be entered. Other statements, like variable assignments and import statements are rejected with an error. Exceptions that occur during evaluation or an expression are not shown, and breakpoints are not reached. To execute other statement types or to debug problems with an expression, use the Debug Console.

13.10.5. Problems Handling Values

Wing's debugger tries to handle debug process data as gently as possible, in order to avoid entering into lengthy computations or triggering errors in the debug process. Even so, not all debug data can be shown on the display. This section describes each of the reasons why this may happen.

Huge Values

Wing may consider values too large to handle if it thinks that packaging the value for transfer to the IDE would hang up the debug process. These values are displayed in the form **<huge type 0x803ca872>** in the **Stack Data** tool.

Some values that are too large for display in the **Stack Data** tool may still be viewed as arrays by right-clicking on the value and selecting **Show Value as Array**. Arrays are loaded incrementally according to what is visible on screen, and thus are less subject to size thresholds.

An alternative available in Wing Pro for viewing large data values is to enter expressions into the Watch tool or Debug Console, in order to view parts of the data without transferring the whole value to the IDE.

The thresholds that are used to determine whether a value is too large to display may be set in the **Debug > Data Display > Huge List Threshold** and **Debug > Data Display > Huge String Threshold** preferences. The former controls how large **len(value)** may be and the latter controls how long a string may be. Setting these preferences higher may increase data transfer times and may require also increasing the **Debugger > Network > Network Timeout** preference to prevent timeouts.

Data Handling Errors

Wing may encounter errors during data handling because the inspection and packaging process may call special methods such as **__cmp__** and **__str__** in your code. If these methods have bugs in them, 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, the value is displayed as **<error handling value>**.

These errors are not reported in the **Exceptions** tool. However, extra output containing the exception being raised can be obtained by setting the **Debugger > Diagnostics > Debug Internals Log File** preference. Or, in Wing Pro, try inspecting the value with the Debug Console.

Options that can prevent some types of data handling errors are documented in Advanced Data Display.

Opaque Values

Wing may treat values as opaque if they cannot be converted into a form that can be displayed in the IDE. This happens only rarely for data types defined within C/C++ code, or if a value contains certain Python language internals. Opaque values are denoted in the form **<opaque 0x80ce784>** and cannot be expanded further. In Wing Pro you may be able to use the Debug Console to access them (for example try typing **dir(varname)**).

Value Timeouts

Wing may time out handling a value when packaging it hangs up the debug process. The debugger tries to avoid this by carefully probing a value's size before packing it up. In some cases, this does not work, causing the debugger to wait for the duration set by the **Debugger > Network > Network Timeout** preference and then displaying the value as **<network timeout during evaluate>**.

Managing Value Errors

Wing remembers all debug data handling errors that it encounters and stores them 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 **Force Reload** in the right-click context menu on the value.

To clear the list of all errors previously encountered, so that all values are reloaded, use **Clear Stored Value Errors** in the **Debug** menu. This operates only on the list of errors known for the current debug main entry point, if a debug session is active, or for the main entry point, if any, when no debug process is running.

To avoid reoccurrence of more severe data value handling errors after clearing stored value errors, see Filtering Value Display.

13.11. Debug Process I/O

For a debug process launched from Wing, I/O associated with **print()**, writing to **stdout** or **stderr**, calls to **input()**, or reads from **stdin**, always occurs in the **Debug I/O** tool, unless an external console has been configured as described in External I/O Consoles.

Debug processes launched outside of Wing, using **wingdbstub**, always do I/O through the environment from which they were launched, whether that's a console window, web server, or any other I/O environment.

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 even while waiting for keyboard input, and (2) if the debug process was launched from Wing, I/O is redirected over the network to the IDE.

Multiplexing I/O can cause problems in some code. See Debug Process I/O Multiplexing for details.

If multiple debug processes are active, Wing creates one output buffer for each process launched from the IDE and the process selected in the top left of the tool may be used to move between them.

When commands are typed in the Debug Console in Wing Pro, I/O is redirected temporarily to the **Debug Console** only during the time that the command is being processed.

Options

The following options are available in the **Options** menu in the **Debug I/O** tool:

Clear clears the contents of the current output buffer.

Close All Terminated unconditionally closes all output buffers for debug processes that have been terminated.

Wrap Lines causes long lines to be wrapped in the display.

Never Auto-Show prevents Wing from ever automatically showing the **Debug I/O** tool.

Always Auto-Show on Output causes Wing to automatically show the **Debug I/O** tool when any output is received from the debug process.

Auto-Show on First Output causes Wing to automatically show the **Debug I/O** tool only the first time output is received from a debug process.

Auto-Focus for Input causes Wing to show the Debug I/O tool and set focus into the I/O buffer whenever a debug process is waiting for keyboard input. This is disabled by default in Wing Personal and Wing Pro and disabling the **Debugger > Advanced > Use sys.stdin Wrapper** preference prevents this feature from working.

Retain History causes Wing to retain old output buffers, up to the number configured with the **Files > Max Recent Items** preference. When this is unchecked, only one buffer is retained. Old buffers are cleared automatically only when a new debug process is started, to avoid losing output for a related group of partially-running processes. Old buffers can be cleared unconditionally at any time with **Close All Terminated** from the **Debug I/O** tool's **Options menu**.

Show Child Processes enables including child processes in the process selector. Otherwise only the top-level parent processes are shown.

Configure External Console allows replacing Wing's builtin **Debug I/O** tool with an OS-appropriate console run in a separate window. See External I/O Consoles for details.

Configure Encoding allows setting the encoding used for I/O to debug processes. This must be set to match the expectation of the debug process.

Show Debug I/O Documentation displays this documentation page.

13.11.1. External I/O Consoles

In cases where a debug process launched from Wing requires specific characteristics provided by a full-featured terminal emulator or Windows console, or to better handle very large amounts of debug process output, you can redirect debug I/O to a new external window using the **Debugger > I/O > 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 code. Alternatively, enable the **Debugger > I/O > 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 and OS X it is possible to select which console applications will be tried for the external console by altering the **Debugger > I/O > External Consoles** preference.

On Windows, Wing always uses the standard DOS Console that comes with your version of Windows.

Environment Limitations

Depending on the terminal implementation used, environment variables set by Wing may not be inherited by the Python process that runs within the external console. This breaks virtualenv, Anaconda environments, and any other case where the configured environment is needed for code to be able to run.

An easy work-around for virtualenv is to selected the **Command Line** option for **Python Executable** in **Project Properties** or the launch configuration. Then enter the full path of the virtualenv's Python. This is the value of **sys.executable** (after **import sys**) in the desired virtualenv.

To work around this in other cases, create a launch script that sets up your environment and then starts Python with all arguments that were passed to the script. Then set this script as the **Command Line** in your **Python Executable** in **Project Properties** or your launch configuration.

For example on Windows:

```
@echo off
set MYENV=value
call C:\path\to\envsetup.bat
C:\path\to\python.exe %*
```

Or on macOS and Linux:

```
#!/usr/bin/env bash
export MYENV=value
```

```
. /path/to/envsetup.sh
/path/to/python "$@"
```

Both examples show setting **MYENV** within the script and calling an external environment setup script **envsetup**. Either may be used as a way to provide the environment to the invoked Python.

13.11.2. 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, and to redirect I/O over the connection to the IDE. This code mimics the environment found outside of the debugger, so any code that uses only Python I/O does not need to worry about this change.

There are however several cases that can affect users that bypass Python I/O by doing C-level I/O from within an extension module:

- C/C++ extension modules that use the C-level **stdin** or **stdout** will bypass Wing's debugger I/O environment. This means that output sent to C-level **stdout** will not be redirected to the IDE. Also, waiting on **stdin** in C or C++ code will make the debug process unresponsive to messages from the IDE, such as **Pause** and changes to breakpoints, until the debug process exits its wait state.
- Calling C-level **stdin** from multiple threads in a multi-threaded program may result in altered character read order when running under Wing's debugger.
- When debugging on Windows, 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.

Disabling I/O Multiplexing

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 unchecking the **Debugger > I/O > Use sys.stdin Wrapper** preference.
2. Turn on the **Debugger > I/O > Use External Console** preference. See External I/O Consoles for details.

Once that is done, I/O should work properly in the external console, but the debug process will remain unresponsive to **Pause** or breakpoint changes from Wing whenever it is waiting for input, either at the C/C++ or Python level.

Also, keyboard input invoked as a side effect of using the **Debug Console** in Wing Pro will happen through unmodified **stdin** instead of within the **Debug Console**, even though command output will still appear there.

13.12. Interactive Debug Console

The **Debug Console** is an interactive Python shell for evaluating and executing Python code in the current debug stack frame, while the debug process is paused. This is a powerful tool for debugging and trying out new code interactively.

The **Debug Console** shares most of the features of the Python Shell, including command history, ability to evaluate code from the editor, active ranges, auto-completion, goto-definition, and integration with the Source Assistant.

Writing New Code Interactively

The **Debug Console** can be used to write new code in the live runtime context in which it is intended to work. To do this, set a breakpoint where you plan to place the new code, debug until you reach that breakpoint, then work in the **Debug Console** to design and try out the code.

Conditional breakpoints are a natural companion for the **Debug Console** because they can be used to isolate the particular case for which a new feature is intended.

Active Ranges

Another way to work with the **Debug Console** is to mark an active range of code in the editor. This is done by selecting a range of lines in the editor and pressing the **Set Active Range** icon at the top right of the **Debug Console**.

Once a range is set, additional icons appear to execute or debug the active range, jump to the active range in the editor, or clear the active range.

The active range is marked in the editor and will adjust its position and extent as code is added or deleted.

13.12.1. 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, change current directory, or otherwise alter the run environment, your debug process will continue within that altered state as if those changes had been made during normal execution.

The Stack Data, Watch, and **Modules** tools update after each command executed in the **Debug Console**, in order to reflect any changes caused by those commands. Since you may not notice these changes, some caution is needed to avoid creating undesired side-effects in the running debug program. When in doubt, restart the debugger.

13.12.2. Debugging Code Recursively

Code executed in the **Debug Console** is run without debug by default, and any exceptions are simply printed to the tool's console. Wing can also debug code recursively, so that any breakpoints or exceptions reached from the **Debug Console** are reported in the debugger. This is enabled by clicking on the bug icon in the upper right of the tool, or by using the **Enable Debugging** item in the **Options** menu.

Debugging code from the **Debug Console** works the same way as described in Debugging Code in the Python Shell.

To interact with recursively debugged code, while the **Debug Console** prompt is busy, you can add additional **Debug Console** instances to the user interface by right clicking on tool tabs. Or, turn on **Enable Recursive Prompt** in the **Options** menu so a new prompt is shown whenever the debugger is paused or at a breakpoint, even if the **Debug Console**'s earlier prompt is still in the process of executing code.

As in the **Python Shell**, **Stop Debugging** and **Start/Continue** will return to the innermost prompt frame. **Stop Debugging** does this without debug but does not preemptively interrupt the current invocation. In cases where this is a problem, the debug process should be restarted instead.

13.12.3. Debug Console Options

The **Options** menu in the **Debug Console** provides the following:

Clear truncates previous text from the shell.

Save a Copy stores a copy of the shell's contents to a disk file.

Wrap Lines causes the shell to wrap long output lines in the display.

Pretty Print causes Wing to use Python's **pprint** module to format output.

Enable Auto-completion controls whether Wing will show the auto-completer in the **Debug Console**.

Filter History by Entered Prefix causes up/down arrow key traversal of history to match only items that start with the string between the prompt and the caret. If no string was typed before pressing the up arrow then all history items are traversed.

Evaluate Only Whole Lines causes Wing to round up the selection to the nearest line when evaluating selections, making it easier to select the desired range.

Enable Debugging controls whether code run in the **Debug Console** will be debugged recursively

Enable Recursive Prompt causes the **Debug Console** to present a new prompt when debugging, even if the previous prompt invocation has not completed because the debugger is paused or at a breakpoint or exception. Execution returns to the previous prompt when the debug process is continued.

The preference **Debugger > Shells > Show Editor on Exceptions in Shells** can be used to determine whether source code windows will be raised when exceptions occur in the **Debug Probe**.

13.12.4. Debug Console Limitations

Some code will work in unexpected ways in Wing's **Debug Console** due to how list comprehensions, generator expressions, and nested functions work in Python 3. This results in inability to evaluate some code when stopped at a breakpoint in a function or method.

Nested Function Scope

The most comonly noticed example is inability access variables in an enclosing scope when within a nested function. For example when the debugger is stopped on the line **return 1** in the following code, typing **self** in the **Debug Console** raises a **NameError**:

```
class C:

  def m(self):
    def nested():
      return 1
    nested()

c = C()
c.m()
```

This is a result of how Python's compiler binds variables from the nested scope into nested functions. If the variable is not used in the nested function then it will not be defined there at all.

There is no work-around for this problem, other than moving up to the enclosing stack frame in the debugger and inspecting the variable there instead.

List Comprehensions and Generators

List comprehensions and generator expressions suffer from a related problem when used in the **Debug Console**. For an example, try stopping on **print(foo)** in the following code:

```
def x():
  from string import capwords
  foo = ['one two', 'three four']
  print(foo)

x()
```

Now typing the following list comprehension in the **Debug Console** will raise a **NameError** indicating that **capwords** is not defined:

```
y = [capwords(x) for x in foo]
```

This is because in Python 3 the list comprehension is implemented internally as a nested function and Python's compiler plays tricks to bind the necessary variables from the enclosing scope into the nested function. Even though capwords is defined in locals() the compiler does not use that when creating the code object for the list comprehension. Instead, it references the symbol table of the enclosing function which in the case of the **Debug Console** is (unavoidably) not x().

Generator expressions have the same problem:

```
y = (capwords(x) for x in foo)
x = list(y)
```

And so do nested functions, if defined within the **Debug Console**:

```
def f():
  capwords('test me')
f()
```

A possible work-around to use in some cases is to first load the locals into globals by typing the following in the **Debug Console**:

```
globals().update(locals())
```

However, this drastically alters program state in ways that may be destructive even if the original contents of **globals()** is restored after the evaluation.

13.13. Multi-Process Debugging

Wing Pro's debugger can debug multiple processes at once, either processes launched separately from the IDE, or (optionally) sub-processes spawned by a parent process.

When multiple processes are running at once, Wing adds a process selector to the stack selection area at the top of the various debugging tools. This selector displays all the connected debug processes, arranged into an indented tree that indicates which processes are children of others. The selector annotates each process entry to show its process ID and whether or not it is paused or running.

Multi-process debugging is on by default but can be disabled with the **Debugger > Processes > Enable Multi-Process Debugging** preference. When disabled, only one debug process can connect at a time or be started from the IDE.

13.13.1. Debugging Child Processes

Sub-processes started with the Python **multiprocessing** module or with **os.fork()** may be debugged automatically, so that each child process appears as a separate debug process in Wing. This is disabled by default but can be enabled with the **Debugger > Processes > Debug Child Processes** preference or by setting **Debug/Execute > Debug Child Processes** in **Project Properties**.

Sub-processes started with **os.system()**, **CreateProcess** (on Windows), **os.exec()** (on Posix), or similar calls will not be debugged automatically because the OS completely replaces the parent process context and there is no way to keep a debug connection intact. However, it is still possible to debug processes launched in this way by manually initiating debug in the sub-process as described in Debugging Externally Launched Code.

Notice that processes started by **os.fork()** followed by **os.exec()** will be debugged only for the (usually brief) period of time between the **os.fork()** and **os.exec()** calls.

Debugging Child Processes Created with sys.executable

By default when debugging sub-processes is enabled, Wing replaces **sys.executable** to cover some of the common ways in which sub-processes may be launched, particularly on Windows. This can be disabled with the **Debugger > Processes > Replace sys.executable** preference.

Because the **multiprocessing** standard library module uses **sys.executable** to launch its children on Windows, this option must be enabled in order to debug children created by that module.

Wing replaces **sys.executable** at startup only. As a result, user code that alters the value (other than by calling **multiprocessing.forking.set_executable**) will break debugging of child processes that are launched with a command line that contains **sys.executable**.

When **sys.executable** replacement is enabled, code that invokes **sys.executable** to start a child process must also provide the environment variables starting with **WINGDB_** to the child process. Otherwise, the debugger cannot determine which Python to run or how to connect to the IDE and the child process will fail to start.

If child processes are created with **sys.executable** the code that starts the child processes will need to correctly handle spaces in the path within **sys.executable**. Otherwise, child processes will fail to launch if Wing is installed into a directory path that has spaces in it and child process debugging is enabled.

One way to work around cases where **sys.executable** replacement does not work is to manually initiating debug in the sub-process as described in Debugging Externally Launched Code.

Target Processes for Handles on Windows

Replacing **sys.executable** will cause problems on Windows if a parent process launches children with a command line that contains a **Handle** created specifically for its child process, for example by setting **hTargetProcessHandle** in a call to **DuplicateHandle**. In this case, the handle will be invalid in the child because replacing **sys.executable** creates an intervening process and the child runs as the grand-child instead.

If a **Handle** is instead set to be inheritable for all child processes, for example by setting **bInheritHandle** in a call to **DuplicateHandle**, then replacing **sys.executable** will work without any problems.

Other Notes and Limitations

When debugging child processes created with the **multiprocessing** module, Wing will stop on exceptions raised in child processes. Continuing debug from that point will pack up and return the exception to the parent process, as in normal operation. Exceptions in children can be ignored with the **Ignore this exception location** checkbox in the **Exceptions** tool.

Overriding the **_bootstrap** method of **multiprocessing.process.Process** (or **multiprocessing.process.BaseProcess** in Python 3.4+) in a custom process class will prevent Wing from stopping on exceptions in child processes unless the exception is propagated to the inherited method. A work-around for this would be to call **logging.exception** with any exception before sending it out to the parent process.

Some approaches to spawning child processes may result in the creation of intermediate processes that appear in Wing's process tree display. For example, using the **shell=True** option in

subprocess.Popen will do this on Linux. When setting **shell=False** you may need to change the command passed to **Popen** to a list rather than a string.

Debug overhead may reveal timing bugs not seen outside of the debugger. For example, a parent process may attempt to interact with a child process too quickly, causing problems only under the debugger. This is particularly likely on Windows, where there is an intermediate process created between the parent and child process.

13.13.2. Process Control

When multi-process debugging is enabled, Wing will allow creation of multiple processes from the **Debug** > **Processes** sub-menu. This menu also provides a way to continue, pause, restart, or terminate all debug processes at once.

Pressing the **Alt** key while clicking on the **Continue**, **Terminate**, or **Restart** toolbar icons also causes the operation to be applied to all applicable debug processes at once.

By default when a new process connects and reaches a breakpoint or exception, it is made into the current debug process only if there is no previously current and paused debug process, or if it is the first process that has stopped for the process group most recently launched from the IDE (this does not include processes that attach using wingdbstub unless they are in a process group started from the IDE). In other cases, Wing displays a message at the bottom of the IDE window indicating that a debug process has stopped but does not make it the current process.

This behavior can be changed using the **Debug > Processes > Switch to Stopped Processes** preference. Setting this preference to **Always Switch** may be confusing if many processes are reaching a stopping point at once. However, this is the only way to automatically switch to a debug process started with **wingdbstub** when another debug process is already active.

Wing also lets you control the maximum number of debug processes that may be attached to the IDE at once using the **Debugger > Processes > Maximum Process Count** preference.

Terminating Processes

When a debug process is terminated from Wing, the IDE will by default also terminate all other processes in the process group. This is appropriate behavior in many but not all cases. The **Debugger > Processes > Termination Model** preference provides several options for managing termination of debug processes in a multi-processing environment:

Leave Other Processes Running kills only the selected process and leaves all other processes running.

Kill Child Processes with Parent also kills all children, grand-children, and other processes spawned by the process that is being terminated. However, any parent, grand-parents, uncles, cousins, etc, of the terminated process are left running.

Kill Entire Process Group kills all processes in the group, including all parents, grand-parents, children, grand-children, uncles, cousins, etc. This is the default termination model.

Prompt for Action When a Process is Killed displays a dialog listing processes associated with the debug process that is being terminated and offers to kill selected processes, all children, or the entire process group.

Note that when a only subset of the processes in a procress group are killed, those remaining processes that expect to interact with one of the terminated processes may raise "broken pipe" or similar errors.

13.14. Debugging Multi-threaded Code

Wing's can debug multi-threaded code, as well as single-threaded code. When a debug process has multiple threads, a thread selector popup is added to the stack selector area at the top of the various debugger tools.

By default, Wing debugs all threads in a debug process, and will stop all threads immediately if a single thread stops. 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 that is shown in the toolbar during debugging offers 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

To avoid stopping all threads in the debugger, you must launch the debug process from outside Wing, import **wingdbstub** to initiate debug, and then 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

Note, however, that specifying a subset of threads to debug may cause problems in some code. For example, if a non-debugged thread starts running and does not return control to any other threads, then the debug process will cease to respond to the IDE. This is unavoidable since there is no way to preemptively force the debug-enabled threads to run again.

13.15. 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. These behaviors can be altered with the **Debugger > Exceptions** preference group, as described below.

Ignoring Exceptions

Individual exceptions can be ignored by checking the **Ignore this exception location** check box in the debugger's **Exceptions** tool and continuing debug.

This is useful in ignoring non-critical exceptions that are being raised by code that is not currently of interest, in order to be able to work on other problems.

Ignored exceptions are remembered in the project and may be cleared with **Clear Ignored Exceptions** in the **Debug** menu.

Ignored exceptions are still reported if they actually lead to program termination.

Exception Reporting Mode

The overall strategy for identifying and reporting exceptions is configured with the **Debugger > Exceptions > Report Exceptions** preference. The following choices are available:

- **When Printed** stops on exceptions at the time that they would have been printed out by the Python interpreter. This is the default.

  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.

  Note that 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** stops 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, acceptable runtime conditions. As a result, this option is usually only useful after already running close to code that requires further examination.

- **At Process Termination** makes 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 , this 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** by the debug process.

Reporting Logged Exceptions

The **Debugger > Exceptions > 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 **Debugger > Exceptions > Never Report** and **Debugger > Exceptions > 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 to uncover bugs that are being masked by overly broad exception handlers. However, this will not work if these are treated as normal expected exceptions. This is common enough that they are not included in Wing's default **Always Report** list.

13.16. Running Without Debug

We recommend using Wing's debugger whenever Python code is executed, since this is the most efficient way of catching and fixing any problems encountered by the code. In most cases, the debugger overhead is low enough that executing outside the debugger has no real benefits.

However, Python code may also be executed outside of the debugger with **Execute Current File** and **Execute Recent** in the **Debug** menu, or with **Execute Selected** after right-clicking on the Project tool. This uses the OS Commands tool to manage the process.

Advanced Debugging Topics

This chapter describes advanced debugging techniques, including debugging externally launched code, remote debugging, alternative methods for starting debug, and using Wing's debugger together with a debugger for C/C++ code.

See also the collection of How-Tos for tips on working with specific third party libraries and frameworks for Python.

14.1. Debugging Externally Launched Code

This section describes how to start debugging from a process that is not launched by Wing. Examples of code that must be launched externally include tasks running under a web server and embedded Python scripts running inside a larger application.

The following instructions can be used to start debugging in externally launched code that is running on the **same machine** as Wing:

1. Copy **wingdbstub.py** from the install directory listed in Wing's **About** box into the same directory as the code you want to debug. Make sure that **WINGHOME** inside **wingdbstub.py** is set to the full path of your Wing installation.
2. At the point where you want debugging to begin, insert the following source code: **import wingdbstub**. If you are debugging code in an embedded Python instance, see the notes in Debugging Embedded Python Code.
3. Make sure the **Debugger > Listening > Accept Debug Connections** preference is turned on, to allow connection from external processes.
4. Set any required breakpoints in your Python source code by clicking on the breakpoint margin to the left of the code in Wing, or with the breakpoint items in the **Debug** menu.
5. Initiate the debug program from outside Wing in a way that causes it to **import wingdbstub** and reach a breakpoint or exception. You should see the status indicator in the lower left of Wing's window change to yellow, red, or green, as described in Debugger Status. When a breakpoint is reached, Wing should come to the front and show the file where the debugger has stopped. If no breakpoint or exception is reached, the program will run to completion, or you can use the **Pause** command in the **Debug** menu.

If you run your debug process as a different user, and in some other cases, Wing will initially refuse the connection and ask you to accept a new security token. After accepting it, debugging again should succeed.

To preauthorize the debug connection, you can copy the file **wingdebugpw** from your Settings Directory into the same directory as your copy of **wingdbstub.py**.

If you have problems making this work, try setting the **kLogFile** variable in **wingdbstub.py** to log additional diagnostic information.

14.1.1. Debugging Externally Launched Remote Code

This section describes how to debug code launched on a remote host. These instructions are needed only if you cannot launch your code from Wing, for example if it runs under a web server or as an embedded script in a larger application.

The following instructions rely on Wing Pro's Remote Hosts feature to display and edit remote files. If you cannot use that feature for some reason, follow the instructions for Manually Configured Remote Debugging instead.

1. First set up a remote host configuration as described in Remote Hosts and create a project that sets the **Python Executable** in **Project Properties** to the remote host and includes your remote source code. Before continuing, check that you can open remote files in Wing's editor.
2. Copy **wingdbstub.py** from the directory where you installed the remote agent into the same directory as your debug program. By default this is **~/.wingpro7/remote-7.2.9.0/wingdbstub.py** where **~** is the remote user's home directory. This will vary if you changed the **Install Dir** under the **Advanced** tab in the remote host configuration. If another copy of **wingdbstub.py** is used, configure it set **WINGHOME** to the installation directory of the remote agent and **localhost:50050** for the Wing host and port.
3. At the point where you want debugging to begin, insert the following into your code: **import wingdbstub**. If you are debugging code in an embedded Python instance, see the notes in Debugging Embedded Python Code. If you are debugging code running as a different user than the one in your remote host configuration, see **Managing Permissions** below.
4. Make sure the Wing preference **Debugger > Listening > Accept Debug Connections** is turned on, to allow connection from external processes. Once this is enabled, Wing will start listening for connections on the remote host you configured in your project.
5. Set any required breakpoints in your Python source code.
6. Initiate the debug program from outside Wing in a way that causes it to **import wingdbstub** and reach a breakpoint or exception.

You should now see the status indicator in the lower left of the main Wing window change to yellow, red, or green, as described in Debugger Status. If no breakpoint or exception is reached, the program will run to completion, or you can use the **Pause** command in the **Debug** menu.

Managing Permissions

If your code is running as a different user than the one specified in your remote host configuration, as may be the case if running under Apache or another web server, then you will need to make some additional changes to make remote debugging work. For example, your remote host configuration may set **Host Name** to **devel@192.168.0.50** so the user that installs the remote agent is **devel** while the code is actually run by the user **apache**.

In this case you must change the disk permissions on the **Install Dir** from which you copied **wingdbstub.py** so it can be read by the user that runs your debug process. The best way to do this is to create a group that includes both users and use that group for the directory, for example with **chgrp -R groupname dirname**.

Then change your copy of **wingdbstub.py** by replacing **~** with the full path to the home directory of the user in the remote host configuration. This is needed because **~** will expand to a different directory if the code is run as a different user.

You may also want to change the permissions on the debugger security token file **wingdebugpw** so that both users can read it, for example with **chmod 640 wingdebugpw**. The default for this file is to allow only the owner to read it. If this isn't done, Wing will generate a different debugger security token on the remote host and will initially reject your debug connection and prompt for you to accept the new security token. Once that is done, future debug connections will be accepted.

Changing Remote Debug Port

Remote debugging is implemented by listening locally and establishing a reverse SSH tunnel to the remote host configured in your project.

By default Wing listens on port **50050** on the remote host. Note that this is different than the default port used to listen on the local host, which is **50005**, in order to prevent the remote agent from interfering with a local copy of Wing, when both are in use.

If this conflicts with another service on the remote host, or if there are multiple remote debug connections to a single host, you will need to change this port number to be unique for each developer. To do this, edit the **Debug Port** property under the **Advanced** tab of your remote host configuration and track this change in **kWingHostPort** in your copy of **wingdbstub.py** on the remote host.

You can verify that Wing is listening on the remote host and inspect the port number being used by hovering your mouse over the bug icon in the lower left of Wing's window.

Debugging on Multiple Remote Hosts

Wing listens locally and on the remote host specified in **Python Executable** in **Project Properties**. To listen on multiple hosts at once, use separate projects and multiple instances of Wing. You can open additional instances of Wing by adding **--new** to the command line.

Diagnosing Problems

If you have problems making this work, try setting the **kLogFile** variable in **wingdbstub.py** to log diagnostic information.

14.1.2. Externally Launched Process Behavior

This section describes what happens if **wingdbstub** cannot attach to Wing, and how termination of remote debug works.

Failure to Attach to IDE

Whenever the debugger cannot contact Wing during **import wingdbstub**, for example if the IDE is not running or can't be reached, then the debug program will be run without debug. This allows debug-enabled web tasks and other programs to work normally when Wing is not present.

You can force the debug process to exit in this case by setting the **kExitOnFailure** flag in **wingdbstub.py**.

In Wing Pro, it is possible attach to processes that import **wingdbstub** but start without debug. See Attaching for details.

Enabling Process Termination

By default, Wing recognizes externally launched processes and disables process termination for them. The **Debugger > Listening > Kill Externally Launched Processes** preference can be set to enable Wing to terminate also externally launched processes.

Avoiding Connection Timeout

Some environments may preemptively close the debug connection from the outside if there is no activity over some period of time. To prevent this from happening, set the **Debugger > Advanced > Connection Keep-Alive** preference to the number of seconds between keep-alive messages.

14.1.3. Debugging Embedded Python Code

Python is designed so it can be embedded into larger applications as a scripting language, as a way to write high-level code that controls the functionality of that application. This is common, for example, in applications designed for 2D and 3D animation, compositing, and rendering, and some game development software. Examples include Blender, Autodesk Maya, NUKE, and Source Filmmaker.

When Python code is run by an embedded interpreter, you may need some extra configuration to make debugging work properly. What is needed depends on how the host application embeds and invokes Python.

Single Python Instance

If the host application is simply creating a single Python instance and reusing it for all script invocations, setting **kEmbedded=1** in **wingdbstub.py** will usually be all that is needed, in addition to adding **import wingdbstub** to your code.

This tells the debugger that complete exit of the debug code does not indicate that Python has exited as well, so that the debug connection can remain intact between script invocations.

Custom Python Thread States

Some host applications manually create or alter the Python thread states that is used for each script invocation. This may disable the debugger and/or disconnect the debug process from the IDE.

To solve this, invoke **Ensure()** in the debugger API, to reset the debugger for each script invocation:

```
import wingdbstub
wingdbstub.Ensure()
```

This tells the debugger to ensure that the debug tracer is properly installed and that the debug process is connected to the IDE, as needed in this particular application.

Multiple Python Instances

In other cases where the host application uses an entirely different Python instance for each invocation, you will need to arrange that the Debugger API function **ProgramQuit()** is called before each instance of Python is destroyed.

In this case, you should leave **kEmbedded=0** in **wingdbstub.py**. The debugger will disconnect and reconnect for each script invocation, as if they were separate debug processes.

You may also need to unset the environment variable **WINGDB_ACTIVE** before importing **wingdbstub**, if this is left in place by the host application. When this is present it will prevent **wingdbstub** from initiating debug in the second or later Python instance because the debugger will think that debugging is already active.

14.1.4. Configuring wingdbstub

In some cases you may need to alter other preset configuration values at the start of **wingdbstub.py**. These values completely replace the corresponding values set in Project Properties File Properties, and Launch Configurations in the IDE. Those are used only when the debug process is launched from Wing.

The following options are available:

- Set **kWingDebugDisabled=1** to disable the debugger entirely. This is equivalent to setting the **WINGDB_DISABLED** environment variable before starting debug.
- Set **kWingHostPort** to specify where Wing is listening for connections from externally launched debug processes, so the debugger can connect to it when it starts. This is equivalent to setting the **WINGDB_HOSTPORT** environment variable before starting debug. The default value is **localhost:50005**.

   Note that hostname will still be **localhost** if you are debugging over an SSH tunnel, as will be the case if you are using a remote host configuration. The SSH tunnel takes care of listening on **localhost** and then tunnels the connection to the host where the IDE is running.

   See Manually Configured Remote Debugging for details on changing this value.
- Set **KLogFile** to write a diagnostic log of debugger activity to a file. Usually, you should set this only at the request of Wingware Technical Support. This is equivalent to setting the **WINGDB_LOGFILE** environment variable before starting debug (use a value of **-** to turn off logging to file even if **kLogFile** is set).

   When setting this value to a file name, the file will be created if it does not exist. Similarly named files are created if multiple processes are being debugged, one for each process.

   Use **<stdout>** or **<stderr>** to write to Python's **sys.stdout** or **sys.stderr**. Note that using **<stderr>** will cause problems on Windows if the debug process is not running in a console.
- Set **kEmbedded** to **1** when debugging embedded scripts, so the debug connection will be maintained across script invocations, rather than closing the debug connection when the script finishes. This is equivalent to setting the environment variable **WINGDB_EMBEDDED**. See Debugging Embedded Python Code for details.

- Set **kAttachPort** to define the default port at which the debug process will listen for requests to attach. This is available in Wing Pro only and is equivalent to setting the **WINGDB_ATTACHPORT** environment variable before starting debug.

  If this value is less than **0**, the debug process does not listen for attach requests. Otherwise, the debugger listens on this port whenever the debug process is running without being connected to the IDE, as might occur if it initially fails to connect or if the IDE detaches from the process.

  See Attaching and Detaching for details.

- Set **kSecurityToken** to the security token used to authenticate with the IDE before the debug connection is accepted. This is the value in the **wingdebugpw** file (the portion after the **:**) in the Settings Directory for the user that is running the IDE. When this value is **None** the security token is located used **kPWFilePath** and **kPWFileName** as described below.

- Set **kPWFilePath** and **kPWFileName** tell the debugger where to find the security token file required for a debug connection to the IDE to succeed. This is equivalent to setting the environment variables **WINGDB_PWFILEPATH** and **WINGDB_PWFILENAME** before starting debug.

  kPWFilePath should be a Python list of strings containing directory names 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 the Settings Directory for the user that is running the debug process.

  kPWFileName sets the file name to use for the security token. The default is **wingdebugpw**.

- Set **WINGHOME** to the Wing installation directory (or the name of Wing's **.app** folder on OS X) so that **wingdbstub.py** can find the debugger. This is equivalent to setting the environment variable **WINGHOME** before starting debug.

  For Windows and Linux, and for copies of **wingdbstub.py** in a remote agent installation, **WINGHOME** will usually be set automatically during installation. The value may need to be set on OS X, if Wing was installed from the **.zip** installer on Windows or the **.tar** installer on Linux, if running Wing from sources, or if configuring remote debug manually.

Setting any of the above-described environment variable equivalents will override any value given in the **wingdbstub.py** file.

14.1.5. Debugger API

The debugger API controls debugging more closely from your Python code. It is used to control threaded debugging, and to develop support for debugging embedded scripting or other custom environments.

High-Level API

To use the high-level API, you must first configure and **import wingdbstub** as described in Debugging Externally Launched Code for code running on the same host as the IDE, or Debugging Remotely Launched Code if you are debugging code running on another host:

- **wingdbstub.Ensure(require_connection=1, require_debugger=1)** ensures that 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

The low-level API can be used through **sys._wing_debugger** (after **import sys**) in debug processes launched from the IDE or those using **wingdbstub**. In the latter case, the same API is available on **wingdbstub.debugger**:

- **SetDebugThreadIdents(threads={}, default_policy=1)** can be used in multi-threaded code to tell the debugger which threads to debug. Set **threads** to a dictionary that maps from thread id, as obtained from **thread.get_ident()**, or **thread_id** in the **PyThreadState**, to one of the following values: **0** to run the thread without debug, or **1** to debug the thread and immediately stop it if any thread stops. Set **default_policy** to the action to take when a thread is not found in the thread map.
- **Break()** pauses the free-running debug program on the current line, as if at a breakpoint.
- **SuspendDebug()** disables debugging, in order to temporarily avoid debug overhead. This leaves the connection to the IDE intact so that resuming is faster.
- **ResumeDebug()** resumes debugging if it has been called as often as **SuspendDebug()**.

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
```

- **StopDebug()** stops debugging completely and disconnects from Wing. The debug program continues executing in non-debug mode and must be restarted to start debugging again.
- **StartDebug(stophere=0, connect=1)** starts debugging, optionally connecting back to the IDE and/or stopping immediately afterwards. This does not work after **StopDebug()** has been called.
- **ProgramQuit()** may need to be called before the debug program is exited if **kEmbedded** was set to **1** in **wingdbstub.py**. This makes sure the debug connection to the IDE is closed cleanly. See Debugging Embedded Python Code for details on when this is needed.

14.2. Manually Configured Remote Debugging

> ### *Note*
>
> **Consider Easier Alternatives**
>
> This section describes the complex process of manually configuring remote debugging with **wingdbstub**. These instructions are needed only if you cannot use the Remote Hosts feature. In most cases, you will want to follow the much simpler instructions in Debugging Externally Launched Remote Code instead.

Another alternative to consider before getting started is installing Wing on the remote host and using remote display of the IDE via Remote Desktop (Windows), Screen Sharing (OS X), or X Windows (Linux/Unix).

Configuration Steps

1. First set up Wing to successfully accept connections from another process within the same machine, as described in section Debugging Externally Launched Code.
2. Optionally, alter the **Debugger > Listening > Server Host** preference to the name or IP address of the network interface on which the IDE listens for debug connections. The default **All Valid Interfaces** indicates that the IDE should listen on all the network interfaces found on the host.
3. Optionally, alter the preference **Debugger > Listening > Server Port** to the TCP/IP port on which the IDE should listen for debug connections. This value only needs to be changed if multiple copies of Wing are running on the same host.
4. Configure any firewall on the system that Wing is running on to accept a connection on the server port from the system that the debug process will run on, or set up an SSH tunnel as described in Manually Configuring SSH Tunneling.
5. Install Wing's debugger on the machine on which you plan to run your debug program, using one of the methods described in Manually Installing the Debugger.
6. Transfer copies of all your debug code so that the source files are available on the host where Wing will be running and at least the ***.pyc** files are available on the remote 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.

   You will need to use Samba, rsync, sftp, NFS, or some other file sharing 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, Wing can automatically discover the mapping if you add all your source files to your project. See File Location Maps for details.
7. On your remote host, copy **wingdbstub.py** out of the debugger installation and into the same directory as your source files and then add **import wingdbstub** to your Python source, as

described in Debugging Externally Launched Code. You will need to set **WINGHOME** in your copy of **wingdbstub.py** to match the location where you unstalled the debugger in step (5).

8. In **wingdbstub.py** on your remote host, set **kWingHostPort**. The host in this value must be the IP address of the machine where Wing is running. The port must match the port configured with the **Debugger > Listening > Server Port** preference on the host where Wing is running. If you set up an SSH tunnel in step (4) the host will be **127.0.0.1** and the port will depend on the SSH tunnel that was created.

9. Restart Wing and try running your program on the remote host. You should see the Wing debugger status icon change to indicate that a debug process has attached.

Example

For an example configuration, see Manually Configured Remote Debugging Example.

Diagnosing Problems

If you have problems making this work, try setting the **kLogFile** variable in **wingdbstub.py** to log additional diagnostic information.

14.2.1. Manually Configuring SSH Tunneling

If you are manually configuring remote debugging *without* Wing Pro's Remote Hosts feature, you may find that firewalls get in the way of making a direct connection between the remote host and Wing running locally. The best way around this is to establish an SSH tunnel that forwards network traffic from the remote host to the local host. This also encrypts all your debugger traffic in a secure way.

Doing this does require a working SSH server, but most remote hosts will already have that running. You will also need to set up remote login using SSH first, and in most case add your SSH key to the list of allowed keys on the remote host, so that SSH can login without any password.

Setting up SSH to a remote host is described in detail in SSH Setup Details.

Once that is done, SSH tunneling can be configured as described below.

Tunneling with OpenSSH

When Wing is running on OS X or Linux, or if you have OpenSSH on Windows provided by cygwin or Git Bash, tunneling can be done as follows from the machine that is running Wing (not the remote host):

```
ssh -N -R 50005:localhost:50005 username@remotehost
```

You'll need to replace **username@remotehost** with the login name and ip address of the remote host.

The **-N** option causes ssh to set up the tunnel but not run any command on the remote host.

The **-R** option sets up a reverse tunnel, which is needed since the debug process initiates the connection back to the IDE. The argument following it indicates that port **50005** should be tunneled from the remote host to **locahost**.

Optionally, an **-f** option could be added just after **ssh** to cause **ssh** to run in the background. Without this option, you can use **Ctrl-C** to terminate the tunnel. With it, you'll need to use **ps** and **kill** to manage the process.

If you also want a login shell on the remote host, use this form instead:

```
ssh -R 50005:localhost:50005 username@remotehost bash
```

SSH Tunneling with PuTTY

When Wing is running on Windows and you don't have OpenSSH available, **PuTTY** can be used instead to configure an SSH tunnel. This is done on the **Connections > SSH > Tunnels** page in **PuTTY** configuration: Set **Source port** to **50005**, **Destination** to **localhost:50005**, and select the **Remote** radio button, then press the **Add** button. Once this is done the tunnel will be established whenever PuTTY is connected to the remote host.

Using Different Port Numbers

The above assumes the default configuration, where Wing is listening for connections on port **50005**. If for some reason you can't use port **50005** as the debug port on either machine, this can be changed on the remote host with **kHostPort** in **wingdbstub.py** or with the **WINGDB_HOSTPORT** environment variable. To change the port the IDE is listening on, use the **Debugger > Listening > Server Port** preference and or **Debug Server Port** in **Project Properties** in Wing.

If this is done, you will need to replace the port numbers in the SSH tunnel invocation in the following form:

```
ssh -N -R <remote_port>:localhost:<ide_port> username@remotehost
```

<remote_port> is the port specified in **kHostPort** or with **WINGDB_HOSTPORT** environment variable on the remote host, and **<ide_port>** is the port set in Wing's preferences or **Project Properties**.

On Windows using PuTTY, the **Source port** is the port set with **kHostPort** or **WINGDB_HOSTPORT** on the remote host, and the port in the **Destination** is the port Wing is configured to listen on.

14.2.2. File Location Maps

If you are manually configuring remote debugging without using Wing Pro's Remote Hosts feature, and the full path to your source code is not the same on both hosts, then you need to take steps to tell Wing how to determine which local files match those on a remote host.

The easiest way to do this is to add all your source code to the project in Wing. This lets Wing discover all your files, so it can automatically build a file mapping using hashes on their contents of the files. If this works for you, no other configuration is necessary.

How it Works

Wing uses an SHA1 hash on the first 2MB of every source file that it finds in the project or through static analysis of all imports in your code. This is matched up to hashes obtained from the debug process to establish file identity, and a location map is built up automatically by looking at which directories appear to match on the local and remote side.

If there are multiple identical local files that match a remote file, Wing will notify you and then pick one arbitrarily. This can usually be fixed by removing the unwanted copies of source files from your project and restarting the debug process.

You can turn off Wing's automatic file matching by unchecking the **Debugger > Network > Use Digests To Identify Files** preference and then specifying a file location map manually, as described in the next two sections.

14.2.2.1. Manually Configured File Location Maps

If you are manually configuring remote debugging without using Wing Pro's Remote Hosts feature, and the full path to your source code is not the same on both hosts, and the automated file identification system described in the previous section won't work for your case, then you will need to create a mapping that tells Wing where it can find your source files on each host. This is done with the **Debugger > Network > Location Map** preference, which lists corresponding local and remote directory locations for each remote host's IP address.

Each host IP address in the location map is paired with one or more **(remote_prefix, local_prefix)** pairs. The **remote_prefix** is the full path on the remote hosts's file system using the file naming conventions for the remote host. The **local_prefix** is the full path of a local directory, using **/** forward slash as the separator regardless of which OS Wing is running on (except when specifying UNC style paths on Windows, in which case **** backslash is used).

The best way to understand this is to look at the Manually Configured Location Map Examples.

SSH Tunnels

When using an SSH tunnel, the IP address entered into the **Location Map** preference is the IP address of the host the IDE is running on, since the IDE thinks the connection is coming from the local host. This is often **127.0.0.1** but on Windows it may instead be the IP address for the host. This depends on the peer ip that is reported on the IDE side for connections opened through the pipe.

Details and Limitations

If multiple matches are found for a given remote file, Wing uses the most specific match, with the longest remote directory specification. Matches that point to existing local files are preferred over non-existing ones, even if the match is more general.

When running Wing on Windows, UNC formatted file names such as **\machine\path\to\file** may be used. 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. As a result, source file names are always resolved to their actual full path location.

Trouble-shooting

When in doubt, an easy way to determine the correct file path to use is to place **assert 0** into a file and refer to the traceback shown in the **Exceptions** tool in Wing when the file is debugged via **wingdbstub**. This can be used to set up the location map correctly, assuming you know the local location of the file.

14.2.2.2. Manually Configured File Location Map Examples

The best way to understand location maps, used for low-level manual configuration of remote debugging, is to inspect a few examples.

Defaults Explained

The default value for the **Debugger > Network > Location Map** preference contains one entry for **127.0.0.1** where the mapping is set to **Same as localhost**. This treats the full paths to files on both the remote host and local host as identical.

Two Linux or OS X Hosts

In this example Wing is running on **desktop1** and debugging some code on **server1** with IP address **192.168.1.1**. Files located in **/home/apache/cgi** on **server1** are the same files seen in **/svr1/home/apache/cgi** on **desktop1** because the entire file system on **server1** is being shared via NFS and mounted on **desktop1** under **/svr1**.

To support this example, the following would be added to the location map preference:

```
Remote IP 192.168.1.1 - Remote: /home/apache/cgi, Local: /svr1/home/apache/cgi
```

To enter this change in Wing's preferences, you would add **192.168.1.1** as a new **Remote IP Address**, select **Specify Mapping**, and enter a single mapping with **Remote** set to **/home/apache/cgi** and **Local** set to **/svr1/home/apache/cgi**.

Two Hosts Using an SSH Tunnel

When using an SSH tunnel, the IP address to which you add a mapping is always **127.0.0.1** because the tunnel forwards traffic in such a way that the IDE sees the connection as coming from the local host. The remote and local file paths given are the same as for the other examples given here. For the previous example it would be:

```
Remote IP 127.0.0.1 - Remote: /home/apache/cgi, Local: /svr1/home/apache/cgi
```

IDE on Linux or OS X with Debug Process on Windows

If you are debugging between two hosts of different type, the native path name format is used for the **Remote** specification and forward slashes are always used for the **Local** specification.

For example, the following entry would be used when running Wing on a Linux or OS X host and the debug process on a Windows host with ip address **192.168.1.1**, where the Linux or OS X directory **/home/myuser** is being shared via Samba to the Windows machine and mapped to the **e:** drive:

```
Remote IP 192.168.1.1 - Remote: e:\src, Local: /home/myuser/src
```

IDE on Windows with Debug Process on Linux/Unix

In this example, Wing is running on Windows and the debug process is on a Linux or OS X remote host with IP address **192.168.1.1**. As in the previous example, the Linux or OS X directory **/home/myuser** is being shared via Samba to the Windows machine and mapped to the **e:** drive:

```
Remote IP 192.168.1.1 - Remote: /home/myuser/src, Local: e:/src
```

Note the use of forward slashes in the the **Local** specification even though the file is on a Windows machine.

Two Windows Hosts

In this example, Wing is running on Windows and the debug process on another Windows machine with IP address **192.168.1.1**. The host where Wing is running has mapped the entire remote host's **c:** drive to **e:**:

```
Remote IP 192.168.1.1 - Remote: c:\src, Local: e:/src
```

Two Windows Hosts using UNC Share

This example is the same as the previous one, except that the UNC style path is used for the host where Wing is running:

```
Remote IP 192.168.1.1 - Remote: c:\src, Local: \\server\share\dir
```

Notice that backslashes are used in the **Local** specification when entering UNC style paths.

Windows and cygwin

In this example Wing runs on a Windows machine that also has cygwin installed. The cygwin files at **/c/src/test** are the same as the directory **c:\srctest** on the Windows side:

```
Remote IP 127.0.0.1 - Remote: /c/src/test, Local: c:/srctest
```

Notice that the IP address is 127.0.0.1 since cygwin runs on the same machine as Windows.

macOS Host and Raspberry Pi accessed via SSH Tunnel

In this example, Wing is running on a macOS host that is connected to a Raspberry Pi through an SSH tunnel. The files in **/home/pi/** on the Raspberry Pi match those in **/Users/pitest/src/** on the machine where Wing is running:

```
Remote IP 127.0.0.1 - Remote: /home/pi, Local: /Users/pitest/src
```

Notice that because of the use of an SSH tunnel, the remote IP address is reported as 127.0.0.1 and not the IP address of the Raspberry Pi.

14.2.3. Manually Configured Remote Debugging Example

> *Note*
>
> This example is for manually configured remote debugging only. It is not relevant for users of Wing Pro's Remote Hosts feature.

Here is a simple example that enables debugging a process running on a Linux host with IP address **192.168.1.200**, using Wing running on a Windows host with IP address **192.168.1.210**.

Configuring the Connection

On the Windows host, the following preferences must be specified:

- The **Debugger > Listening > Accept Debug Connections** preference should be enabled
- The **Debugger > Listening > Server Host** preference should be set to **All Interfaces** (this is the default)
- The **Debugger > Listening > Server Port** preference should be set to 50005 (this is the default)

On the Linux host, 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 host and see the debug connection establish on the Windows host.

File Sharing and Location Map

Next you will need to set up file sharing between the two machines (for example, with Samba) and then establish a location map in your Wing preferences on the Windows host.

After file sharing has been set up, you can add all your source files to your project, to allow Wing to automatically discover the locations of files on the local and remote host without any other configuration. See File Location Maps for details.

14.2.4. Manually Installing the Debugger

When manually configuring remote debugging with using Wing Pro's Remote Hosts feature, Wing's debugger must be installed on the remote host. To do that, you can either install Wing on that host, or download the appropriate debugger package from https://wingware.com/downloads/wing-pro/ and unpack it on the remote host.

Compiling from Source

On OSes for which there is no debugger package, choose the closest match and then recompile the debugger core from source code. This option is only available to Wing Pro customers, and requires signing a non-disclosure agreement. The compilation instructions are located in **build-files/README.DBG-SRC.txt** inside the debugger source distribution.

14.3. Using wingdb to Initiate Debug

Debug can be started on the command line by running **wingdb** (or **wingdb.exe** on Windows) from the top level of the Wing installation. These are invoked like the Python command line, after setting some environment variables that tell Wing which Python installation to use and how to connect to the IDE.

Minimal Configuration

First make sure that Wing is listening for debug connections by clicking on the bug icon in the lower left and enabling **Accept Debug Connections**.

Next set the environment variable **WINGDB_PYTHON** to the full path to the python or python.exe to use. This is needed only if you do not want to use the default **python**.

Now you can start debugging by running **wingdb** (or **wingdb.exe**) as if it were Python. Debugging should start and the process should connect back to Wing.

For example on Windows:

```
set WINGDB_PYTHON=C:\Python37\python.exe
C:\Program Files (x86)\Wing Pro 7.2\wingdb.exe myscript.py arg1 arg2
```

On Linux:

```
export WINGDB_PYTHON=python3.7
/usr/lib/wingpro7/wingdb myscript.py arg1 arg2
```

On OS X:

```
export WINGDB_PYTHON=python3.7
/Applications/WingPro.app/Contents/Resources/wingdb myscript.py arg1 arg2
```

Advanced Configuration Options

Other environment variables that control the debugger's behavior include:

WINGDB_PYARGS provides any arguments to send to Python itself. Do not use this for arguments sent to your Python code. Those are specified on the command line instead.

WINGDB_STEPINTO is set to 0 or 1 to indicate whether to stop on the first line of code (default=0)

WINGDB_WAIT_ON_EXIT controls whether the debug process should wait on exit for further interaction with the debugger (default=don't wait)

WINGDB_ENV_FILE causes the debugger to load environment from this file and then exec **sys.executable** in the environment, rather than running in the inherited environment. The environment file contains a sequence of byte strings, each separated by a **'\0'** byte. The 1st of every pair is a key and the 2nd is the value. (default=run in inherited environment)

WINGDB_HOSTPORT is the host:port where the IDE is running, if different than the default of **localhost:50005**. The host can be either a host name or an IP address and the port is the one shown when the mouse is hovered over the bug icon in the lower left of Wing's main window. We *strongly* recommend using Wing Pro's Remote Hosts feature instead. Otherwise, you'll also need most of the tedious manual configuration described in Manually Configured Remote Debugging.

WINGDB_SECURITYTOKEN can contain the security token to use for authentication with the IDE. If not specified, the default is to read the token from the **wingdebugpw** file in the user settings directory (the value used is the portion after the **:**).

WINGDB_WINGHOME is the Wing installation directory (default=compute based on location of the wingdb or wingdb.exe file)

WINGDB_USERSETTINGS is used only to find the debugger implementation if provided by an update made while running Wing with a non-default Settings Directory, as specified using the **--settings** command line argument (default=use the standard location for the directory)

WINGDB_LOGFILE is the full path to a diagnostic log file. Set this only at the request of Wingware Technical Support. It will slow down the debugger (default=no logging)

WINGDB_LOGVERYVERBOSE controls whether to print extremely verbose low-level diagnostic logging. Set this only at the request of Wingware Technical Support. It will *drastically* slow down debugging (default=off)

The following optional envs are only used to support Python 2.5; in Python 2.6+ set PYTHONIOENCODING instead:

WINGDB_STDOUT_ENCODING sets the encoding to use for stdout

WINGDB_STDIN_ENCODING sets the encoding to use for stdin

14.4. Attaching and Detaching

Debug processes normally connect to Wing automatically during startup. However, Wing can also attach to debug processes that are not already connected with the IDE. There are two cases where this is useful:

(1) When an externally launched process that uses **wingdbstub** (as described in section Debugging Externally Launched Code) cannot reach the IDE at 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 > Processes** sub-menu.

Detaching

Detach from Process in the **Debug > Processes** sub-menu detaches from the current debug process. **Detach from All Processes** detaches from all currently connected debug processes.

Whenever a process is detached, it continues to run outside of the debugger, without stopping at any breakpoints or exceptions. If a process is paused in the debugger when it is detached, the process will start running again immediately after the IDE disconnects.

Attaching

Attach to Process in the **Debug > Processes** sub-menu displays a dialog that contains known processes that were previously attached to Wing, and any additional host/port pairs given with the **Debugger > Network > Common Attach Hosts** preference. You may also type in a host/port value here (see **Identifying Processes** below).

Once you are attached to a process, it continues running until it reaches a breakpoint, unhandled exception, or **Pause** is used.

Identifying Processes

When debugging externally launched code in Wing Pro, the **kAttachPort** constant in **wingdbstub.py** sets the port on which the debug process will listen for attach requests from Wing.

If there are multiple concurrent processes and the specified port is in use then a random port number will be used instead. This port number will be communicated to the IDE if the debug process succeeds in connecting to it at startup, so the process can be listed in the **Attach to Process** dialog. Otherwise, you must use a unique value for the **kAttachPort** for each process.

Access Control

Wing creates a security token that is used to control who can attach to debug processes. As long as your debug process is running as the same user and on the same host as the IDE, attach and detach should work without any additional configuration.

If you run your debug process as a different user, or on a different machine than the IDE, Wing will initially refuse the connection and ask you to accept the security token from the other account or host. After accepting it, attaching again should succeed.

To preauthorize the debug connection, you can copy the file **wingdebugpw** from the Settings Directory where Wing is running into the Settings Directory for the other user or host, or into the same directory as **wingdbstub.py** if you are using that to initiate debug.

14.5. Debugging C/C++ and Python Together

Wing's debugger is for Python code only and doesn't itself handle stepping into C/C++. However, you can use Visual Studio, gdb, or another debugger concurrently, in order to debug Python and C/C++ at the same time.

The easiest way to do this is to launch the debug process from Wing, note the process ID shown when hovering the mouse over the bug icon in the lower left of Wing's window, and then attach the C/C++ debugger to that process.

Alternatively, it is also possible to launch the debug process with the C/C++ debugger and then initiate debug as described in Debugging Externally Launched Code.

To debug the C/C++ code you need to be running with a copy of Python compiled from sources with debug symbols. Note that Wing's debugger will be unavailable whenever the C/C++ debugger is paused.

14.5.1. Debugging Extension Modules on Linux/Unix

The first step in debugging C/C++ modules with gdb is to make sure that you are using a version of Python that was compiled with debug symbols. To do this, you need a source distribution of Python and you need to configure the distribution as described in the accompanying **README.rst** file.

In most cases, this can be done as follows: (1) Type **./configure**, (2) type **make OPT=-g**, and (3) type **make**. Once the build is complete you can optionally install it with **make install** or just run Python in place without installing it.

When this is complete, compile your extension module against that version of Python.

Starting Debug

In order to run code both within Wing's Python debugger and gdb, launch your debug process from Wing first, then note the process ID shown in the tooltip that appears when you hover the mouse over the debug icon in the lower left of Wing's main window.

Next, start gdb and type **attach <pid>** where **<pid>** is replaced with the process ID reported by Wing. This will pause the process as it attaches, which gives you a chance to set breakpoints. When you're ready to continue the process, type **c** in gdb.

You are now debugging both at the Python and C/C++ level. You should be able to pause, step, and view data in Wing whenever gdb is not paused. When gdb is paused, Wing's debugger cannot be used until the process is continued at the gdb level.

Tips and Tricks

- **Misc/gdbinit** in the Python source distribution contains useful macros for inspecting Python code from gdb. For example, **pystack** will print the Python stack, **pylocals** will print the Python locals, and **pyframe** prints the current Python stack frame. To use it, copy it into your **~/.gdbinit**.
- The following works to view Python data in **PyObject * obj**:

```
(gdb) p PyObject_Print (obj, stderr, 0)
```

- Breakpoints in a shared library cannot be set until after the shared library is loaded. **^C^C** can be used to interrupt the debug process, set breakpoints, and then continue.
- If **LD_LIBRARY_PATH** or other environment is not set as expected, check whether it is set in **.cshrc**. This file is read each time gdb runs so may overwrite your value. To work around this, set **LD_LIBRARY_PATH** in **.profile** instead. This file is read only once at login time.

See Debugging with Gdb for more information.

14.6. Debugging Non-Python Mainloops

The debug process connects to the IDE using a TCP/IP socket which is serviced from the debug tracer. Since Python only calls the tracer when Python byte codes are being executed, the debug process may become unresponsive if it spends long periods of time in non-Python code, such as in a C or C++ event loop. In this case, messages from Wing, such as **Pause** or changes to breakpoints, will be ignored by the debug process until some Python code is run again.

This is rarely an issue in practice since most code calls Python code periodically, and Wing's debugger contains hooks that entirely avoid the problem in PyQt, Gtk, Tkinter, wxPython, and Zope.

In the rare cases where the problem does occur, simple work-arounds include: (1) Schedule some Python code to run periodically as an idle task or timeout, or (2) just don't try to **Pause** or change breakpoints while the debug process is busy.

An alternative is to write a plug-in that services the debugger's sockets even when no Python code is being called, as described below.

Writing Non-Python Mainloop Support

Wing provides an API for adding the hooks necessary to ensure that the debugger's network sockets are serviced at all times. In order to use this, you must be able to register the debugger's socket in your environment's mainloop, or cause your mainloop to call **select()** on the socket periodically and invoke a provided callback when there is activity on the socket.

Mainloop hooks are written as separate modules that are placed into **src/debug/tserver** in your Wing installation directory (on OS X, this is inside **Contents/Resources** in Wing's **.app** folder). This directory contains several examples that can be used as a starting point.

To add your own non-Python mainloop support, you must:

1. Copy one of the source examples, such as **_gtkhooks.py**, to a file name **_xxxxhooks.py** where **xxxx** is the name of your non-Python mainloop environment.
2. Determine the names of indicator modules Wing can used to identify that this mainloop environment is being loaded and set **kIndicatorModuleName**.
3. Implement the **_Setup()**, **RegisterSocket()**, and **UnregisterSocket()** methods. Do not alter any code from the examples except the code within the methods. The name of the classes and constants at the top level of the file must remain the same.

4. Add the name of your module, minus the **'.py'** to the list **kSupportedMainloops** in **_extensions.py**

Don't hesitate to contact support@wingware.com if you need help.

14.7. Debugging Code with XGrab* Calls

Under X11, Wing does not attempt to break **XGrabPointer** or **XGrabKey** and similar resource grabs when your debug process pauses. This means that X may be unresponsive to the keyboard or mouse or both in some debugging cases.

Here are some tips for working around this problem:

(1) Most Linux systems offer some way to break through X11 pointer and keyboard grabs.

For example, **X.org** installations define a key symbol that releases all pointer and keyboard grabs. You can map a key sequence to it with xdotool as in the following example:

```
xdotool ctrl+alt+n XF86Ungrab
```

(2) Some toolkits have an option to disable resource grabs specifically to avoid this problem during debugging. For example, PyQt has a command line option **-nograb** that prevents it from ever grabbing the keyboard or pointer. Adding this to the debug process command line solves the problem.

When this option is not available, another option is to move processing into a timer or idle task so it occurs after the grab has been released.

(3) If all else fails, you can log in remotely, use **ps** to find the debug process, and kill it with **kill** or **kill -9** . This will unlock your X display.

(4) Setting DISPLAY to send your debug process window to another X display avoids tying up Wing in this way. The remote display will release its grabs once you kill the debug process from the IDE.

14.8. Debugger Limitations

Note

If you are having problems getting the debugger to stop at breakpoints or to display source as you step through your code, always read the Trouble-shooting Failure to Debug section first.

This section documents all know limitations in the debugger implementation. Many of these 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 that some files are out of sync so this case should only occur if you ignore its warnings.

(3) There are several cases where Wing may fail to stop at breakpoints or exceptions, or may fail to find source files. 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 and OS X 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.

(4) For code that spends much of its time in C/C++ without calling Python at all, the debugger may not reliably stop at breakpoints added during a run session, and may not respond to **Pause** requests. See Debugging Non-Python Mainloops for details.

(5) You cannot use **pdb** or other debuggers together with Wing's debugger. The two debuggers conflict because they attempt to use the same debug tracer in the Python interpreter.

(6) 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.

(7) 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.

(8) 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 while waiting for keyboard input, I/O redirection to the **Debug Console** 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.

(9) Using partial path names in module **__file__** attributes 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**.
- 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()** or the C API, a relative filename or an incorrect filename are used for the filename argument, and **os.chdir()** is called before the code is executed.

(10) 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.

(11) 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.

(12) 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.

(13) When using Stackless Python, overriding **stackless.tasklet.__call__** without calling the Wing debugger's **__call__** will break the debugger.

Integrated Version Control

Wing Pro provides integrated support for Git, Mercurial, Bazaar, Subversion, CVS, and Perforce. This supports adding, moving, renaming, and removing files, status, log, commit, update, revert, diff, push/pull, and some other operations specific to each system.

These operations are accessed from menus in the menubar and tools in the **Tools** menu that Wing adds according to which version control systems are used for the directories and files that you have added to your project. The operations are also available by right-clicking on an editor or in the **Project** tool.

File operations are integrated with the **Project** tool's file management features, so that moving, renaming, or deleting files in the **Project** tool uses the appropriate version control operations.

When a VCS is active, Wing also adds **Compare to Repository** to right-click context menus. This kicks off Difference and Merge between the working version and the repository version it is based on.

15.1. Setting Up Version Control in Wing

Wing relies on being able to run the command line executable, such as **hg**, **git**, or **p4** for each version control system. These must be installed first, if you don't already have them.

You will also need to check out a repository, or add your files to a new repository, according to the instructions for the VCS that you are using. This must be done outside of Wing, since the version control integration is not designed to create repositories or initially check out files from a VCS.

Wing assumes you are using an external SSH key manager to authenticate version control operations, or that the version control commands are configured to display an authentication dialog. Wing does not store passwords, nor does it provide a way to enter them for each operation. Refer to the documentation for each version control system to set up the appropriate authentication method. If you've never set up SSH before, see also SSH Setup Details.

Activating Version Control in Wing

Once you have your files added to version control, you can set them up in Wing simply by adding directories and files to your Wing project, using the items in the **Project** menu. The relevant version control menus should appear in the menu bar.

Which VCSs will be considered for projects can be controlled in the **Version Control** preferences group, with the **Active** preference under each version control system's preferences page. This supports entirely disabling a version control system, enabling it only if used in the project, or setting it as always active, so its menu and tool will always be available.

Trouble-Shooting

To diagnose problems with the version control integration, enable **Show Console** in the VCS tool's **Options** menu. This adds a tab to the tool that displays the commands that are executed and their output.

In some cases you may need to point Wing to the executable for your VCS using the **Executable** preference on the VCS's page, under the **Version Control** preferences area. This should be set to the full path to the command line executable and **not** the executable for GUIs like **TortoiseHg**.

15.2. Version Control Tools

The version control tools for each active version control system can be shown by selecting them from the **Tools** menu or as a side effect of selecting operations from any of the version control menus.

When initially shown, the version control tool contains a **Project Status** view that shows the file status for 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.

Each operation invoked for a version control system displays an additional view within the version control tool. These views collect any parameters for the operation and display the result of the operation. Use the menu in the top of the version control tool to switch between operations or to return to the **Project Status** view. Clicking on the **X** icon closes the view for the current operation.

The **Options** menu can be used to access the version control preferences, documentation, and a console that displays the version control invocations that Wing is making.

15.3. Common Version Control Operations

Some operations are similar across all the supported version control systems. While there are some minor variations among these, the basic idea is the same and they perform within Wing in the same way as they would on the command line.

Commit 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 Mercurial, or it may be on a remote host in centralized systems such as Subversion and CVS.

The view 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.

Diff displays the differences between files on the local file system and files in the repository. The diff appears as a view in the revision control tool. Its right-click context menu may be used to copy the diff text, go to the source for a particular section of the diff, or re-run the diff command.

Status displays the status of files. 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 but can be shown by right-clicking on the tool and selecting **Show Unregistered**.

Log displays a list of all the revisions, with commit comments, for one or more files or directories.

Revert disposes of any local changes and reverts the local files to match the revision that they were based on in the repository.

Commit Project runs the commit operation against all the files in the project.

Project Status runs the status operation against all the files in the project.

Add marks a file or directory to be added to the repository with the next commit.

Remove requests that a file or directory be removed from the repository with the next commit.

15.4. Bazaar

Wing's integration for Bazaar requires that the **bzr** command line executable is installed separately from Wing. Please see https://bazaar-vcs.org/ for information about Bazaar. The **bzr** executable may either be in your path or set with the **Version Control > BZR > Bazaar Executable** preference.

The Bazaar integration in Wing adds the following commands to those documented in Common Version Control Operations:

Annotate shows the origin of each line in a file.

Merge Entire Branch runs **bzr merge <remote>** to merge the changes in a remote branch into the local branch.

Push Entire Branch runs **bzr push <remote>** to push changes in the local branch to the remote branch.

15.5. CVS

Wing's **CVS** integration 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 with the **Version Control > CVS > CVS Executable** preference.

The CVS integration works best if usernames and passwords are handled by an SSH agent. For details on this see SSH Setup Details and the CVS documentation.

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.

The CVS integration defines the following commands, in addition to those documented in Common Version Control Operations:

Update updates the local copy with changes from the repository.

Update Project updates all the directories in the project with changes from the repository.

15.6. Git

Wing's Git integration requires the **git** command line executable to be installed separately from Wing. Please see https://git-scm.com/ for information about Git. The **git** executable may either be in your path or set with the **Version Control > Git > Git Executable** preference.

The Git integration defines the following commands, in addition to those documented in Common Version Control Operations:

Blame/Praise shows the revision, author, and date for every line in a file.

List Branches lists all branches in the local repository.

Switch Branch switches to a different named branch with **git checkout <branch>**.

Fetch Repository Changes fetches changes from a remote repository with **git fetch <remote>**.

Pull Branch Changes pulls changes on a branch from a remote repository to the local repository with **git pull <remote> <branch>**.

Push Branch Changes pushes changes on a branch from the local repository to a remote repository using **git push <remote> <branch>**.

15.7. Mercurial

Wing's Mercurial integration requires the **hg** command line executable to be installed separately from Wing. Please see https://www.mercurial-scm.org/ for information about Mercurial. The **hg** executable may either be in your path or set with the **Version Control > Mercurial > Mercurial Executable** preference.

The Mercurial integration defines the following commands, in addition to those documented in Common Version Control Operations:

Resolve marks merge conflicts in a file to be resolved, by running **hg resolve -m**.

Annotate shows the revision number for every line in a file.

Pull Changes fetches changes from a remote repository to the local repository and optionally updates the working directory.

Update updates the entire working directory with changes from the local repository.

Merge merges changes in the local repository into the working directory.

Push Changes pushes changes in the local repository to a remote repository.

15.8. Perforce

Wing's Perforce integration 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 with the **Version Control > Perforce > Perforce Executable** preference.

Wing's **Perforce** integration is disabled by default and must be enabled with the **Version Control > Perforce > Active** preference.

Wing finds the Perforce working directory 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.

Perforce defines the following command, in addition to those documented in Common Version Control Operations:

Sync updates the client work space with changes from the depot.

Edit prepare files for editing and makes 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.

Sync Project updates all client work space directories project with changes from the depot.

Configuration Details

If the **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 the preference **Version Control > Perforce > Extra Global Arguments**.

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.

15.9. Subversion

Wing's Subversion integration 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 with the **Version Control > SVN > SVN Executable** preference.

The Subversion integration works best if usernames and passwords are handled by an SSH agent. For details on this see SSH Setup Details and the Subversion documentation.

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 **Version Control > SVN > Extra Global Arguments** preference.

Subversion defines the following commands, in addition to those documented in Common Version Control Operations:

Update updates the local copy with changes from the repository.

Resolved indicates 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 displays the the revision number and author for every line in a file.

Last Revision Diff shows the differences in the most recently committed change set for a file.

Update Project updates all the directories in the project with changes from the repository.

Source Code Analysis

Many of Wing's features rely on a powerful source code analysis engine that runs in the background as you work. This inspects all the Python code in your project, and all the code that it uses, as found through **import** statements.

The source code analysis engine inspects code using type inference, type annotations, and user-provided interface description files. It also makes use of live runtime state whenever available, by loading and inspecting extension modules, and by introspecting symbols in the context of an active debug process or the integrated **Python Shell**.

16.1. How Analysis Works

To analyze your source code, Wing uses the **Python Executable** and **Python Path** that you have specified in your Project Properties and any main entry point. This environment defines which modules are found by **import** statements and alters some aspects of type inference, according to Python version.

Show Analysis Stats in the **Source** menu displays the Python environment that is being used for source code analysis.

Note that this environment is used to analyze *all* files in your project, even if some of them use Launch Configurations or File Properties to set up a different Python environment for themselves. This is usually OK, but in some cases it may be better to set up a separate project for each Python environment.

Wing's source code analysis process can be summarized as follows:

- To resolve an **import** statement, Wing searches the **Python Path** and same directory for a matching importable module.
- If the module is Python code, Wing runs static analysis on the code to extract information from it.
- If the module is an extension module, Wing looks for a ***.pi** or ***.pyi** interface description file, as described later in this chapter.
- If the module cannot be inspected, Wing tries to import it in a separate process space, in order to analyze its contents.
- If a debug process is active, or when working in the Python Shell, Wing tries to read relevant type information from the live runtime state associated with the source code

The results of this analysis are cached on disk and recomputed only as necessary when the Python environment or code changes.

16.2. Helping Wing Analyze Code

There are a number of ways to assist Wing's source code analyzer in determining the type of values in difficult-to-inspect dynamic Python code, C/C++ extension modules, and other code that is resistant to analysis.

16.2.1. Setting the Correct Python Environment

The most common reason that Wing fails to provide useful source code analysis is failure to configure **Python Executable** and **Python Path** in Project Properties. This is important so that Wing knows which version of Python your code is designed for, and so it can find any modules that are not on Python's default **sys.path**.

In cases where code makes changes to **sys.path** at runtime, it may help to set the file where those changes are made as the main entry point. Wing tries to read **sys.path** changes and incorporate them into the Python environment used for source code analysis. If this fails, add the appropriate items to **Python Path** in **Project Properties**.

16.2.2. Using Live Runtime State

Running to a breakpoint is a great way to help Wing analyze code. This allows Wing to extract complete and correct type information from the live runtime state, as a supplement to the information found through static analysis. The auto-completer, Source Assistant, and other tools make use of this information when it is available.

This approach also has the advantage that new code can be tried out immediately in Wing Pro's Debug Console, in the context of the runtime environment for which it is being designed.

Working in the Python Shell also provides access to runtime type analysis.

16.2.3. Adding Type Hints

Wing can understand several different kinds of type hints added to Python code.

PEP484 and PEP 526 Type Annotations

Adding type hints in the styles standardized by PEP 484 (Python 3.5+) and PEP 526 (Python 3.6+) is another way to help Wing understand difficult-to-analyze code.

For example, the following indicates to Wing the argument and return types of the function **myFunction**:

```
from typing import Dict, List

def myFunction(arg1: str, arg2: Dict) -> List:
  return arg2.get(arg1, [])
```

The type of variables can be indicated by a comment that follows an assignment:

```
x = Something() # type: int
```

Or in Python 3.6+ the type can instead be specified inline:

```
x:int = Something()
```

The types that Wing can recognize include basic types like **str** and **int** and also the following from the **typing** module: **List**, **Tuple**, **Dict**, **Set**, **FrozenSet**, **Optional**, and **Union**.

Type Hinting with isinstance()

Another way to inform Wing of the type of a variable is to add an **isinstance** call to your code. For example **isinstance(obj, CMyClass)**. This is useful in older Python versions, or when combined with debug-only runtime type checking like **assert isinstance(obj, CMyClass)**.

In cases where doing this introduces a circular import or other problems, use a conditional:

```
if 0:
  import othermodule
  isinstance(obj, othermodule.CMyClass)
```

The source code analysis engine will still pick up on the type hint, even though it is never executed.

16.2.4. Defining Interface Files

Creating a ***.pyi** Python Interface file is another way to describe the contents of a module that Wing has trouble analyzing. This file is simply a Python skeleton with the appropriate structure, call signature, and return values to match the functions, attributes, classes, and methods defined in a module.

Wing reads the ***.pyi** and merges its contents with any information it obtained through direct inspection of the module. **.pyi** files can use PEP 484 (Python 3.5+) and PEP 526 (Python 3.6+) type annotations, regardless of whether Python 2 or Python 3 is being used.

Wing also supports reading interface files named ***.pi** but these cannot use PEP 484 or PEP 526 type annotations. The **.pi** extension was used in previous versions of Wing that predated the PEPs. It is still supported but should not be used for newly created interface files.

In somes cases, as for Python bindings for GUI and other toolkits, ***.pyi** or ***.pyi** files can be auto-generated from interface description files. The code that Wing uses to automatically generate ***.pi** files from extension modules is in **src/wingutils/generate_pi.py** in your Wing installation, and another example that is used to generate interface information for PyGTK is in **src/wingutils/pygtk_to_pi.py**.

Naming and Placing *.pyi Files

Wing expects the ***.pyi** file name to match the name of the module. For example, if the name referenced by **import** is **mymodule** then Wing looks for **mymodule.pyi**.

The most common place to put the ***.pyi** file is in the same directory as the ***.pyd**, ***.so**, or ***.py** module it is describing. ***.pyi** files that describe entire packages (directories containing **__init__.py**) should be placed in the package directory's parent directory.

If Wing cannot find the ***.pyi** file in the same directory as the module, it proceeds to search as follows, choosing the first matching ***.pyi** file:

1. In the path set with the **Source Analysis > Advanced > Interface File Path** preference.
2. In the **resources/builtin-pi-files** in the Wing installation. This is used to ship type overrides for Python's builtin types and standard library.

3. In **resources/package-pi-files**, which is used to ship some ***.pyi** files for commonly used third party packages.

For all of these, Wing inspects the path directory for a matching ***.pyi** file and treats any sub-directories as packages.

In cases where Wing cannot find a ***.pyi** at all for a C/C++ extension module, it will still attempt to load the extension 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 **Cache Directory** shown in Wing's **About** box. This file is regenerated only if the ***.pyd** or ***.so** for the loaded extension module changes.

Merging *.pyi Name Spaces

When Wing finds a ***.pyi** file, it merges the contents of the ***.pyi** file with any information found by analyzing or introspecting the module itself. The contents of the ***.pyi** file take precedence when symbols are defined in both places.

Creating *.pyi Variants by Python Version

In rare cases, you may need to create variants of your ***.pyi** files according to Python version. An example of this is in **resources/builtin-pi-files**, the directory used to ship type overrides for Python's builtin types and standard library.

Wing always looks first at the top level of an interface path directory for a matching ***.pyi** file. If this fails 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.

16.2.5. Helping Wing Analyze Cython Code

Wing works best with Cythonâ■■s pure Python mode. In this case, the source code is stored in **.py** files, and source analysis works the same as it does in all other Python files. Debugging also works when the **.py** file is executed directly rather than compiling it. See Pure Python Mode for details on using Cython this way.

Cython-compiled modules that don't use pure Python mode are inspected in the same way as extension modules, which means that some type information, including name and type of arguments to functions, is unavailable. In that case, ***.pyi** files may be used to improve Wing's analysis of the interface in the module, as described in Defining Interface Files.

Wing cannot analyze **.pyx** files directly and uses the simplified non-Python completion support when working within those files.

16.3. Analysis Disk Cache

The source code analyzer writes information about files it has examined into the **Cache Directory** that is listed in Wing's **About** box, accessed from the **Help** menu.

Wing does not perform well if the space available for this cache is smaller than the space needed for a single project's source analysis information. This can be solved by increasing the **Source Analysis > Max Cache Size** preference.

The analysis cache may be removed in its entirety by pressing **Clear Cache** next to the preference. Wing will reanalyze your code and recreate the cache as necessary.

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.

Remote Development

Wing Pro can work with Python code that is stored on a remote host, device, or container in the same way that you work with code stored locally. This includes editing, debugging, testing, searching, version control, running a Python shell, executing command lines, and project management.

Remote development is supported to OS X and Linux (Intel or ARM). A detailed list of supported remote host types is available in Supported Platforms. Wing Pro itself can be running on Windows, Linux, or OS X.

How it Works

Wing's remote development support works by installing a remote agent that carries out operations on the remote host. All communication to the remote host is over secure SSH tunnels, one to access the remote agent, and one for each debug process.
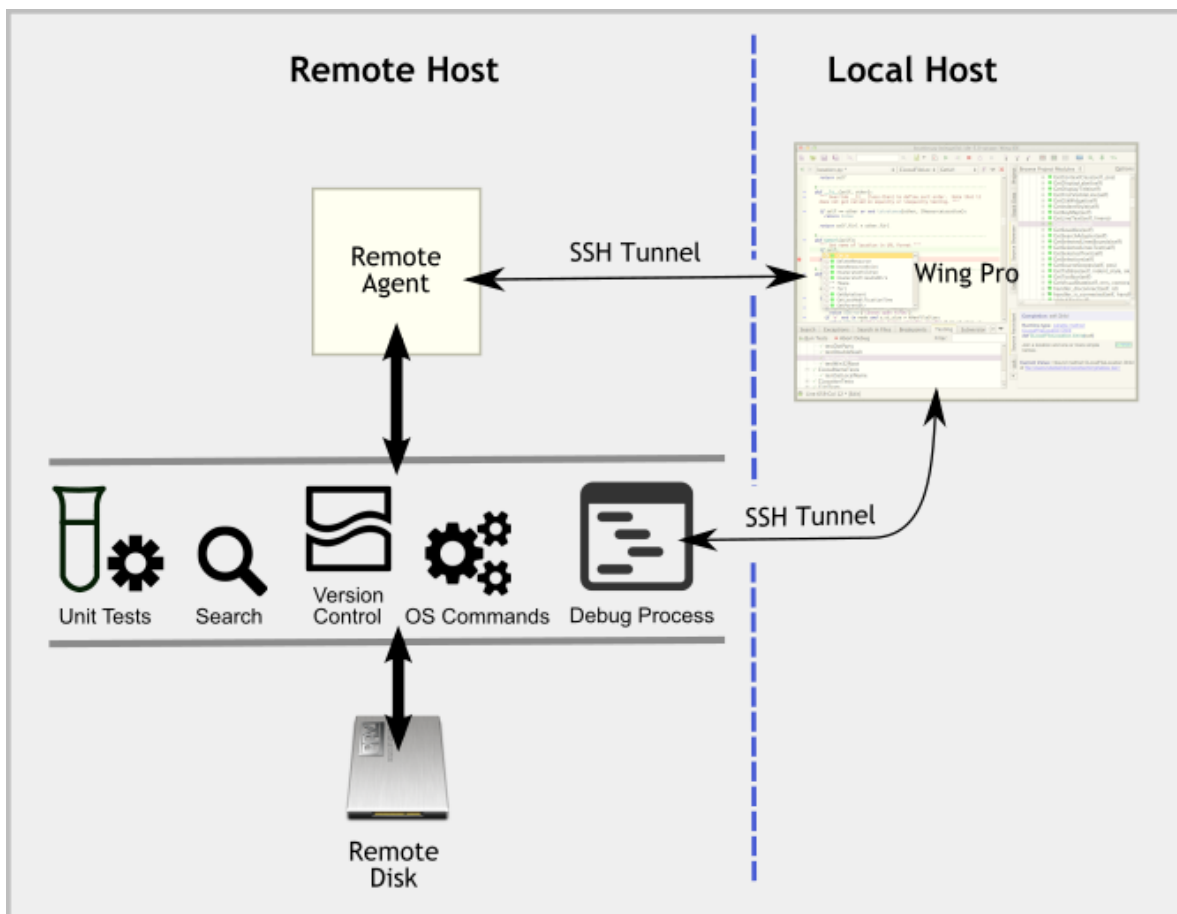
Files are stored on the remote host, and everything you do is run on the remote host, including running tests, debugging, executing files and command lines, searching, and issuing version control operations.

The remote agent replaces the need for setting up file sharing to the remote host, manually establishing SSH tunnels, defining file location maps, and other manual configuration steps required for remote debugging in Wing 5 and earlier.

If you have used **wingdbstub** for manually configured remote debugging in the past, you can continue to use that approach. Or you can switch to the new approach, which supports both launching your remote debug process directly from Wing or continuing to use wingdbstub through the remote agent if you need to launch your code from outside of the IDE.

If you prefer to store the master copy of your code on your local system, you can do this as well by setting up file sharing to the remote host using Samba, NFS, or other method. However, you will still use the remote agent to access the files on the remote system, rather than opening them directly from local disk into the IDE.

Configuration Overview

There are several steps in setting up remote development:

1) Setting up SSH and connecting to it from Wing using an SSH user agent, so that connections can be made without asking for a password. Many developers already have this set up. Wing just invokes the **ssh** or **plink** command line tools that you may already be using.

2) Defining a remote host configuration to tell Wing about the remote host and how to access it.

3) Setting up a remote project in much the same way as is done for local projects.

These steps are detailed in the next three sections.

17.1. Setting up SSH for Remote Development

To work with a remote host, you first need to set up secure SSH remote access outside of Wing Pro. You should configure this so that you can connect to the host without having to enter a password each time you connect. Instead, you want authentication to occur using an SSH key pair, and by entering your password once to load the key into an SSH user agent.

On Linux and OS X this is done with the standard OpenSSH tool suite that comes with the OS. On Windows we recommend using PuTTY. It may also work to use OpenSSH provided by Cygwin, Git Bash, or Windows 10 optional feature, but these still seem to be more prone to problems.

If you do not already have an SSH client set up, or you cannot log into the remote host without entering a password every time you connect, please refer to SSH Setup Details before going any further.

Accessing the SSH Agent From Wing

Once you have SSH working outside of Wing, and can connect to the remote host without entering a password, start Wing in an environment where it will be able to access your SSH keys via the SSH user agent. How this is done varies according to OS and which SSH implementation you are using:

Windows with PuTTY -- Start **pageant.exe** from the command line. Then right-click on the icon that appears in the lower right of your screen and select **Add Key**. You may need to redo this each time you log in. The private key file can also be passed to **pageant.exe** on the command line.

Windows with OpenSSH in Cygwin or Git Bash -- Start Cygwin and type **ssh-agent bash** on the command line and use **ssh-add** to add your key. Then type **set | grep SSH_**, copy the **SSH_AGENT_PID** and **SSH_AUTH_SOCK** lines, and paste them into the **Environment** in Wing's **Project Properties**. You will need to redo this each time you quit Cygwin since the contents of the environment variables will change. One way to avoid having to set these environments is described in **Connecting without SSH User Agent** below.

OS X -- Add your key to **Keychain Access** and optionally set usage restrictions for the key with **Get Info** from the **File** menu. Depending on how your key is configured in **Keychain Access** you may need to unlock your key again or run **ssh-add** on the command line each time you log in.

Linux -- Add your key with **ssh-add** on the command line. You need to rerun **ssh-add** each time you log in. If for some reason your Linux distribution does not run **ssh-agent** on its own, run **ssh-agent bash** followed by **ssh-add** and then launch Wing from that command line with **wing-7.2** so it inherits the necessary environment.

Specifying the OpenSSH or PuTTY Executables

Wing uses the following command line tools to implement secure access to remote hosts: **ssh** and **scp** (with OpenSSH) or **plink.exe** and **pscp.exe** (with PuTTY).

Wing looks for these tools on the **PATH** on the machine where it is running, and on Windows it also searches for **PuTTY** and Cygin-provided **ssh** (in that order) in common installation locations if it cannot find them on the **PATH**.

If Wing cannot find **ssh** or **plink.exe** you will need to add its directory to your **PATH** or set the **Network > SSH Executable** preference in Wing to the full path of the command. If this is set, Wing also tries to find **scp** (or **pscp.exe** on Windows with PuTTY) in the same directory as the specified executable.

Connecting without an SSH User Agent

Although we recommend against it from the standpoint of maintaining proper security, it is possible to use SSH without an SSH user agent. This is done by creating an SSH key without an encryption password and using it as the **Private Key** under the **Options** tab in your remote host configuration.

If you need your unencrypted SSH key also outside of Wing, you can instead configure OpenSSH or PuTTY to use it with a specific host:

With OpenSSH, the key can be specified for a host by adding it to **~/.ssh/config** as follows:

```
Host somehost
    User someuser
    IdentityFile ~/.ssh/someuser@somehost/id_rsa_no_pw
```

With PuTTY on Windows, configuration is done with **putty.exe** where you create a saved session and set the **private key file for authentication** in the **Connection > SSH > Auth** configuration area. Then save the session and **plink.exe** will use that unencrypted private key without prompting for a password.

17.2. Configuring Remote Hosts

Remote hosts are configured using **Remote Hosts** in the **Project** menu, to tell Wing about the remote host and how to connect to it. The following values may be specified in the three tabs of the remote host dialog:

Identifier (required) is the unique short name used to reference this remote host configuration. It is used in the URLs that reference resources on the remote host. If an existing remote host configuration's ID is changed, Wing will track that change in all the remote host references stored in the project. However, for shared remote host configurations, it's best not to change the identifier after it is used.

Host Name (required) is the remote host's name or IP address. The the host name may include the username, in the form **username@hostname** or **username@ipaddress**. This is needed if the user name on the remote host is different from the user on the local host. For containers hosted by Vagrant or similar tools, the host name must include **username@** even if the remote and local user names are the same. This tells Wing that an SSH tunnel is necessary even if the host name or IP address is for the local host. If this field is changed in an existing remote host configuration, Wing will try to find remote resources on the new host name.

Python Executable is the Python to use for running Wing's remote agent and for debugging or executing remotely. This can be left blank if Python can be found on the **PATH**. In this case, Wing first looks for **python3** and then falls back to using **python**. Otherwise, it can either be set to **Activated Env** to enter a command that activates a virtualenv or Anaconda environment on the remote host (so that **python** launches the correct Python), or it can be set to **Command Line** to specify the **python** to run. In the latter case, it should be the name of a Python that can be found on the **PATH**, the full path to the Python executable, or a path relative to the configured **Base Directory** (see below). When in doubt about the location of the Python you want to use, run it outside of Wing and execute **import sys; print(sys.executable)** to obtain the value to use. If your Python cannot be run without certain environment variables, such as **PYTHONHOME** or **PYTHONPATH**, you will need to set up a custom startup script as described in Specifying Environment for the Remote Python.

Base Directory is the directory on the remote host from which all file references are made, so that Wing will show only the relative path from the configured base directory. By default, it is the remote user's home directory. If this value is a partial path, it is interpreted to be relative to remote user's home directory. When this value is changed on an existing configuration, Wing will try to find resources relative to the new base directory.

Forward X11 enables X11 display from the remote host to the host where Wing is running. On OS X and Windows this requires installing and configuring an X11 server, such as XQuartz on OS X or MobaXTerm on Windows. With OpenSSH this uses **ForwardX11Trusted** style forwarding. For finer

control of authentication options, leave this option disabled in Wing and instead set options in your **.ssh/config** file. On Windows with PuTTY, this is done in the **SSH > Auth > X11** section of host configuration in **PuTTY**. On Windows with VNC, you may instead need to set **DISPLAY=:1** in the **Environment** in **Project Properties**.

SSH Port sets the port on which OpenSSH is running on the remote host. The default is port **22** or whatever port number is configured in **.ssh/config** or PuTTY's saved sessions.

Private Key specifies how Wing accessed the private key to use when connecting to the remote host. The default is to use the SSH user agent (**ssh-agent** for OpenSSH or **pageant** for PuTTY). The key file format must match the SSH implementation being used (usually **.rsa** or **.pem** for OpenSSH and **.ppk** for PuTTY). With OpenSSH on Linux or macOS, the key file must be set to be readable only by the user running Wing, for example with **chmod 600 mykey.pem**. Private key files used with this property cannot be encrypted. Thus, from a security standpoint, using an SSH user agent instead is usually preferable.

File Encoding is the default text encoding to use when opening or creating files on the remote host, if the file does not explicitly set the encoding.

I/O Encoding is the text encoding to use for I/O to and from processes started on the remote host by the debugger or **OS Commands** tool.

Install Dir is the full path to the installation location of Wing's remote agent on the remote host. Wing will automatically install and update the remote agent as needed. Using the default for this setting is recommended, since that will automatically add and remove remote agent installations according to which versions of Wing you are using.

Manage SSH Tunnel controls whether Wing manages SSH tunnels to allow the remote agent and debugger to connect from the remote host to the IDE. The default of **Auto-configured** establishes SSH tunnels only if the remote host is not the same as the local host. This should be disabled for container systems that automatically forward network traffic, such as Windows Subsystem for Linux (WSL), and it must be enabled when connecting to isolated containers that appear to be the same as localhost, like Vagrant. **Important:** When this option is disabled, network traffic between the IDE and the remote system is entirely unencrypted, both for the remote agent and the debugger. This option should only be disabled when working on the local host or if the underlying network is otherwise encrypted (for example, by a VPN or a manually configured encrypted tunnel).

Remote Agent Port is the TCP/IP port to use for the remote agent on the remote end of the SSH tunnel. When this is not specified, Wing uses a random port number determined on the IDE side of the connection. This usually works but there is no guarantee that the port will also be available on the remote end. When set, this property should be an unused unprivileged ephemeral port number (usually between **1025** and **65535** on Windows, **32768** and **61000** on Linux, and **49152** and **65535** elsewhere). When a fixed port is specified, Wing still uses a random port on the local end of connections, unless **Manage SSH Tunnel** is also disabled. In that case, the same port number is used at both ends of the connection, and this must match port mappings established by configuration made outside of Wing. This option must be set to **Use Random Port** when using **ControlMaster** in the OpenSSH configuration. Using a fixed port in that case may fail because the control master can prevent reusing the port when the remote agent is restarted.

Remote Debug Port is the first TCP/IP port to use for the debugger on the remote end of the SSH tunnel. By default, as for Remote Agent Port, a random port is used. When a value is specified, Wing uses only ports starting with the given port, up to however many ports are needed for active debug sessions and **Python Shells**. When a port is specified, Wing still uses a random port on the local end of connections, unless **Manage SSH Tunnel** is also disabled. In that case, the same port number is used at both ends of the connection, and this must match port mappings established by configuration made outside of Wing. This option must be set to **Use Random Port** when using **ControlMaster** in the OpenSSH configuration because that will hold onto previously used SSH tunnel ports indefinitely until the remote host is restarted.

2FA Card Selector is used with OpenSSH integrations that prompt for selection of a 2FA card at startup. The text entered is written to the SSH process at startup. It is stored as plain text in the remote host configuration so should not be used for passwords. This value is only used with OpenSSH and not PuTTY.

Installing and Running the Remote Agent

After a remote host is configured, Wing will try to connect to that host and install the remote agent if it is not already present. If installation of the remote agent fails, you will be presented with diagnostic output to send to support@wingware.com for help.

In rare cases you may need to install the remote agent manually as described in Manually Installing the Remote Agent. One such case can occur on Linux when **uname** reports a different bittedness than is being used by Python. For example, **uname** may report a 64-bit system but Python may be 32-bit.

Once installed, the remote agent is started or restarted as needed and will exit after a timeout period if it is unused. The remote agent allows Wing to search, inspect, read, and write files and directories, create or delete files, start debug or execution, run unit tests, invoke version control operations, run **Python Shell**, invoke commands in **OS Commands**, and perform other actions on the remote host to support the IDE's functionality. The necessary SSH tunnels for communication to the remote agent and to support debugging files remotely are also managed automatically.

You can find a log of the remote agent's activities in the file **remote-agent.log** within the Settings Directory on the remote host.

Shared Remote Hosts Configurations

Remote host configurations can either be stored in the project file or shared in the Settings Directory so they can be accessed from all projects. To make a remote host configuration shared, check the **Shared** box for that configuration in the remote host manager accessed from **Remote Hosts** in the **Project** menu.

In general, a shared remote host configuration should be used when the project file is stored on the remote host, and non-shared remote host configurations should be used when a project file is stored locally but accesses resources on a remote host.

17.3. Setting up Remote Projects

There are two ways to work with remote hosts: (1) a locally stored project file can reference remote resources, and (2) a project file stored on a remote host and opened remotely can transparently access resources on that remote host.

Local Project Files

For projects stored locally that need to access resources on another host, the **Python Executable** property in **Project Properties** is set to **Remote** to indicate that a project's Python resides on a remote host. The remote host configuration that is selected is typically an unshared configuration, so that it is stored in the project and will be accessible if the project is moved to another machine. Note, however, that remote host configurations may be specific to an individual machine's network environment, and may need to be edited on other hosts.

After **Python Executable** has been set, other properties that reference files, such as **Initial Directory** and **Python Path**, will be resolved on the remote host. The **Browse** buttons for those properties will browse the remote host, and paths will be stored as partial paths relative to the configured **Base Directory** or as full paths if located outside of the **Base Directory**. Paths on remote hosts are always expressed using forward slash **/** and will be converted as needed to the native separator on the remote host.

The selected remote host will also be used for adding files and directories to the project. When a URL for a remote file or directory is shown, it will be in the form **ssh://hostid/path/to/file.py** where **hostid** is one of the configured **Remote Host IDs**.

A locally stored project can include files and directories on multiple hosts, by adding several hosts and using **Add Existing File** and **Add Existing Directory** with each host.

Remote Project Files

Projects stored on a remote host are opened with **Open Remote Project** in the **Project** menu. This menu item is not shown unless you have already created a shared remote host configuration. Projects stored like this are normal Wing projects and may also be opened locally, if Wing can also be run on the remote host itself. In this case, **Python Executable** is simply set to **Default**, **Command Line**, or **Activated Env**, as if the project were stored locally. Wing resolves all the resources in the project file in a way that allows it to access them on the host where the project is stored.

If any remote host configurations are added to a remotely stored project, in order to access other remote hosts, those configurations must work on the host where the IDE is running.

Creating Project Files

To set up a new project that accesses a remote host, use **New Project** in the **Project** menu and specify **Connect to Remote Host (via SSH)** as the project type. This will ask for the same fields described in the previous section, for creation of a remote host configuration. If you have already created a configuration previously, use the **Recent Hosts** drop down to copy that configuration.

Projects created with the **New Project** dialog are saved locally, unless **Save Project on Remote Host** in the **Project** menu is used to store the project on the remote host. In this case, the remote host configuration must first be set to shared in the **Remote Hosts** dialog, so it is always accessible on the local machine and can be used to open the remote project.

A regularly created local project can also be moved to a remote host with **Save Project On Remote Host`** in the **Project** menu. This menu item is visible only if there is at least one shared remote host configuration. Saving the project in this way moves only the project file itself, and assumes that resources referenced by the project will also be available on the remote host, with the same relative paths from the project file.

17.4. Remote Development Features

Once you have your remote project set up, you should be able to edit, debug, test, and otherwise work with Wing in the same way as you in the local case.

Editing

Editing on a remote host is no different than editing on a local host, except that in some cases the contents of a file may take a bit longer to appear when it is first opened.

Debugging

Debugging also works the same way as for local files. Wing will initiate the debug connection automatically through its SSH tunnels to the remote host. File names will be shown in the form **hostid:filename** but otherwise debugging works the same way as on the local host.

To debug on several different remote hosts, use **Launch Configurations** in the **Project** menu to create debug configurations on each host. This is done in the same way as for **Project Properties**, by setting **Python Executable** under the **Python** tab to **Remote**. Then set up a **Named Entry Point** that pairs a file on that remote host with a launch configuration for the same remote host.

Whether you use the Project-wide settings or a launch configuration, the file you debug needs to be stored on the selected remote host. You cannot debug a file from one host on another host using this style of remote debug configuration.

When debugging on a remote host, the **Debugger > I/O > Use External Console** preference is ignored and I/O always appears in the **Debug I/O** tool. If a remote process needs to run in a different console, start it there and initiate debug from your code as described in Debugging Externally Launched Remote Code.

The **Debugger > Diagnostics** preferences are also not used when debugging on a remote host. The following environment variables can be used instead to collect debugger diagnostics. These should only be used at the request of Wingware Technical Support, and the resulting log file can be emailed along with your bug report to **support@wingware.com**:

WINGDB_LOGFILE can be used to set up a diagnostics log file when trouble-shooting problems with the debugger. The environment variable should be set to the full path of the log file on the remote host.

WINGDB_LOGVERYVERBOSE selects whether to print extremely verbose low-level logging. This is almost never needed and will drastically slow down debugging.

Debugging Externally Launched Code

If you need to start your debug processes from outside Wing, as for services running on a remote host, you can debug those processes by importing **wingdbstub**. When you install the remote agent, Wing writes a correctly configured copy of **wingdbstub.py** into the remote agent's installation directory. To use it, follow the instructions in Debugging Externally Launched Remote Code.

Python Shell

Once you have set **Python Executable** in **Project Properties** to a remote host, you can restart the **Python Shell** from its **Options** menu to launch a shell that is running on the remote host.

Testing Remotely

If remote files have been added to the **Testing** tool the unit tests can be run or debugged as if they are on the same host as the IDE.

Version Control

If remote files are checked into a version control system, Wing should identify this as it does for local files and include the appropriate tools in the **Tools** menu. Version control features work the same way for remote files as for local files. However, it may be necessary to configure version control for the remote host using the **VCS** tab in **Project Properties**.

Operations that access the version control repository (such as push and pull) may not work due to lack of access to the necessary SSH keys. There are two possible solutions for this:

1. If the remote VCS command tries to display a password collection dialog, you can turn on the **Forward X11** option in your remote host configuration, so that the dialog will appear on the machine where Wing is running. On Windows and OS X this requires installing an X11 server on the local machine.
2. You can forward the local host's SSH agent credentials to the remote host by adding **ForwardAgent yes** to your **.ssh/config** on the machine where Wing is running. It's best to limit this to the hosts that require it and you should do it only if you understand the security implications.

OS Commands

The **OS Commands** tool also supports working remotely with the **Hostname** property under the **Environment** tab of **Command Line** style commands. For **Python File** and **Named Entry Point** style OS Commands, the host name is inferred from the location of the file being executed.

17.5. Remote Agent User Settings

The remote agent uses the same default location for the Settings Directory that the IDE does. In some cases, such as on some embedded devices, this cannot be used because the file system is read-only. In this case, the remote agent will fall back on using a directory named **user-settings** inside of the **WINGHOME** specified in the remote host configuration. The **user-settings** directory will be created automatically after the remote agent has been installed.

17.6. Specifying Environment for the Remote Python

Wing uses any **Environment** you specify in **Project Properties** to execute, debug, or test your code. But this environment cannot be used when running the remote agent, since it is started in the environment provided by **ssh** or **plink.exe**.

As a result, if the Python installation on your remote host needs certain environment variables in order to run, it may fail to start when Wing attempts to run the remote agent with it.

To work around this, create a shell script that sets the necessary environment and starts up Python. For example, if your Python needs **PYTHONHOME** and **PYTHONPATH** to be set you might write something like this:

```
#!/bin/bash
export PYTHONHOME=/
export PYTHONPATH=/lib/python2.7
python "$@"
```

Then **chmod +x** the above script so it is executable and set the **Python Executable** in your remote host configuration to its full path.

17.7. Manually Installing the Remote Agent

If for some reason you cannot use Wing's automated installation of the remote agent, for example if Wing does not recognize the type of the remote system, you can install it manually as follows.

(1) Find the Debugger Package most closely matching your remote host at https://wingware.com/downloads/wing-pro/7.2.9.0/debugger, copy it to the remote host, and unpack it. The resulting directory can be renamed if desired.

You can unpack it with **tar xf wing-debugger-*** or **tar xjf wing-debugger-***. Note that using **tar xzf** does not work because the package is compressed with bzip2 and not gzip.

(2) Run **chmod +x wingdb** in the remote agent install directory to make that file executable

(3) Set **Installation Directory** under the **Advanced** tab in your remote host configuration to match the remote agent install location (this should be the full path of the directory that contains **remoteagent.py**)

If you plan to use **wingdbstub** to initiate debug from outside of Wing, as described in Debugging Externally Launched Remote Code you'll also need to:

(4) Copy **wingdbstub.py** from your Wing installation to the remote host and place it in the remote agent install directory (same directory as **remoteagent.py**)

(5) Set **WINGHOME** in **wingdbstub.py** to the full path of the remote agent install directory

(6) Set **kWingHostPort** in **wingdbstub.py** to **localhost:50050** (assuming default debug port settings)

Optionally, if want to preauthorize debug connections from the remote host:

(7) Copy the **wingdebugpw** file from the Settings Directory on the host where the IDE is running into the remote agent install directory on the remote system.

Once this is done, Wing should be able to probe and use the remote host from **Remote Hosts** in the **Project** menu.

Running on Unsupported OSes

These instructions should work on any system that has Python installed. If the remote host is not one that Wing fully supports, you will still be able to edit, search, and manage files, run unit tests, execute version control operations, and run OS commands.

However, debugging or running a remote Python Shell will not work on unsupported OSes unless you compile the debugger core yourself (requires signed NDA). Or contact us to request support for your device.

17.8. SSH Setup Details

This guide will help you set up secure password-less SSH access to remote hosts that you want to use with Wing Pro. If you already know how to set up password-less SSH access to a remote system, the process is the same for Wing and you can skip this section.

17.8.1. Working With OpenSSH

Use these detailed instructions to set up SSH access with OpenSSH from a host running Linux, OS X. This instructions also can be used on Windows using Cygwin, Git Bash, or Windows 10's native OpenSSH implementation. However, we still recommend using **PuTTY** since it seems to be prone to fewer problems.

The necessary tools for SSH access are already installed on Linux and OS X systems. They are also included in Cygwin on Windows if the **openssh** package is selected at installation time, and they come with Git Bash, which is actually a scaled down version of Cygwin. Newer versions of Windows 10 also make OpenSSH available as an optional feature that can be enabled as described in Enabling Windows 10 OpenSSH Client.

Generating an SSH Key Pair

On these systems many developers already have an SSH key generated and in use. If you do not already have one, you will need to generate one with **ssh-keygen** as follows on the system where you will be running Wing Pro. On Windows, these commands need to be executed in the Cygwin or Git Bash terminal and not the Windows Console:

```
mkdir ~/.ssh
chmod 700 ~/.ssh
ssh-keygen -t rsa
```

Use the default settings and enter a password for encrypting the private key. This will produce **~/.ssh/id_rsa** (private key file) and **~/.ssh/id_rsa.pub** (public key file).

Moving the SSH Public Key to the Remote Host

A copy of the public key needs to be transferred to the remote host you want to connect to and added to **~/.ssh/authorized_keys**. The following is one way to accomplish this:

```
ssh username@remotehost "mkdir .ssh; chmod 700 .ssh"
ssh username@remotehost "sed -i -e '$a\' .ssh/authorized_keys"
scp ~/.ssh/id_rsa.pub username@remotehost:.ssh/pub.tmp
ssh username@remotehost "cat .ssh/pub.tmp >> .ssh/authorized_keys; rm .ssh/pub.tmp"
```

The first line above is only needed if you do not already have the directory **~./ssh** on the remote system.

The second line is only needed if you already have **~.ssh/authorized_keys** on the remote system, to ensure that it ends in a newline so your added key is on its own line. On some systems, the **** on this line must be written **** so the local shell does not try to process it as an escape character.

The third and fourth lines transfer the public key to the remote host and add it as a key that is authorized to log in without entering a password.

Loading the SSH Private Key into the User Agent

Wing expects you to use an SSH user agent to store your private keys, so that **ssh** can access them as needed without having to prompt you for a password. If you normally use a command like **ssh -i mykey.pem user@remote** to connect to your remote host, you will need to instead load your key into the user agent.

To do this, run **ssh-add** on the host where the IDE is running. You will be prompted for the encryption password for the private key, if any, and then the key will be loaded into the user agent.

On OS X Sierra, you will need to add the following to your **~/.ssh/config** to tell **ssh** to communicate with **Keychain Access**:

```
Host *
  UseKeychain yes
  AddKeysToAgent yes
```

On Cygwin you will first need to run **ssh-agent bash** and then **ssh-add** because **ssh-agent** is not running by default.

Now you should be able to connect to the remote host without having to enter a password as follows:

```
ssh username@remotehost
```

Trouble-Shooting

The most common cause of problems in making this work is misconfiguration of OpenSSH on the remote host. OpenSSH will entirely ignore your **.ssh** directory if you do not **chmod 700 .ssh** to make its contents accessible only by its owner.

The **.ssh** directory must be in the home directory of the account used to connect to the remote host, and must be owned by that user. The home directory on the remote host is typically referred to as **~** and will be printed by **echo ~** on the remote host.

In addition, the **authorized_keys** file must contain **\n** line delimiters and not Windows style **\r\n** newlines.

The commands earlier above take care of each of these requirements. If you transfer the key to the **authorized_keys** file some other way (for example, through a file share) then you will need to make sure that these requirements are met.

For more detail on solving SSH configuration problems, see How to Troubleshoot SSH Authentication Issues and How to Troubleshoot SSH Connectivity Issues.

Using a Non-Default SSH Port

If your remote server is running SSH on a non-default port, then you will also need to edit your SSH configuration on the host where the IDE is running to set that port. This is done in **~/.ssh/config** with an entry that looks like this:

```
host myhost.mydomain.com
   port 8022
```

17.8.2. Working With PuTTY

Use the following instructions to set up SSH access from Windows using Putty.

If you don't already have it, download and install the complete suite of tools provided by PuTTY. You will need **putty.exe**, **plink.exe**, **pscp.exe**, and **puttygen.exe**. We recommend using the **MSI** installer, so you have all the necessary tools placed in a location where Wing can find them.

Generating an SSH Key Pair

If you don't already have an SSH key set up, you will need to generate one by running **puttygen.exe**, pressing the **Generate** button, providing the requested random input by moving your mouse over the blank area, entering and confirming a passphrase, and then saving both the public and private key files. The private key file is typically named **id_rsa.ppk** and the public key file is **id_rsa.pub**.

Moving the SSH Public Key to the Remote Host

Next paste the contents of the area labeled **Public key for pasting into OpenSSH authorized_keys file** in the **puttygen** window into a file that you will transfer to the remote host to add it to **~/.ssh/authorized_keys**. If you didn't just generate a new key, you can instead start **puttygen** and load your existing key with the **Load** button. Then you can right-click to select all and then copy from the **puttygen.exe** window.

You may have to create the directory **~/.ssh** on the remote system and/or the **authorized_keys** file within it. Note that the directory **~/.ssh** must be readable only by the login user and no one else. Otherwise ssh refuses to use it. You can make sure it has the correct permissions with **chmod 700 ~/.ssh**.

Loading the SSH Private Key into the User Agent

Finally, run **pageant.exe** on Windows, right click on the small icon that appears in the lower right of your screen, select **Add Key**, and select your **id_rsa.ppk** private key file.

Note that you may need to restart **pageant** and load your key into it each time you restart Windows or log out and back in. Be sure to run **pageant** as the same user that is running Wing. For example, if run in a console that is running as **Administrator** then Wing will not be able to connect to **pageant**.

Now you should be able to connect to the remote host without having to enter a password as follows:

```
plink username@remotehost
```

Trouble-Shooting

The most common cause of problems in making this work is misconfiguration of OpenSSH on the remote host. OpenSSH will entirely ignore your **.ssh** directory if you do not **chmod 700 .ssh** to make its contents accessible only by its owner.

The **.ssh** directory must be in the home directory of the account used to connect to the remote host, and must be owned by that user. The home directory on the remote host is typically referred to as **~** and will be printed by **echo ~** on the remote host.

In addition, the **authorized_keys** file must contain **\n** line delimiters and not Windows style **\r\n** newlines.

For more detail on solving SSH configuration problems, see How to Troubleshoot SSH Authentication Issues and How to Troubleshoot SSH Connectivity Issues.

Using a Non-Default SSH Port

If your remote server is running SSH on a non-default port, then you will also need to edit your SSH configuration on the host where the IDE is running to set that port. This is done by running **putty**, entering a host name or ip address and the port number to use, and saving that host name as a saved session (all on the initial **Session** tab). Once this is done, any connection to that host name, also if made from the command line or Wing, will use the configured port.

17.8.3. Enabling Windows 10 OpenSSH Client

Newer versions of Windows 10 offer OpenSSH as an optional feature. This still seems to be prone to more problems than **PuTTY** but may work as an SSH client for setting up Wing's remote development support.

To enable Windows 10 OpenSSH, open the Settings application and go to the **Apps > Apps & features** page. Click on **Manage optional features** and use **Add a feature** to add **OpenSSH Client**. This installs OpenSSH into **\Windows\System32\OpenSSH** on your system drive. You may need to restart Wing before it finds it the installation, and you may need to log out and back in again before you can use it from the command line.

Once OpenSSH is installed, you will need to enable **ssh-agent** separately. To do that, open a Command Prompt as Administrator by typing **cmd** into Window's search area and right clicking on the **Command Prompt** result to select **Run as Administrator**. Then type the following at the prompt:

```
sc config ssh-agent start= demand
```

After this is done you can close the Command Prompt that is running as Administrator and proceed with the above instructions to generate a key pair and load it into **ssh-agent**.

Scripting and Extending Wing

Wing Pro and Wing Personal provide an API that can be used to extend the IDE's functionality with scripts written in Python. Scripts add to the IDE's command set, which is accessible from menus, the toolbar, and key bindings.

Wing finds and loads scripts at startup, and reloads them when they are edited within Wing and saved to disk. The API allows scripts to access the editor, debugger, project manager, search tools, source code analysis engine, asynchronous task manager, and a range of other functionality. The scripting API also provides access to all of Wing's preferences and commands.

Simple scripts can be developed and debugged using error messages displayed in the **Scripts** channel of the **Messages** tool. It is also possible to configure a project that supports auto-completion and integrated documentation for the scripting API, and that allows debugging extension scripts within Wing.

More advanced scripting, including the ability to add new tools, is available as well.

18.1. Scripting Example Tutorial

Trying a simple example script is the best way to get started with Wing's scripting API. The following quick tutorial will take you through the process.

Creating an Extension Script

User-defined scripts are usually placed inside a directory named **scripts** located inside the Settings Directory. The **scripts** sub-directory needs to be created if it does not already exist.

Try adding a simple script now by pasting the following into a file called **test.py** inside the **scripts** directory:

```
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 from the IDE.

Executing the Script

Try executing the script by selecting **Command by Name** in the **Edit** menu. This displays an entry area at the bottom of the window, where you can type **test-script** and then press the **Enter** key. Since the script has an argument without a default value, Wing will collect that in the same entry area at the bottom of the IDE window. Type a string and then press **Enter**. The script will pop up a modal message dialog containing the text that you typed.

Of course this is not how you will usually invoke a script. Instead, scripts be assigned to a key binding or added to a menu, as described in the next section.

Try assigning a key binding now to the command **test-script** with the **User Interface > Keyboard > Custom Key Bindings** preference. For details on adding key bindings in Wing, see Key Bindings.

Editing the Script

In order to place your script in a new menu in the menu bar, add the following after the function definition:

```
test_script.contexts = [wingapi.kContextNewMenu("Scripts")]
```

As soon as you save this change, a menu **Scripts** should appear in the menu bar with one item **Test Script**. This illustrates how scripts are auto-reloaded as they are saved from Wing. For more information on adding scripts to menus, see Adding Scripts to the GUI.

Next, 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.

Auto-Completion and Integrated Documentation

With some additional configuration, it is possible to enable auto-completion, auto-invocation, integrated documentation, and goto-definition for the scripting API. This is done as follows:

(1) First create a new project from the **Project** menu with the default settings.

(2) Next locate the **src** directory inside the **Install Directory** shown in Wing's **About box**. This is the directory that contains **wingapi.py**.

(3) Finally, add the full path of the directory found in step (2) to the **Python Path** in **Project Properties**.

Once this is done auto-completion in the editor, documentation in the Source Assistant, and goto-definition should all work when you **import wingapi** and work with its contents. In Wing Pro, **Find Uses** and the auto-invocation auto-editing operation will also work for the API.

Debugging Extension Scripts

With some additional project setup, it is also possible to debug scripts using Wing. This is a much richer way to develop extension scripts than clicking on tracebacks in the **Messages** tool. See Debugging Extension Scripts for details.

Other Example Scripts

Wing ships with many other example scripts. These are in **scripts** inside the **Install Directory** listed in Wing's **About box**. The most relevant examples for simple scripting can be found in **editor-extensions.py**. This shows how to access and alter text in the current editor, among other things.

Other extensions scripts are available in **scripts** in the contributed extensions repository.

18.2. Overview of the Scripting Framework

Scripts are Python modules or packages containing one or more Python functions which implement the script's functionality. Any top-level function with a name that starts with a character other than underscore _ is added to Wing's command set, so it becomes accessible from menus, key bindings, and the toolbar. Scripts can also use the scripting API to hook into IDE functionality in other ways, for example to perform an action every time an editor is saved to disk.

When Wing starts up, it will search for scripts in all directories in the path configured with the **IDE Extension Scripting > Search Path** preference. By default this path contains a directory named **scripts** within the Settings Directory. Scripts can also be placed in **scripts** inside the **Install Directory** shown in Wing's **About box**, but this is not recommended since it is harder to manage across updates of Wing.

Scripts can be modules named ***.py** and packages, which are directories that contain a file named **__init__.py** file and any number of other ***.py** files or sub-packages. For packages, Wing loads only the modules that are imported in the **__init__.py** file.

Script files within each directory are scanned in alphabetical order. When multiple script-defined commands with the same name are found, the command that is loaded last overrides any loaded earlier under the same name. However, scripts cannot replace internally defined commands, as detailed below.

Naming Commands

Commands added by scripts 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, optionally with underscores 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 **do_it** is defined inside a module named **xpext.py**, then the short name of the command created will be **do-it** and the FQN will be **.user.xpext.do-it**.

Overriding Internal Commands

Wing will not allow a script to override any of the commands 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 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.

Execution Context

Scripts are run in the same process space as the IDE, using Wing's private Python 2.7 interpreter. Because they are in the same process space, scripts have the potential for breaking the IDE. For example, a script entering into an infinite loop will lock up Wing.

To avoid this, script-provided functionality must be written within the framework for cooperative asynchronous multi-tasking that Wing uses internally. In this approach, lengthy computations are split into small units that are interleaved with the main event loop. This is supported in the scripting API by **InstallTimeout** in **CAPIApplication**. This calls a given function periodically until it is removed with **RemoveTimeout**, until it returns a value where **bool(value)** is **False**, or until the script that installed it is reloaded.

This example implements a command that counts down from 10 in the status area at the bottom of the screen:

```python
import wingapi
def start_counting():
  counter = [10]
  def count():
    counter[0] -= 1
    wingapi.gApplication.SetStatusMessage("Time left: {}".format(counter[0]), timeout=1)
    return counter[0]
  wingapi.gApplication.InstallTimeout(1000, count)
```

To interact asynchronously with a sub-process, use this approach in combination with **AsyncExecuteCommandLine***. Here is an example that runs **ping** for ten seconds and shows status messages at the bottom of the IDE window:

```python
import wingapi
import sys
import time
def process_example():
  cmdline = ['ping', '-t', '9', 'wingware.com']
  handler = wingapi.gApplication.AsyncExecuteCommandLine(cmdline[0], None, *cmdline[1:])
  timeout = time.time() + 10
  def poll(timeout=timeout):
    kill = time.time() > timeout
    if kill or handler.Iterate():
      stdout, stderr, err, status = handler.Terminate(kill)
      if kill:
        msg = "Time out"
      elif err is not None:
        msg = "Process failed to start;  exit_status={}, errno={}".format(status, err)
      else:
        msg = "Process exited; stdout len={}; stderr len={}".format(len(stdout), len(stderr))
      wingapi.gApplication.SetStatusMessage(msg)
      return False
    else:
      if handler.stdout:
        msg = "Last output line: {}".format(handler.stdout[-1].splitlines()[-1])
      else:
        msg = "No output yet"
      wingapi.gApplication.SetStatusMessage(msg)
      return True

  wingapi.gApplication.InstallTimeout(100, poll)
```

For additional examples, see the **scripts** folder inside the **Install Directory** listed in Wing's **About box**.

Signals

Another important concept in writing extension scripts is the use of signals to control when script-provided functionality is implemented. Each of the classes in the scripting API provides signals that notify of different events in the user interface. Signals may be connected to handlers that are called when the signal is emitted with the defined set of parameters for that signal.

For example, **CAPIApplication** emits **project-open(filename)** when a new project has been opened. The signal can be connected to a handler function as follows:

```
import wingapi
def _proj_open(filename):
  wingapi.gApplication.SetStatusMessage("Project opened: {}".format(filename))
gSignalID = wingapi.gApplication.Connect('project-open', _proj_open)
```

Once this script is loaded into Wing, **_proj_open** will be called every time a new project is opened. This example displays message in the status area at the bottom of the IDE window. The message includes the filename, which is the single parameter sent with this particular signal.

Disconnecting from the signal later would be accomplished as follows for the above example:

```
wingapi.gApplication.Disconnect(gSignalID)
```

Signals for each class are documented in **wingapi.py**. For additional examples, see the **scripts** folder inside the **Install Directory** listed in Wing's **About box**.

Reloading Scripts

Wing watches script files and automatically reloads them when they are edited inside Wing and saved to disk. The only exception to this occurs when a new script is added. In this case, Wing will not load the new script until **Reload All Scripts** in the **Edit** menu is executed or the IDE is restarted.

Reloading will not work for any file that sets **_ignore_scripts** at the top level, or for modules outside of the script path. For details on how reloading works, see Advanced Scripting.

18.3. Scripting API

Wing's formal scripting API consists of several parts:

1. The contents of the **wingapi.py** file in **src** inside the **Install Directory** listed in Wing's **About box**. Scripts gain access to the API with **import wingapi**. See the API Reference for details or work directly with **wingapi.py** as described under *Auto-Completion and Integrated Documentation* in the Scripting Example Tutorial.

2. The portions of the **wingutils.datatype** and **guiutils.formbuilder** modules that are documented in Argument Collection.

3. All of the documented commands which can be invoked using **ExecuteCommand()** in **wingapi.gApplication**. 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 read and changed using **GetPreference()** and **SetPreference()** in **wingapi.gApplication**.

5. The standard library modules from Python.

Advanced scripts may also "reach through" the API into Wing internals. However, this requires reading Wing's source code and no guarantee is made that internals will remain unchanged or will change only in a backward compatible manner.

18.4. Script Syntax

Scripts are syntactically valid Python with certain extra annotations and structure that are used by Wing to determine which scripts to load and how to execute them.

18.4.1. Script Attributes

The scripting API uses function attributes as a way to annotate script functions that define a new command for the IDE. These are used to define the type of arguments the command expects, command availability, the display name and documentation for the command, and the contexts in which the command should be made available in the GUI.

The following function attributes may be set. Each one can be a value or a callable object that returns the value:

arginfo defines the argument types for any arguments passed to the command. This is used by Wing to drive automatic collection of argument values from the user. When this is missing, all arguments are treated as strings. See Argument Collection for details.

available defines whether or not the command is available. If missing, the command is always available. If set to a constant, **bool(available)** defines availability of the command. If set to a callable object, it is invoked with the same arguments as the command and the return value determines availability of the command.

label provides 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()**)

contexts lists the GUI contexts the which the command should appear. See Adding Scripts to the GUI for details.

doc is the documentation for the command if for some reason a docstring in the function definition can't be used.

flags is a dictionary of options that control behavior of the script. Currently the only option is **force_dialog_argentry** which may be set to **True** to collect arguments for the script in a dialog, rather than at the bottom of the IDE window.

plugin_override may be set in scripts that are designated as plugins, in order to indicate that the command should be enabled even if the plugin is not.

The following example uses the above to add a script-defined command to the editor context menu, set the label used in the menu, and indicate when the command is available:

```
import wingapi
def test_script():
  pass
test_script.contexts = [wingapi.kContextEditor()]
test_script.label = "Do Nothing"
def _test_script_available():
  return 1
test_script.available = _test_script_available
```

Script-Wide Default Attributes

Default values for some of the attributes defined above can be set at the top level of the script file:

_arginfo is the default argument information to use for scripts that don't have an **arginfo** attribute of their own.

_available defines the default availability of scripts without an **available** attribute.

_contexts sets the default GUI contexts into which scripts should be added if they do not have their own **contexts** attribute.

Some additional attributes are also supported, to control how Wing treats the script file as a whole:

_ignore_scripts can be set to **True** to completely ignore this script file.

_i18n_module names the **gettext** internationalized string database to use when translating docstrings in this script. See Internationalization and Localization for details.

_plugin 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 Plugins for details.

18.4.2. Adding Scripts to the GUI

Scripts that define a new command for the IDE may add that command to the user interface in various ways. This is done by setting the **contexts** attribute on the function that implements the command.

The following example adds the script-provided command **test-script** to a new menu **Scripts** and the editor's right-click context menu:

```
test_script.contexts = [
  wingapi.kContextNewMenu("Scripts"),
  wingapi.kContextEditor(),
]
```

These contexts are available for script-provided commands:

kContextNewMenu(title, group=0) adds an item to a menu in the menu bar. If multiple scripts use the same context, they are all added to the same menu. The required argument **title** specifies the title to use for the menu, and the optional argument **group** is a number that allows separating items in the menu into groups. Groups are created as needed and items are listed in alphabetical order within them.

kContextEditor() adds an item to the end of the editor's right-click context menu.

kContextProject() adds an item to the end of the project's right-click context menu.

kContextCommonMenu adds an item to the end of the common actions item in the top right of Wing's window.

Regardless of whether script-provided command is added to any GUI context, it will always be listed under both short and fully qualified name in the auto-completer for **Command by Name** in the **Edit** menu, and in the **User Interface > Keyboard > Custom Key Bindings** preference.

18.4.3. Argument Collection

Commands that are defined in scripts can take arguments, optionally with a default value. Wing can collect any missing arguments for a command invocation by interacting with the user in a dialog or, in some keyboard personalities, in the status area at the bottom of the IDE window.

By default, Wing derives the labels to use for arguments from the argument name and assumes that the argument being collected is a string. When this is not the case, argument type can be specified by setting the **arginfo** function attribute on the script function that defines the command. This uses the **CArgInfo** from **wingapi.py** and the **datatype** and **formbuilder** modules from Wing's internals, as documented below.

Example

The following sets up two arguments, one that is a filename, and another that allows selecting from a popup menu:

```
import wingapi
from wingutils import datatype
from guiutils import formbuilder

def test_arg_entry(filename, word):
  wingapi.gApplication.ShowMessageDialog('Choice {}'.format(word), "You chose: {}".format(filename))

_choices = [
  ("Default", None),
  None,
  ("One", 1),
  ("Two", 2),
  ("Three", 3)
]
```

```
test_arg_entry.arginfo = {
  'filename': wingapi.CArgInfo(
    "The filename to enter",              # The tooltip shown to use over this field
```

```
    datatype.CType(''),                    # The data type is string
    formbuilder.CFileSelectorGui(),        # Use a file selection field to collect the value
    "Filename:"                            # The field label
    ),
  'word': wingapi.CArgInfo(
    "The word to enter",
    datatype.CType(''),
    formbuilder.CPopupChoiceGui(_choices), # Use a popup menu to collect this value
    "Word:"
  )
}
```

CArgInfo

The arguments used to instantiate a **CArgInfo** instance for the **arginfo** function attribute are:

doc sets the documentation string for the argument.

type sets the data type, using one of the classes descended from **wingutils.datatype.CTypeDef**. See below for the most commonly used ones.

formlet sets the type of GUI formlet to use to collect the argument from the user. This is one of the classes descended from **wingutils.formbuilder.CDataGui** See below for the most commonly used ones.

label sets the label to use for the argument when collected from the user. When this argument is omitted, the label is built from the function name with **cmdname.replace('_', ' ').title()**.

Commonly Used Types

The following classes in **wingutils.datatype** cover most cases needed for scripting:

CBoolean specifies a boolean. The constructor takes no arguments.

CType specifies the type matching one of the parameters sent to the constructor. For example, **CType("")** is a string, **CType(1)** is an integer, and **CType(1.0, 1)** is a float or an integer.

CValue restricts a value to one of those passed to the constructor. For example **CValue("one", "two", "three")** allows a value to be either **"one"**, **"two"**, or **"three"**.

CRange specifies a numeric range between the first and second argument passed to the constructor. For example, **CRange(1, 10)** allows a value between **1** and **10**, inclusive.

Additional types are defined in **src/wingutils/datatypes.py** in the Wing source code, but these are not usually needed in describing scripting arguments.

Commonly Used Interface

The following classes in **guiutils.formbuilder** cover most of the data collection fields needed for scripting:

CSmallTextGui collects a short text string, with history, auto-completion, and other options. The constructor takes the following keyword arguments, all of which are optional:

max_chars sets the maximum allowed text length. Set this to **-1** to allow any length. Default: **80**

history is a list of strings for the history, most recent first, that is accessed with the up and down arrow keys. This may be the list or a callable that returns the list. Default: **None**

choices is a list of strings with all valid choices, to use in the auto-completer that is shown as the user types. This may be a list or a callable that takes a fragment and returns all possible matches. Default: **None**

partial_complete is set to **True** to only complete as far as the unique match when the **Tab``key is pressed for auto-completion.  When set to ``False**, all of the currently selected auto-completion match will be entered instead. Default: **True**

stopchars is a string of characters that always stop partial completion. For example, **'/'** might be used to prevent completion of an entire url. Default=``"``

allow_only is a list of characters allowed for input. All others are not processed. When this is set to **None**, it allows all characters to be input. Default: **None**

auto_select_choice is set to **True** to automatically select all of the entry text when browsing on the auto-completer. This is used so that the entry will be erased if any subsequent typing occurs. Default: **False**

default is the default value to auto-enter initially. Default: **"**

select_on_focus can be set to **True** to select any existing text when focus enters the field. Default: **False**

editable can be set to **False** to display the field but to prevent editing it. Default: **True**

selection can be set to a **(start, end)** tuple to select a range of text in the auto-entered **default** value. If omitted, nothing is selected. Default: **None**

CLargeTextGui is an multi-line entry area for longer text strings. The constructor takes no arguments.

CBooleanGui is a single checkbox for collecting a boolean value. The constructor takes no arguments.

CFileSelectorGui is a keyboard-driven file selector with auto-completion, history, and ability to browse using a standard file dialog. The constructor takes the following optional keyword arguments:

name_type specifies what type of file or directory is being selected: **'existing-file'**, **'existing-dir'**, **'existing-executable-file'**, **'new-dir'**, or **'save-as-file'**

default is the default value to pre-fill into the field. Default: **"**

default_ext specifies the default file extension to use. Default: **None**

filters is a list of valid file name extensions. For example **['py', 'pyi']** to select either a ***.py** or ***.pyi** file. Default: **None**

history can be set to a list of past choices, most recent first, to traverse with the up and down arrow keys. Default: **()**

tab_shows_completer indicates that pressing the **Tab** key should show the auto-completer. Default: **False**

hostname (Wing Pro only) specifies the name of a remote host from which the file or directory should be selected. Default: **''**, which indicates the local host.

CPopupChoiceGui is 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 to insert a divider into the menu

A string to insert that value into the menu. The label used in the menu is derived from the value: **label = str(value).replace('_', ' ').title()**

(value, label) inserts the value into the menu using the given label.

(value, label, tooltip) inserts the value into the menu using the given label and displays the given tooltip when the mouse hovers over the item in the menu.

CNumberGui is a small entry area for collecting a number. The constructor takes the following required arguments:

min_value is the minimum allowable value.

max_value is the maximum allowable value.

page_size is the increment to use when the when scroller is used.

num_decimals is the number of decimal places to show. This is set to **0** to collect an integer.

Additional fields for collecting data are defined in **src/guiutils/formbuilder.py** in the Wing source code, but these are not usually needed for scripting.

18.4.4. Importing Other Modules

Scripts can import other modules, including of course **wingapi**, but also Python's standard library, and even modules from Wing's internals.

However, because of the way in which Wing loads scripts, users should generally avoid importing one script module into another. If this is done, the module loaded by the **import** will not be the same as the one loaded by the scripting manager, and two entries in **sys.modules** will result. This happens because Wing uniquifies the module name internally to prevent conflicts between different like-named script modules and/or Wing's internals.

In practice, this is only a problem if data at the top level of the script module is shared in some significant way, so that two loaded copies of the module would be a problem. Be sure to completely understand how modules and import work in Python before importing one script module into another.

18.4.5. Internationalization and Localization

String literals and docstrings defined in scripts 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 should be 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 **_()**.

The **pygettext.py** script included with Python can be used to extract and merge strings into a ***.po** file and then convert that file into an ***.mo** file. See Python's documentation for gettext for details.

18.4.6. Plugins

When a script contains the **_plugin** attribute at the top level, it is treated as a plugin that can enable or disable itself as a whole and be enabled or disabled by the user from **Plugins** in the **Edit** menu.

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:

```
import wingapi
def _activator_cb(plugin_id):
  return True
_plugin = ('myplugin', _activator_cb)
```

EnablePlugin may also be called from any other script code, including signal handlers. For example, a script might watch the current project using the **project-open** signal on **CAPIApplication** and enable or disable the plugin based on which project is open:

```
import wingapi

# Activator is needed to store the uniquified plugin_id; start out disabled
_plugin_id = [None]
```

```
def _activator_cb(plugin_id):
    _plugin_id[0] =  plugin_id
    return False
_plugin = ('myplugin', _activator_cb)

# Watch project and activate plugin based on project name
def _proj_open(filename):
    wingapi.gApplication.EnablePlugin(_plugin_id[0], 'ide' not  in filename)
wingapi.gApplication.Connect('project-open', _proj_open)
```

When a plugin is inactive, its commands are undefined and any menus or menu items it added 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 **Edit** menu, this completely prevents loading the plugin, which overrides **_activator_cb** and any **_plugin_override** attributes for functions that define commands for the plugin.

18.5. Debugging Extension Scripts

Wing can debug extension scripts that you develop for the IDE. This is done by setting up a new project from Wing's **Project** menu. Select the default project type and for **Python Executable** select **Command Line** and then enter the full path to the Python executable that Wing uses to run itself. This located below the **Install Directory** listed in Wing's **About box**, as follows:

> **OS X: Contents/Resources/bin/__os__/osx/runtime-python2.7/bin/python**

> **Linux: bin/__os__/linux-x64/runtime-python2.7/bin/python**

> **Windows: bin__os__\win32\runtime-python2.7\bin\python.exe**

Be sure to use the *full path* to the executable and not the above partial paths.

Press **OK** in the **New Project** dialog to create the project, then select **Add Existing Directory** from the **Project** menu and add the **Install Directory**.

Next navigate to **bootstrap/wing.py** in the **Project** tool, right click on it, and select **Set As Main Entry Point**.

Then set up Wing to run Python in optimized mode, so it can load the precompiled code in your Wing installation, by setting **Python Options** under the **Debug/Execute** tab of **Project Properties** to **Custom** with a value of **-u -O**.

Finally, on OS X only, you will need to open **Project Properties** from the **Project** menu, select **Add To inherited environment** under **Environment**, and paste in the following:

```
INSTALLDIR=/Applications/WingPro.app/Contents/Resources
RUNTIMES=${INSTALLDIR}/bin/__os__/osx
QTVERSION=qt5.10
QTRUNTIME=${RUNTIMES}/runtime-${QTVERSION}
SCIRUNTIME=${RUNTIMES}/runtime-scintillaedit-${QTVERSION}
```

```
DYLD_LIBRARY_PATH=${QTRUNTIME}/lib:${SCIRUNTIME}/lib
DYLD_FRAMEWORK_PATH=${DYLD_LIBRARY_PATH}
```

If you didn't install Wing in the **/Applications** folder then you will need to edit the first line to specify the correct **Install Dir**. You may also need to adjust the value of **QTVERSION** for your Wing installation.

Then save your project to disk.

You should now be able to select **Start/Continue** from the **Debug** menu to start up a copy of Wing in the debugger. Any breakpoints set in scripts that you have added in the **scripts** directory will be reached as you work with the debugged copy of Wing. You will see and can navigate the entire stack, but Wing will not be able to show files for most of Wing's code. If you need to see the source code of Wing itself, you will have to obtain the source code as described in Advanced Scripting.

18.6. Advanced Scripting

While Wing's API will remain stable across future releases of the IDE, not all functionality is exposed by the API. Scripts can also be written to reach through Wing's API into internal functionality that may change from release to release, but in most cases stays the same. The most common reason to reach through the API is to add a new tool panel to Wing.

An example of this can be seen in **pylintpanel.py** in the **scripts** directory inside the the **Install Directory** listed in Wing's **About box**.

Working with Wing's Source Code

More advanced scripts like those that define a new tool are be easier to develop if Wing is run from its source code, usually as a debug process that is controlled by another copy of Wing.

To obtain Wing's source code, you must have a valid license to Wing Pro 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 further instructions.

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.

Script files that define **_no_reload_scripts** at the top level of the module 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. Wing executes the top level of the module as normal when importing a module in Python. This may cause signal connections and other calls to the API to occur.
7. If module load fails due to an error in the code, any timeouts or other connections registered by the module during partial load are removed and the module is removed from **sys.modules**.
8. If the module contains **_ignore_scripts**, then any timeouts or other connections are removed and scripts in the file are ignored.
9. Otherwise, Wing adds all the script-defined commands in the module to the command registry and loads any sub-modules in the same way, 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.

18.7. API Reference

This chapter documents the scripting API available in **src/wingapi.py** inside the **Install Directory** listed in Wing's **About box**. To use the API, add **import wingapi** to the top of your script.

> ### Note
>
> This documentation is also available interactively in the **Source Assistant** if a project is set up as described as described in *Auto-Completion and Integrated Documentation* in the Scripting Example Tutorial.

See also the examples **scripts** directory in the **Install Directory**.

18.7.1. API Reference - Utilities

A Note on Filenames

File names in the API may either be the name of a local file on disk or a URL for untitled or remote files.

IsUrl(filename)

Tests whether the given **filename** is a URL. Use this on filenames obtained from the API to determine how to treat them.

When this returns **False**, the **filename** is a local file name.

Otherwise, the **filename** is a URL in the one of the following forms:

- Untitled buffers use filenames starting with **unknown:** For example, **unknown:untitled-1.py** and **unknown:Scratch** both refer to an unsaved file.
- Remote files and directories use filenames in the form **ssh://hostname/path/to/item** where **hostname** is the **Identifier** in a Remote Host.

18.7.2. API Reference - Application

Class CAPIApplication

API for the top-level of IDE functionality. This should be accessed through **wingapi.gApplication**.

Signals

A callback can be connected to the following signals using **Connect(signal, cb)**:

destroy: The application is closing. Calls **cb(app:CAPIApplication)**

editor-open: An editor was opened. Calls **cb(editor: CAPIEditor)**

document-open: A document was opened. Note that several editors may share a document. Calls **cb(doc:CAPIDocument)**

project-open: A project was opened. Calls **cb(filename:str)**

project-close: A project was closed. Calls **cb(filename:str)**

active-editor-changed: Active editor has changed. Calls **cb(editor:CAPIEditor)**

active-window-changed: Active window has changed. The window is None if Wing is no longer at front. Calls **cb(window_name:str)**

perspective-changed: Current perspective has been changed. Calls **cb(perspective_name:str)**

Use **Disconnect(signal_id)** to preemptively disconnect a signal handler, where **signal_id** is the signal ID previously returned from **Connect**.

Top-level Settings and Environment

CAPIApplication.GetProductInfo()

Returns the current Wing version, update, product code, product name, and release date.

If the full version of Wing is 1.2.3.4 then version will be '1.2.3' and update will be '4'. If the current version is a pre-release then update may be prepended with one letter as follows:

```
'a': alpha release
'b': beta release
'c': release candidate
```

The valid product codes and names are:

```
0x00000001 'Personal'
0x00000002 'Professional'
0x00000008 '101'
```

Example return values:

```
('5.1.3', '2', 0x00000002, 'Professional', 'Mar 17, 2014')
('7.0.0', 'a1', 0x00000001, 'Personal', 'Aug 1, 2018')
```

CAPIApplication.GetWingHome()

Returns the Install Directory from which Wing is running.

CAPIApplication.GetUserSettingsDir()

Returns the active User Settings directory.

CAPIApplication.GetStartingDirectory(force_local=True)

Get the most logical starting directory to use when browsing for files or directories. This varies based on the focus and selection on the user interface. When **force_local** is True, only a local starting directory is returned. Otherwise a starting directory on a remote host may be returned as a URL in the form **ssh://hostname/path/to/dirname**, where **hostname** is the **Identifier** of a Remote Host. Use **IsUrl** to distinguish urls from directory names.

CAPIApplication.FindPython()

Find the default Python interpreter that Wing will use if none is specified with **Python Executable** in **Project Properties**. Wing tries looking for it as follows:

On Linux:

- Try **python** in the current environment
- Search **PATH** for **python*** (such as **python2.7** or **python3.7**)
- As a last resort, use the last known working Python if there was one

On OS X:

- Use **/Library/Frameworks/Python.framework/Versions/Current/bin/python** if is exists and is valid
- Search as for Linux

On Windows:

- Try **python** in the current environment
- Look for the latest version in the registry using the keys **HKEY_CURRENT_USER\SOFTWARE\PYTHON\PYTHONCORE\#.#\INSTALLPATH** and **HKEY_LOCAL_MACHINE\SOFTWARE\PYTHON\PYTHONCORE\#.#\INSTALLPATH**
- As a last resort, use the last known working Python if there was one

Return Value: the full path to the interpreter. The value is validated in that the interpreter is actually executed and sys.executable is returned.

NOTE: This call will ignore versions of Python that Wing does not support.

Command Execution

These methods are used to execute IDE commands that are documented in the Command Reference.

CAPIApplication.CommandAvailable(cmd, **args)

Check whether a command is available for execution.

The **cmd** can be the name of a command in Wing's Command Reference, or the name of a command added by an extension script.

Any arguments are passed as keyword arguments using the documented argument names for the command being invoked. In most cases the value of optional arguments won't affect command availability, so they may usually be omitted.

CAPIApplication.ExecuteCommand(cmd, **args)

Execute a command with the given keyword arguments.

The **cmd** can be the name of a command in Wing's Command Reference, or the name of a command added by an extension script.

Any arguments are passed as keyword arguments using the documented argument names for the command being invoked.

To execute an external command or command line, use **ExecuteCommandLine**, **AsyncExecuteCommandLine***, or **ExecuteOSCommand** instead.

Asynchronous Timeouts

CAPIApplication.InstallTimeout(timeout, fct)

Install a function to be called as a timeout after a given number of milliseconds. The function is called repeatedly at the given interval until its return value evaluates to **False** or **None**.

Returns a **timeout_id** that may be sent to **RemoveTimeout** to remove the timeout prematurely.

Note that the timeout will be removed if its script module is reloaded, in order to avoid calling old byte code. For this reason, script modules must reinstall timeouts during initialization.

CAPIApplication.RemoveTimeout(timeout_id)

Remove a timeout previously installed with **InstallTimeout**.

Access to Key Objects

CAPIApplication.GetActiveWindow()

Get the internal name of the currently active window. This is **None** if no window in Wing has the focus.

CAPIApplication.GetActiveEditor()

Get the currently active **CAPIEditor** or **None** if no editor has the focus.

CAPIApplication.GetActiveDocument()

Get the **CAPIDocument** for the currently active editor, or None if no editor has the focus.

CAPIApplication.GetCurrentFiles()

Get a list of the the currently selected files. The list returned depends on the current focus and selection in the user interface. Files may be selected in the current editor, or in the **Project**, **Source Browser**, and other tools.

Returns a list of full path filenames for the file or files, or **None** if none are selected.

For untitled or scratch buffers, the file name is prefixed with **unknown:**. For remote files, the file name will be a URL. Use **IsUrl** to distinguish between file names and URLs.

CAPIApplication.GetCurrentSourceScopes()

Get the current source scopes, including file name, line number, and Python scope name. The value returned depends on the current focus and selection in the user interface. Source scopes may be selected in the current editor, or in the **Project**, **Source Browser**, or other tools.

Returns **None** if nothing is selected or a list of scopes, each of which is a list that contains a filename, a line number (0=first), and zero or more source symbol names indicating the nested scope that the user has selected.

For example, if **Class1.Method1** on line **120** of the file **/x/y/z.py** is selected, the return value would be:

```
[["/x/y/z.py", 120, "Class1", "Method1"],]
```

Line **1** is used without any source symbols to indicate the whole file is selected. The following would be returned if multiple items in the **Project** or **Open Files** tools are selected:

```
[["/x/y/a.py", 1], ["/a/b/z.py", 1]]
```

For untitled or scratch buffers, the file name is prefixed with **unknown:**. For remote files, the file name will be a URL. Use **IsUrl** to distinguish between file names and URLs.

CAPIApplication.GetAnalysis(filename)

Get a **CAPIStaticAnalysis** object for the given Python file.

CAPIApplication.GetAllFiles(visible_only=False, sticky_only=False)

Get a list of the full path names of all currently open files, whether or not a **CAPIDocument** object or editor has been created for them.

Optionally filter the result to omit non-visible files or those that are opened in non-sticky transient mode. See Transient Non-Sticky Editors for details.

For untitled or scratch buffers, the file name is prefixed with **unknown:**. For remote files, the file name will be a URL. Use **IsUrl** to distinguish between file names and URLs.

CAPIApplication.GetOpenDocuments()

Get all currently open **CAPIDocument** objects. This includes only those documents that have already been shown in an editor, since documents are not created until they need to be shown.

Note that this may also include documents active for searching or source analysis, for which no editor is open.

CAPIApplication.GetProject()

Get the currently open **CAPIProject**. Returns **None** if no project is open.

CAPIApplication.NewProject(completed_cb, failure_cb=None)

Create a new project. **complete_cb** is called with no arguments when the new project has been created and opened and **failure_cb** is called with no arguments if the user cancels closing the current project.

CAPIApplication.GetDebugger()

Get the **CAPIDebugger** singleton, for access to debugger functionality.

CAPIApplication.ShowTool(name, flash=True, grab_focus=True)

Show the given tool in the user interface. The most recently used instance of the tools is shown, or a new instance is created at its default location.

The **name** can be one of:

'project', 'browser', 'batch-search', 'interactive-search', source-assistant', 'debug-data', 'debug-stack', 'debug-io', debug-exceptions', 'debug-breakpoints', 'debug-console', 'debug-watch', debug-modules', 'python-shell', 'about', 'messages', 'help', 'indent', bookmarks', 'testing', 'open-files', os-command', 'snippets', diff', 'uses', 'refactoring', 'code-warnings'

The tool title is flashed if **flash** is **True** and focus is moved to the tool if **grab_focus** is **True**.

CAPIApplication.OpenURL(url)

Open the given URL with an external viewer.

Manage Windows

CAPIApplication.CreateWindow(name)

Create a new window with given internal name. The window is initially blank. Use **OpenEditor(**) with the given name to fill it.

CAPIApplication.CloseWindow(name, allow_cancel=True)

Close the window with given internal name, and all the editors in it. When **allow_cancel** is **False**, the window is closed without prompting to save any changes made there.

Manage Editors

CAPIApplication.OpenEditor(filename,   window_name=None,   raise_window=False, sticky=True)

Open the given file into an editor.

If the **window_name** is given, the editor opens into the window with that internal name. Otherwise the most recently visited window is used. If **window_name** is not the name of an existing window, a new window is created with that name.

The window is not brought to the front unless **raise_window** is **True**.

Set **sticky** to **False** to cause Wing to auto-close the editor when hidden and more than the configured number of non-sticky editors is open. See Transient Non-Sticky Editors for details.

Returns the **CAPIEditor** or **None** if opening the file failed.

Note that the file may open under a different name if symbolic links exist.

CAPIApplication.ScratchEditor(title='Scratch', mime_type='text/plain', raise_window=False, raise_view=True, sticky=True, window_name=None)

Create a scratch editor with the given title and mime type. The document can be edited but will never be marked as changed or requiring a save to disk. However, it can be saved with **Save As** if desired.

If **title** contains **%d**, a sequence number will be inserted automatically.

mime_type sets the file type to use. Use **text/x-python** for Python.

The window is raised only if **raise_window** is True.

The view is brought to front within the window only if **raise_view** is True.

Set **sticky** to **False** to cause Wing to auto-close the editor when hidden and more than the configured number of non-sticky editors is open. See Transient Non-Sticky Editors for details.

If the **window_name** is given, the editor opens into the window with that internal name. Otherwise the most recently visited window is used. If **window_name** is not the name of an existing window, a new window is created with that name.

Returns the **CAPIEditor** or **None** if the scratch buffer failed to create.

CAPIApplication.GetMimeType(filename)

Get the mime type Wing is using for the given filename, based on the file name, contents, and **Files > File Types > Extra File Types** preference.

Clipboard

CAPIApplication.SetClipboard(txt)

Store the given text to the clipboard. The text should be a utf-8 string or unicode object.

CAPIApplication.GetClipboard()

Get the text currently on the clipboard, as a unicode string.

Application State

CAPIApplication.GetVisualState(errs=[], style='all')

Get the application's visual state.

The **style** of the state may be one of:

'all' to capture all of the application visual state

'tools-and-editors' to capture which tools are visible, the overall layout of the windows, and which editors are open (but not details like scroll positions, selection, or current search string)

'tools-only' to capture only which tools are visible and the overall layout of the windows (but not which editors are open).

Returns an opaque dictionary with the state, for later use with **SetVisualState**.

Any errors encountered are added to **errs** as strings.

CAPIApplication.SetVisualState(state, errs=[])

Restore saved application state, as previously obtained from **GetVisualState**. Any errors encountered are added to **errs** as strings.

Preferences

CAPIApplication.GetPreference(pref)

Get the value of the given preference.

The **pref** argument should be the fully qualified name of the preference, as given in the Preferences Reference.

CAPIApplication.SetPreference(pref, value)

Set value for the given preference.

The **pref** argument should be the fully qualified name of the preference, as given in the Preferences Reference.

The **value** must conform to the documentation for the preference.

CAPIApplication.ConnectToPreference(pref, cb)

Connect to the given preference so that the given callback is called whenever the value of the preference changes.

The **pref** argument should be the fully qualified name of the preference, as given in the Preferences Reference.

cb is called with no arguments. It may obtain the new value of the preference with **GetPreference**.

The callback will be uninstalled automatically if the caller's script is reloaded.

Returns a signal id that can be used with later **DisconnectFromPreference**.

CAPIApplication.DisconnectFromPreference(pref, id)

Disconnect a preference value callback.

The **pref** argument should be the fully qualified name of the preference, as given in the Preferences Reference.

id is the signal id that was returned from **ConnectToPreference**.

CAPIApplication.ShowPreference(prefname)

Show the given preference by name in the preference manager dialog.

prefname should be the fully qualified name of the preference, as given in the Preferences Reference.

Messages and Status

CAPIApplication.ShowMessageDialog(title, text, checks=[], buttons=[('OK', None)], modal=True, sheet=False)

Display a message dialog to the user with the given **title** and **text**.

If **checks** is non-empty it contains a list of **(label, default, callback)** tuples for extra check boxes to add below the message and above the buttons. The callback is called with the label and check state immediately when the checkbox is used.

Set **buttons** to a list of **(label, action)** pairs to override the default of a single **OK** button. The button action can be **None** to simply close the dialog or it can be a callable taking no arguments that returns **True** to prevent closing of the dialog or **False** to allow it to close when the button is pressed.

The dialog is modal unless **modal** is set to **False**.

sheet is deprecated in Wing 7.0 and should no longer be used.

CAPIApplication.SetStatusMessage(text, timeout=5)

Display a transient status message in the status area at the bottom of the IDE window. The message persists for the given **timeout** (in seconds) or until another status message is shown.

CAPIApplication.ClearStatusMessage()

Clear the status message area at the bottom of the IDE window to blank.

Sub-Process Control

CAPIApplication.ExecuteCommandLine(cmd, dirname, input, timeout, env=None, bufsize=100000, return_stderr=False)

Run a command line synchronously until it completes.

cmd can either be a command line (as a string) or a list containing the executable and any arguments.

The command is run in the directory specified by **dirname**.

input is any text to send to the sub-process, or **None** to send nothing.

timeout sets the maximum number of seconds to wait for the command to complete.

Unless **env** is given, the command is run in the environment configured in the current project. Otherwise, **env** should be in the same form as **os.environ**.

The **bufsize** is used for the I/O buffer to the sub-process, as follows: If < 0, system default is used; if 0, I/O is unbuffered; if 1, line buffering is used if the OS supports it; if >1, the buffer is set to that number of bytes.

When **return_stderr** is **True**, both **stderr** and **stdout** are returned. Otherwise only **stdout** is returned.

Returns **(err, output_txt)** where **err** is one of:

```
0 -- Success
1 -- Command could not be launched
2 -- Command timed out
```

And **output_txt** is either a string containing **stdout** or, a tuple **(stdout, stderr)** (when **return_stderr** was set to **True**).

Use **AsyncExecuteCommandLine** to avoid locking up Wing while the command runs, or to access process exit status.

CAPIApplication.AsyncExecuteCommandLine(cmd, dirname, *args)

Run the given command asynchronously in the given directory.

cmd contains the executable to run, either the full path or the name on the **PATH**.

Additional command line arguments are passed as extra parameters via **args**.

Returns a handler instance that is used asynchronously to monitor the progress of the command and to obtain its output and exit status. This can be placed into a timeout function installed with **InstallTimeout**. For example:

```
handler = wingapi.AsyncExecuteCommandLine('ls', '/path/to/dir', '-al')
def watch():
  if handler.Iterate():
    stdout, stderr, err, exit_status = handler.Terminate()
```

```
    print "Done"
    print stdout
    return False
  else:
    print "Start time (relative to time.time()):", handler.time
    print "Iteration #", handler.count
    print "Output so far: %i characters", len(handler.stdout)
    return True
wingapi.InstallTimeout(500, watch)
```

Be sure that the timeout function returns **True** until the handler has completed, so that the timeout is called again and **Terminate** is eventually called.

The return values from **Terminate** are as follows:

```
stdout      -- The text received from child process stdout
stderr      -- The text received from child process stderr
err         -- Error code if execution failed, or ``None`` on success
exit_status -- Exit status of the child process, or ``None`` if it was
               never launched or could not be determined
```

The environment specified in the project is used for the sub-process. Use **AsyncExecuteCommandLineE** to specify a different environment.

To send input to the sub-process, use **handler.pipes.tochild.write()** optionally followed by **handler.pipes.tochild.flush()**. When I/O to the child is complete, call **handler.pipes.tochild.close()**. Some processes wait for the pipe to be closed before continuing.

CAPIApplication.AsyncExecuteCommandLineE(cmd, dirname, env, *args)

Same as **AsyncExecuteCommandLine** but accepts also the environment to send into the debug process.

CAPIApplication.AsyncExecuteCommandLineEB(cmd, dirname, env, bufsize, *args)

Same as **AsyncExecuteCommandLineE** but accepts also the I/O buffer size: if < 0, system default is used; if 0, I/O is unbuffered; if 1, line buffering is used if the OS supports it; if >1, the buffer is set to that number of bytes. To pass the project-defined environment to this call, use **CAPIProject.GetEnvironment**.

Sub-Process Control with OS Commands

CAPIApplication.AddOSCommand(cmd, dirname, env, flags, *args)

Add the given command line to the **OS Commands** tool. The **cmd** can be the whole command line as a list, or just the executable if its arguments are passed through **args**.

dirname can be **None** to indicate using the project-defined default starting directory.

env can be **None** to use the project defaults, or a dictionary to add values to the project-defined environment.

flags is a dictionary containing zero or more of the following:

title: The display title for the command Default: same as **cmd** argument.

hostname: The name of the configured remote host to run on, or **"** to indicate local host. Default: **None** which indicates that project settings will be used.

io-encoding: Encoding name for I/O (such as utf-8). Default: **None**

key-binding: Textual representation of key binding to assign (such as "Ctrl-X Ctrl-Shift-T"). Default: **None**

raise-panel: **True** to raise the OS Commands tool when this command is executed. Default: **True**

auto-save: **True** to auto-save files before executing the command. Default: **False**

pseudo-tty: **True** to use a Pseudo TTY for the command. Default: **False**

line-mode: **True** to set buffering to line mode. Default: **False**

Returns the internal command ID for the added command.

This adds a **Command Line** style OS Command. Adding **Python File** and **Named Entry Point** style OS Commands is not supported by the API.

CAPIApplication.RemoveOSCommand(cmd_id)

Remove an OS Command.

cmd_id is the internal command ID returned from **AddOSCommand**.

The command is terminated first if it is currently running.

CAPIApplication.ExecuteOSCommand(cmd_id, show=True)

Execute the given command in the **OS Commands** tool, using the internal command ID returned from previous call to **AddOSCommand**.

If **show** is True then the **OS Commands** tool will be shown. Otherwise, the tool is shown only if the command was configured to always show the tool when executed.

CAPIApplication.TerminateOSCommand(cmd_id)

Terminate an OS Command if it is currently running.

cmd_id is the internal command ID returned from **AddOSCommand**.

Scripting Framework Utilities

CAPIApplication.ReloadScript(module)

Reload the script file(s) associated with the given module or filename.

CAPIApplication.EnablePlugin(plugin_id, enable)

Enable to disable a plugin.

This may be called from plugins that auto-enable in response to signals, to indicate whether the plugin should be active or not.

Note that the user can override the plugin-determined state to either set a plugin as always enabled or never enabled, either in preferences or in project properties.

Returns **True** if the plugin was enabled, False if not.

18.7.3. API Reference - Editor

API support for the editor has two parts:

(1) CAPIDocument is used to access the buffer that contains the text for one or more editors. Multiple editors may share a single buffer, and buffers are also used for search or source analysis operations.

(2) CAPIEditor is used to access a single editor in the user interface, with a single file open in it. Each editor tab in Wing is a separate editor.

Class CAPIDocument

API to access an open editor document. This class should not be instantiated directly. Use the methods on **CAPIApplication** and **CAPIEditor** instead.

A single document may be shared by multiple open editors, and/or search and static code analysis tasks.

Signals

A callback can be connected to the following signals using **Connect(signal, cb)**:

> **destroy**: The document is closing. Calls cb(doc:CAPIDocument).

> **modified**: The document's text has been modified. Calls **cb(insert:bool, pos:int, length:int, text:str, lines_added:int)** where:

>> **insert** is **True** if text was inserted and **False** if text was deleted.

>> **pos** is the position of the change.

>> **length** is the length of text affected.

>> **text** is the text that was inserted or deleted.

>> **lines_added** is the number of lines added.

> **presave**: The document is about to be saved to disk. Calls **cb(filename:str, encoding:str)** where **filename** and **encoding** are **None** if the document-specified location and encoding will be used. The callback may make changes to the buffer if desired, though this is best avoided if **filename** is not **None**.

save-point: The document has entered or left a save point, where it matches the copy that was read from or written to disk. Calls **cb(save_point:bool)** where **save_point** is True if a save point was reached and **False** if leaving the save point.

filename-changed: The filename for this document has changed. Calls **cb(old_name:str, new_name:str)** where **old_name** and **new_name** are full paths.

Use **Disconnect(signal_id)** to preemptively disconnect a signal handler, where **signal_id** is the signal ID previously returned from **Connect**.

General Access

CAPIDocument.GetMimeType()

Get the mime type for this document, as determined by file name, contents, and **Files > File Types > Extra File Types** preference.

CAPIDocument.GetFilename()

Get the file name this document. For untitled or scratch buffers, the file name is prefixed with **unknown:**. For remote files, the file name will be a URL. Use **IsUrl** to distinguish between file names and URLs.

CAPIDocument.GetEditors()

Get all existing editors for this document. This may be an empty list if the document is only open for searching or static analysis.

Buffer Access

CAPIDocument.GetText()

Get the document's contents as a utf-8 encoded string

CAPIDocument.SetText(txt)

Set the document contents, replacing any existing content. The **txt** must be either a unicode string or utf-8 encoded text.

CAPIDocument.DeleteChars(start, end)

Delete characters in given range, including the character at **end**. The offsets are relative to the utf-8 encoded buffer.

CAPIDocument.InsertChars(pos, txt)

Insert characters at the given position. The **txt** must either be a unicode string or utf-8 encoded text. The offset is relative to the utf-8 encoded buffer.

CAPIDocument.GetLength()

Get the total length of document's utf-8 buffer.

CAPIDocument.GetLineCount()

Get the total number of lines in the document.

CAPIDocument.GetCharRange(start, end)

Get the text in the given range as a utf-8 encoded string. The offsets are relative to the utf-8 encoded buffer.

CAPIDocument.GetLineNumberFromPosition(pos)

Get the line number (0=first) at the given position in the utf-8 encoded buffer

CAPIDocument.GetLineStart(lineno)

Get the character position for the start of the given line number (0=first). The offset is relative to the utf-8 encoded buffer.

CAPIDocument.GetLineEnd(lineno)

Get the character position for the end of given line number (0=first). The offset is relative to the utf-8 encoded buffer.

CAPIDocument.GetAsFileObject()

Get the document's contents in a file-like object

Undo/Redo

CAPIDocument.BeginUndoAction()

Mark the start of an undoable action group. All edits between this call and **EndUndoAction** will be undone in a single **undo** operation.

It is critical to call **EndUndoAction** at the end of the action or the user will experience undos that span many more edits than intended. Use try/finally to guarantee this a follows:

```
doc.BeginUndoAction()
try:
  # edits here
finally:
  doc.EndUndoAction()
```

CAPIDocument.EndUndoAction()

Mark the end of an undoable action group.

CAPIDocument.CanUndo()

Check whether undo is available.

CAPIDocument.CanRedo()

Check whether redo is available.

CAPIDocument.Undo()

Undo one edit action in the document.

CAPIDocument.Redo()

Redo edits previously undone with **Undo**.

Saving

CAPIDocument.Save(filename=None)

Save the document to disk. If a file name is given, a copy is saved there without altering the document's primary file.

CAPIDocument.IsSavePoint()

Check whether the buffer matches its file on disk. Returns **True** if it does.

Class CAPIEditor

API to access an editor. This class should not be instantiated directly. Use the methods on **CAPIApplication** instead.

Signals

A callback can be connected to the following signals using **Connect(signal, cb)**:

destroy: The editor has been destroyed. Calls **cb(editor:CAPIEditor)**.

selection-changed: The current selection has changed. Calls **cb(start:int, end:int)** with the new selection, relative to the utf-8 contents of the **CAPIDocument**.

selection-lines-changed: The starting and/or ending line for the selection has changed. In Python files this is *not* emitted when the selection moves to a new physical line within the same logical line of code. Calls **cb(first_line, last_line)** with the new first and last lines (0=first line in file).

scrolled: The editor view has scrolled. Calls **cb(top_line)** with the new first visible line (0=first line in file).

visible-lines-changed: The range of visible lines has changed. Calls cb(top_line, **bottom_line)** with the new top and bottom lines (0=first line in file).

readonly-edit-attempt: An edit was attempted and rejected on a readonly file. **Calls cb()** without arguments.

data-entry-stopped: Data entry mode has stopped. Calls **cb(data_entry_id)** where **data_entry_id** is the ID returned from **StartDataEntry**.

Use **Disconnect(signal_id)** to preemptively disconnect a signal handler, where **signal_id** is the signal ID previously returned from **Connect**.

General Access

CAPIEditor.IsReadOnly()

Check whether the editor is readonly. Returns **True** or **False**.

CAPIEditor.SetReadOnly(readonly)

Set whether or not the editor is read-only. **readonly** should be **True** or **False**.

CAPIEditor.GetDocument()

Get the **CAPIDocument** object being shown in this view.

Selections

CAPIEditor.GetSelection()

Get **(start, end)** for the selection on the editor. **start** is always less than **end**. The offsets are relative to the utf-8 encoded text in the editor's **CAPIDocument**.

CAPIEditor.GetSelectedDottedName()

Get **(dotted_name, lineno)** for the current selection on the editor. The dotted_name may be a simple symbol like 'text', an expression like 'modname.classname.attrib', or None if no dotted name is found at the current selection position.

CAPIEditor.GetAnchorAndCaret()

Get the current selection anchor and caret position. The anchor may come after the caret position if the user has selected backwards in the text. The offsets are relative to the utf-8 encoded text in the editor's **CAPIDocument**.

CAPIEditor.SetSelection(start, end, expand=1)

Set the selection on the editor, optionally expanding any folded parts to show the selection. **start** is the selection anchor and **end** is the caret position. The anchor can be before or after the caret. Does not alter scroll position. Offsets are relative to the utf-8 encoded text in the editor's **CAPIDocument**.

CAPIEditor.GetClickLocation()

Get the offset in the utf-8 encoded text buffer for the last mouse click on the editor.

CAPIEditor.GetSourceScope()

Get the current source scope, based on position of selection or insertion caret.

Returns **None** if nothing is selected or a list that contains a filename, a line number (0=first), and zero or more source symbol names indicating the nested scope that the user has selected.

For example:

```
["/x/y/z.py", 120, "Class1", "Method1"]
```

For untitled or scratch buffers, the file name is prefixed with **unknown:**. For remote files, the file name will be a URL. Use **IsUrl** to distinguish between file names and URLs.

Scrolling and Visual State

CAPIEditor.GetFirstVisibleLine()

Get the line number of the first visible line (0=first in file) on screen in this editor.

CAPIEditor.GetNumberOfVisibleLines()

Get the number of visible lines on screen for this editor.

CAPIEditor.ScrollToLine(lineno, select=0, pos='slop', store_history=1, callout=0)

Scroll so that given line (0=first) or selection is visible in the editor.

select can be one of:

> **0** to make no changes in selection.

> **1** to select the whole line.

> **2** to place the caret at the start of the line.

> **(start, end)** to select the given character range, relative to the utf-8 buffer for the editor. In this case, **lineno** may be set to **-1** to compute the line number from the selection.

pos can be one of:

> **slop** to ensure visibility without a specific position.

> **center** to always center the line on the display.

> **top** to always position the line at the top of the display.

> Only **center** and **top** will work if the editor has not yet been shown.

Set **store_history** to **False** to avoid remembering the current editor position in the visit history.

Set **callout** to **True** to briefly display a callout to highlight the given text selection.

CAPIEditor.GetVisualState(errs=[], style='all')

Get the current visual state of the editor, including scroll position, selection, and so forth.

The **style** of the state may be one of:

> 'all' to capture all of the application visual state

> 'tools-and-editors' to capture which tools are visible, the overall layout of the windows, and which editors are open (but not details like scroll positions, selection, or current search string)

> 'tools-only' to capture only which tools are visible and the overall layout of the windows (but not which editors are open).

Returns an opaque dictionary with the state, for later use with **SetVisualState**.

Any errors encountered are added to **errs** as strings.

CAPIEditor.SetVisualState(state)

Restore a visual state previously obtained with **GetVisualState**.

Folding

CAPIEditor.FoldingAvailable()

Check whether folding is available and enabled on this editor.

CAPIEditor.FoldUnfold(fold_check_cb)

Folds or unfolds all the fold points, as determined by the given call back, which returns 1 to expand, 0 to collapse, and -1 to leave a fold point untouched. If the callback is not a callable, all the folds are either expanded or collapsed according to the value of **bool(fold_check_cb)**.

Returns a list of lineno's that were folded and a list of line numbers (0=first) that were expanded.

Indentation

CAPIEditor.GetTabSize()

Get the effective tab size for this editor, as determined by the contents of the file and indentation preferences.

CAPIEditor.GetIndentSize()

Get the indent size for this editor, as determined by the contents of the file and indentation preferences.

CAPIEditor.GetIndentStyle()

Get the predominant indent style used in this file. Returns one of:: 1 -- spaces only 2 -- tabs only 3 -- mixed tabs and spaces

CAPIEditor.SetIndentStyle(style)

Set the indent style to use in this editor. This should only be used on an empty file or to force indent style regardless of existing file content (not a good idea with Python files).

CAPIEditor.GetEol()

Get one end-of-line that matches the content of this editor. Returns one of: **"\n"**, **"\r"**, or **"\r\n"**.

Command Execution

These methods execute editor commands documented in the **Active Editor Commands** section of the Editor Commands reference."""

CAPIEditor.CommandAvailable(cmd_name, **args)

Check the whether an editor command is available for execution with the given arguments. Arguments may be omitted if they don't affect command availability, which most don't.

The available commands for an editor are documented in the **Active Editor Commands** section of the Editor Commands reference.

CAPIEditor.ExecuteCommand(cmd_name, **args)

Execute the given command in the editor. Any command arguments are passed on the command line via **args**. This is used to execute commands in an editor even if it does not have focus.

The available commands for an editor are documented in the **Active Editor Commands** section of the Editor Commands reference.

Snippets and Data Entry mode

Class CAPIEditor.CAPIFieldMetaData

Stores meta data for fields used with the **meta_data** argument for **StartDataEntry** and **PasteSnippet**, in order to control how fields are filled and visited.

Available keywords arguments for the constructor are:

auto_enter_from specifies the field index (0=first) from which data for this field should be auto-entered, rather than allowing the user to type into the field. This is used for fields that appear several times. Default: **-1**, which indicates no auto-entering for the field.

force_tab_stop may be set to **True** to force including the field as a tab stop even if it is auto-entered from another field. This allows the user to change the auto-entered value. Default: **False**

skip_tab_stop is set to **True** to skip this field when traversing fields. This is useful for placing a marker that is tracked during editing but not visited as a tab stop. Default: **False**

For example to auto-enter a value from field **3**:

```
meta = CAPIFieldMetaData(auto_enter_from=3, force_tab_stop=True)
```

CAPIEditor.PasteSnippet(txt, fields, auto_terminate=False, meta_data={})

Paste a utf-8 text snippet into the editor and place the editor into inline data entry mode. Snippet syntax is documented in Snippet Syntax.

The **txt** is the text to paste.

fields provides the **(start, end)** offsets within that text for fields the user can enter or alter.

Set **auto_terminate** to stop data entry mode when the last field is reached.

To control behavior of the fields, set **meta_data** to a dictionary from field index (0=first) to **CAPIEditor._CAPIFieldMetaData**.

CAPIEditor.StartDataEntry(fields, active_range=(0, -1), goto_first=True, auto_terminate=False, meta_data={})

Start inline data entry mode so the user can use the **Tab** and **Shift-Tab** keys to move between data fields inline in the editor.

fields is a list of **(start, end)** positions where the fields are located, in tab traversal order.

active_range indicates the range of text within which the data mode will exist. Data entry terminates if the caret moves outside of this range or if the user presses **Esc**. The default range of **(0, -1)** indicates the entire document.

When **goto_first** is set, the first field in the tab sequence will become the current selection.

Set **auto_terminate** to stop data entry mode when the last field is reached.

To control behavior of the fields, set **meta_data** to a dictionary from field index (0=first) to **CAPIEditor._CAPIFieldMetaData**.

Returns **None** if data entry failed to start or otherwise a unique ID for the data entry action.

This may be invoked recursively so that another data entry action is used to fill in a field of a previously created data entry action.

CAPIEditor.StopDataEntry()

Exit inline data entry mode. If **StartDataEntry** was recursively invoked then the innermost data entry action is exited.

CAPIEditor.ActiveDataEntry()

Get the id of the active data entry action, or **None** if there is none.

Utilities

CAPIEditor.GrabFocus()

Set keyboard focus on this editor.

CAPIEditor.SendKeys(keys)

Send a string of one or more keys to the editor so they are processed as if they were typed by the user. Key processing includes any auto-editing, auto-indentation, etc.

18.7.4. API Reference - Project

Class CAPIProject

API to access the project. This class should not be instantiated directly. Use **CAPIApplication.GetProject()** instead.

Signals

A callback can be connected to the following signals using **Connect(signal, cb)**:

destroy: The project is closing. Calls **cb(proj:CAPIProject)**.

files-added: Files have been added. Calls **cb(filenames)** where **filenames** is a list of full paths. File names starting with **ssh:** are on a remote host. Use **IsUrl** to distinguish between urls and local file names.

files-removed: Files have been removed. Calls cb(filenames) where **filenames** is the same as for the **files-added** signal above.

Use **Disconnect(signal_id)** to preemptively disconnect a signal handler, where **signal_id** is the signal ID previously returned from **Connect**.

Project Contents

CAPIProject.GetAllFiles()

Returns a list of all the full path filenames in this project.

File names starting with **ssh:** are on a remote host. Use **IsUrl** to distinguish between urls and local file names.

CAPIProject.GetAllDirectories(top_only=False)

Get list of full path names for all directories in this project.

If **top_only** is **True**, only top-level directories are returned.

Directory names starting with **ssh:** are on a remote host. Use **IsUrl** to distinguish between urls and local file names.

CAPIProject.AddFiles(files)

Add the files with given full path filenames to the project.

CAPIProject.RemoveFiles(files)

Remove the given full path filenames from the project.

CAPIProject.AddDirectory(dirname, recursive=True, filter='*', include_hidden=False, watch_disk=True, excludes=())

Add the given directory to the project, given its full path name.

When **recursive** is **True**, all children, grand-children, etc, are also added.

The **filter** specifies which files to display.

Set **include_hidden** to **True** to show also hidden files like **.name**, ***pyc**, and ***~**.

Set **watch_disk** to watch the disk for changes and update the **Project** tool accordingly.

Set **excludes** to a list of relative path names from **dirname** for files to explicitly exclude from the display.

If the directory is already in the project this call will replace it properties according to the arguments.

CAPIProject.RemoveDirectory(dirname)

Remove the given directory, and any recursively added sub-directories, from the project.

Project Properties

CAPIProject.GetEnvironment(filename=None, set_pypath=True, overrides_only=False)

Get the runtime environment for the given debug file in the context of this project. This is determined by overriding the environment inherited at startup with any values set in **Project Properties** and **File Properties**.

If the given **filename** is **None**, only the project-wide settings are used.

If a **Python Path** is set in **Project Properties** and **set_pypath** is True, it is added to the environment as **PYTHONPATH**, overwriting any **PYTHONPATH** in the inherited environment.

When **overrides_only** is **True**, this call only returns the environment that is configured for the given file (or project-wide if **filename** is **None**) and not inherited environment values. This result can be used as the basis for calling **SetEnvironment**.

CAPIProject.ExpandEnvVars(txt, filename=None)

Expand **$(envname)** and **${envname}** style environment variables within the given text in the context of the environment returned by **GetEnvironment(filename, set_pypath=False)**.

CAPIProject.SetEnvironment(filename, base, env={})

Set the runtime environment for debugging or executing the given file or for the project as a whole if filename is **None**.

The argument **base** indicates which base environment the given environment should modify:

> **'startup'**: Modify startup environment

> **'project'**: Modify the project environment (not a valid value when **filename** is **None**)

In either case, the given **env** is applied to the selected base environment by removing any keys with empty values and adding/updating any keys with non-empty values. If the order of the environment keys is important, use collections.OrderedDict for the value of this argument.

If **PYTHONPATH** is included in the environment, it is stored in (or cleared from) the **Python Path** property in **Project Properties** or **File Properties** and not the **Environment** property.

Using **SetFileLaunchConfig** and related API is preferable when **filename** is not **None**.

CAPIProject.GetPythonExecutable(filename)

Get the Python executable set for the given file or the project as a whole if the **filename** is **None**.

Returns **None** if using the inherited value, which is the project-wide value for a file, and the default found Python for a project (when **filename** is **None**). The default Python can be determined with **CAPIApplication.FindPython**.

GetFileLaunchConfig and related API is preferable when **filename** is not **None**.

The executable can be on a remote host, in which case this function returns a value in the form **ssh://hostname/** where **hostname** is the **Identifier** of the Remote Host that specifies which Python to use.

The executable may be a command that activates a virtualenv or other environment. In this case, the value returned is in the form **env://command_line**.

If executable includes arguments, spaces within arguments are managed by escaping them with or by delimiting arguments with quotes.

CAPIProject.SetPythonExecutable(filename, executable)

Set the Python executable to use when debugging or executing the given file. The filename may be **None** to set the project-wide **Python Executable**. The executable may be **None** to use the project-wide value or default found Python.

The executable can be a url in the form **ssh://hostname/** (without any additional url path) to use the Python specified by the Remote Host for **hostname**. In this case, if a non-**None** filename is given it must also be a file on that remote host, using a url in the form **ssh://hostname/path/to/file.py**.

The executable may be a command that activates a virtualenv or other environment, in which case it is in the form **env://command_line**.

If the executable includes arguments, spaces within the command line must be managed by escaping them with or delimiting arguments with quotes.

Using SetFileLaunchConfig() and related API is preferable when **filename** is not **None**.

CAPIProject.GetPythonExecutableProperties(executable)

Get information about the given Python executable. The argument is in the same format as for SetPythonExecutable.

On success, the return value is a dictionary containing the following values:

> fullpath : The full path to the interpreter's 'python.exe' or 'python' version : The Python version in #.#.# form prefix : The value of sys.prefix base_prefix : The value of sys.base_prefix pypath : The builtin Python Path (an item set to None indicates current directory) keywords : The keywords in this Python version builtins : The builtins in this Python version

Each of these values is None if the executable was not found or is invalid.

If the executable is on a remote host and the interpreter has not yet been inspected then None is returned instead of a dictionary. In this case, a remote inspection is launched and this method may be called again later to obtain the values.

CAPIProject.GetInitialDirectory(filename)

Get the initial directory for debugging the given file. The **filename** may be **None** to set the project-wide setting. Returns **None** when using the startup directory.

Using **GetFileLaunchConfig** and related API is preferable when **filename** is not **None**.

CAPIProject.SetInitialDirectory(filename, dirname)

Set the initial directory to use when debugging the given file. The **filename** may be **None** to set the project-wide setting. The **dirname** may be **None** to use the startup directory.

Using **SetFileLaunchConfig** and related API is preferable when **filename** is not **None**.

CAPIProject.GetMainEntryPoint()

Get the full path for the main entry point for this project. This returns a string in the form **"entry:name"** if the main entry point is a Named Entry Point, or **None** if there is none, in which case debugging and execution start with the current editor file.

CAPIProject.SetMainEntryPoint(filename)

Set the main entry point for this project. Pass in a full path, a string in the form **"entry:name"** to use a Named Entry Point, or **None** to unset the main entry point so that debugging and execution start with the current editor file.

CAPIProject.SetDebugChildProcesses(enable=True)

Enable or disable child process debugging in this project. Set **enable** to **True** to always debug child processes, **False** to never debug child processes, and **None** to refer to the **Debugger > Processes > Debug Child Processes** preference.

Launch Configurations

CAPIProject.GetLaunchConfigs()

Get a list of the internal IDs for all the defined Launch Configurations in the project.

CAPIProject.CreateLaunchConfig()

Create a new Launch Configuration. Returns the launch configuration's internal ID.

CAPIProject.GetLaunchAttrib(launch_id, attrib, include_hostname=False)

Get a Launch Configuration attribute.

launch_id is the internal launch configuration ID.

attrib is the attribute to get, which may be one of the following. The return value varies in type, according to which attribute was retrieved:

name: The display name of the launch configuration.

runargs: The run arguments as a string.

rundir: **(which, value)** where **which** is one of **'project'** to use the project-defined value, **'default'** to use the directory of the file being launched, or **'custom'** to use the specified string **value**.

env: **(which, env)** where **which** is one of **'project'** to use the project-defined value, **'merge'** to use env to add/remove from the project-defined value, **'default'** to use the startup environment, or **'custom'** to use **env** to add/remove from the startup environment. **env** is a list of **var=value** strings where **value** can be blank to remove the named **var** from the modified environment.

buildcmd: **(which, value)** where **which** is one of **'project'** to use the project-defined value, **'default'** to use no build command, or **'custom'** to use the specified value, which is the OS Command internal ID for a build command defined with **CAPIApplication.AddOSCommand** or by the user in the **OS Commands** tool.

pyexec: **(which, pyexec)** where **which** is one of **'default'** to use the project-defined value, or **'custom'** to use given **pyexec** string. When arg **include_hostname** is **True**, this is instead in the form **(which, (hostname, pyexec))** where hostname is **''** to indicate localhost.

pypath: **(which, value)** where **which** is one of **'project'** to use the project-defined value, **'default'** to use the startup value, or **'custom'** to use the specified **value**, which is a list of strings.

pyrunargs: **(which, value)** where **which** is one of **'project'** to use the project-defined value, **'default'** to use **'-u'**, or **'custom'** to use the specified string value

shared: **True** or **False**, to indicate whether the launch configuration is shared between all projects.

Any string value can contain environment variable references in the form **${ENVNAME}** or **$(ENVNAME)**.

Raises **KeyError** if the specified launch configuration does not exist.

Compatibility note:

In Wing 6+ the pyexec attribute stored by Wing internally changed from **(which, pyexec**) to **(which, (hostname, pyexec))** where hostname is **''** for localhost. Set **include_hostname=True** to receive that form in the return value.

In Wing 6+ the pypath attribute stored by Wing internally changed from a string with **os.pathsep** delimiter to a list of strings.

CAPIProject.SetLaunchAttrib(launch_id, attrib, value)

Set a single Launch Configuration attribute.

launch_id is the internal launch configuration ID to modify.

See **GetLaunchAttrib** for the valid attribute names and values. It is up to the caller to validate the values specified.

Raises **KeyError** if the specified launch configuration does not exist.

CAPIProject.DeleteLaunchConfig(launch_id)

Delete the given Launch Configuration. Any files or Named Entry Points that reference the configuration will revert to using the project-defined environment.

CAPIProject.GetFileLaunchConfig(filename)

Get the internal ID of the Launch Configuration used by default with the given filename. Returns **None** if the project-wide configuration is being used.

CAPIProject.SetFileLaunchConfig(filename, launch_id)

Set the default Launch Configuration to use with the given file.

filename is the full path of the file or a URL in form **ssh://hostname/path/to/file.py** for remote files, where **hostname** is the **Identifier** of a Remote Host.

launch_id is the internal launch configuration ID.

CAPIProject.ClearFileLaunchConfig(filename, runargs)

Clear the default Launch Configuration for the given file so that launching the file will use the project-defined environment and the given **runargs**.

filename is the full path of the file or a URL in form **ssh://hostname/path/to/file.py** for remote files, where **hostname** is the **Identifier** of a Remote Host.

runargs are the run arguments to use, as a string.

Named Entry Points

CAPIProject.GetNamedEntryPoints()

Get a list of names for all the defined Named Entry Points in the project.

CAPIProject.CreateNamedEntryPoint(name)

Create a new Named Entry Point. Raises **KeyError** if the name already exists.

CAPIProject.GetNamedEntryPointAttrib(name, attrib)

Get an attribute for the given Named Entry Point. The valid attribute names are as follows. The return value varies in type, according to which attribute is being retrieved:

filename: The Python file to launch. This is the full path or a URL in the form **ssh://hostname/path/to/file.py** for remote files where **hostname** is the **Identifier** of a Remote Host.

runargs: The command line arguments to use when the named entry point's **launch-id** is **None**. This valuecan contain environment variable references in the form **${ENVNAME}** or `` $(ENVNAME)``.

launch-id: The internal ID of the Launch Configuration to use, or **None** to use the project-defined environment with the command line arguments in the **runargs attribute**.

key-binding-debug: The key binding for debugging the Launch Configuration. See Key Names for details on valid key names.

key-binding-execute: The key binding for executing the Launch Configuration.

auto-show: **True** when the named entry point dialog should be shown automatically before launching.

Raises **KeyError** if the named entry point does not exist.

CAPIProject.SetNamedEntryPointAttrib(name, attrib, value)

Set a Named Entry Point attribute. **name** is the entry point's name.

See **GetNamedEntryPointAttrib** for the valid attribute names and values. It is up to the caller to validate the values specified.

Raises **KeyError** if the named entry point does not exist.

CAPIProject.DeleteNamedEntryPoint(name)

Delete the given Named Entry Point.

Attributes

Use these to store information in a project file.

CAPIProject.GetAttribute(attrib_name, filename=None)

Get the value for given named attribute previously set with **SetAttribute**.

If **filename** is not **None**, the attribute is a per-file attribute for the given file. Otherwise, it's a project-wide attribute.

Raises **KeyError** if the attribute is not defined.

CAPIProject.SetAttribute(attrib_name, value, filename=None)

Set value for the given attribute. This is used to store data in the project file. Attributes may either be associated with the project as a whole or with a particular file.

attrib_name is the attribute name, which can be any string containing letters, numbers, and dashes.

The **attrib_name** is uniquified internally to avoid conflicts between scripts. If this is not desired, so that other scripts can also access the attribute, prefix the **attrib_name** with '.'.

If **filename** is not **None**, the attribute is a per-file attribute. Otherwise, it's a project-wide attribute.

Utilities

CAPIProject.GetFilename()

Gets the filename where the project is stored, as a full path. Returns the ***.wpr** file's name. If the project is a shared project, a file ***.wpu** in the same directory will also exist and contain user-specific project state.

For untitled or scratch buffers, the file name is prefixed with **unknown:**. For remote files, the file name will be a URL. Use **IsUrl** to distinguish between file names and URLs.

CAPIProject.GetSelectedFile()

Returns the full path filename of the currently selected file on the **Project** tool, or **None** if there is no selection.

File names starting with **ssh:** are on a remote host. Use **IsUrl** to distinguish between urls and local file names.

Deprecated Methods

CAPIProject.GetRunArguments(filename)

Get the run arguments for debugging the given file, or **''** if there are none. The **filename** should not be **None**.

Deprecated in Wing 7.0. Use **GetFileLaunchConfig`** and related API instead.

CAPIProject.SetRunArguments(filename, args, add_recent=True)

Set the run arguments (as a string) for debugging the given file. Use **None** for no arguments.

See **add_recent** to **False** to prevent adding the arguments to the recent arguments list.

Deprecated in Wing 7.0. Use **SetFileLaunchConfig**, **ClearFileLaunchConfig**, and related API instead.

18.7.5. API Reference - Debugger

The debugger API consists of two parts:

(1) CAPIDebugger is the debug manager, which is used to manage multiple debug processes.

(2) CAPIDebugRunState is used to start, control, inspect, and terminate a single debug process.

Class CAPIDebugger

API for the debugger as a whole. This class should not be instantiated directly. Use **wingapi.gApplication.GetDebugger()** instead.

Signals

A callback can be connected to the following signals using **Connect(signal, cb)**:

> **new-runstate**: A new runstate has been created. Calls **cb(runstate:CAPIDebugRunState)**.

> **current-runstate-changed**: A new runstate has been selected as the current runstate. Calls **cb(runstate:CAPIDebugRunState)**.

Use **Disconnect(signal_id)** to preemptively disconnect a signal handler, where **signal_id** is the signal ID previously returned from **Connect**.

CAPIDebugger.GetRunStates()

Get the list of all **CAPIDebugRunState** objects that currently exist in the debugger.

CAPIDebugger.GetCurrentRunState()

Get the currently active **CAPIDebugRunState**.

CAPIDebugger.SetCurrentRunState(rs)

Set the currently active **CAPIDebugRunState**.

Class CAPIDebugRunState

API to access an individual debug process. This class should not be instantiated directly. Use the methods on **CAPIDebugger** instead.

Each run state is associated with a single debug process. It is created before any debug process is started, takes care of starting and controlling individual debug sessions, and outlives individual debug process termination in order to support subsequent inspection or launch of a new debug process.

Signals

A callback can be connected to the following signals using **Connect(signal, cb)**:

> **debug-started**: A debug session has been started. Calls **cb()**.

> **debug-terminated**: The debug session has ended. Calls **cb()**.

> **exception**: The debug process has encountered an exception. Calls **cb()**.

> **paused**: The debug process has paused or reached a breakpoint. Calls **cb()**.

> **running**: The debug process has started running again. Calls **cb()**.

Use **Disconnect(signal_id)** to preemptively disconnect a signal handler, where **signal_id** is the signal ID previously returned from **Connect**.

Starting and Stopping Debug

CAPIDebugRunState.Run(filename, stop_on_first=0, launch_id=None)

Start debug, using the given file as the main entry point.

filename is the full path of the file to debug. For remote files, **filename** is in the form **ssh://hostname/path/to/file.py** where **hostname** is the **Identifier** of a Remote Host.

Set **stop_on_first** to stop immediately on the first line of code. Otherwise debugging continues until it reaches a breakpoint, exception, or program termination.

Set **launch_id** to the internal ID of a Launch Configuration to use for the debug environment, or to **None** to use the file's default environment configured in **Project Properties** or **File Properties**.

CAPIDebugRunState.RunNamedEntryPoint(name, stop_on_first=0)

Run the given Named Entry Point with its configured launch environment.

Set **stop_on_first** to stop immediately on the first line of code. Otherwise debugging continues until it reaches a breakpoint, exception, or program termination.

CAPIDebugRunState.Kill()

Stop debugging by terminating the debug process. If the debug process was launched by Wing, all its child processes are also terminated.

Flow Control

CAPIDebugRunState.Step(over=1, out=0)

Step in the code, either into, over, or out of the current execution point, as follows:

- If **out** is True then step out
- If **over** is **CAPIDebugRunState.kStepOverInstruction** step over the current instruction
- If **over** is **CAPIDebugRunState.kStepOverLine** step over the current physical line
- If **over** is **CAPIDebugRunState.kStepOverStatement** step over the current statement
- If **over** is **CAPIDebugRunState.kStepOverBlock** step over or finish the current block
- If **over** is a **(start_line, end_line)** tuple, step until debugging leaves that range of lines (0=first line)
- In all other cases, step in

CAPIDebugRunState.RunToCursor()

Run until the current editor caret location is reached, or to the next breakpoint, exception, or program termination if the caret's location is not reached first.

CAPIDebugRunState.Continue()

Contine running the debug process to the next breakpoint, exception, or termination.

Threads and Stacks

CAPIDebugRunState.GetThreads()

Get a list of **(thread_id, name, running)** for the active debug process, where **thread_id** is the thread ID, **name** is the thread function name, and **running** is **True** if the thread is running or **False** if the thread is paused a breakpoint or exception.

Returns **None** if no thread in the process is paused at a breakpoint or exception.

The currently selected thread can be determined by calling **GetStackFrame**.

CAPIDebugRunState.GetStackIndex()

Get **(thread_id, stack_index)** where **thread_id** is the currently selected thread id and **stack_index** is **(stack_no, frame_idx)** where **stack_no** is the stack number (0=primary stack) and **frame_idx** is the frame index (0=outermost frame).

If the thread is still running then **thread_id** and **stack_index** will both me **None**.

Stacks 1+ are PEP 3134 chained exception stacks, in order of the chain.

CAPIDebugRunState.SetStackFrame(thread_id, idx)

Set the currently selected thread ID (**None** to use current thread) and stack index. The stack index is in the form **(stack_no, frame_idx)** to allow access to chained exception stacks. Set **stack_no** to **0** for the primary stack and **frame_idx** to **0** for the outermost frame.

Returns **(thread_id, stack_index)** where **thread_id** is the actual thread ID and **stack_index** is the stack index that was actually set. If the thread is still running then **thread_id** and **stack_index** will both me **None**.

Compatibility note:

This call changed in Wing 7.0 to support PEP 3134 chained exceptions. However, it still accepts idx set to an integer to indicate a frame in the primary stack. When idx is passed as an integer, the return value's stack index is also an integer.

CAPIDebugRunState.GetStack()

Get the stack for the currently selected debug thread as a list of frames, each of which is a tuple containing **(filename, lineno, line_text, scope, local_varnames)** where:

filename is the full path of the file, or for remote files a URL in the form **ssh://hostname/path/to/file.py** where **hostname** is the **Identifier** of a Remote Host.

lineno is the line number (0=first line) or a tuple **(start, end)** to indicate the position of the current statement in template files.

line_text is the text of the line or statement.

scope is the name of the scope for this frame (for example, **MyClass.MethodName**)

local_varnames is a list of the local variable names for the frame.

Returns **None** instead if the currently selected thread is not paused, at a breakpoint, or at an exception.

The currently selected thread is changed by calling **SetStackFrame**.

Breakpoints

CAPIDebugRunState.SetBreak(filename, lineno, temp=0, cond=None, enable=1, ignore=0)

Set a new breakpoint at the given position.

filename is the full path of the file, or for remote files a URL in the form **ssh://hostname/path/to/file.py** where **hostname** is the **Identifier** of a Remote Host.

lineno is the line number at which to set the breakpoint (0=first line)

temp is **True** to set a temporary breakpoint that will be removed the first time it is reached.

cond is a conditional string that must evaluate to **True** in the context of the breakpoint's stack frame for the breakpoint to stop, or **None** to always stop on this breakpoint.

enable can be set to **False** to disable stopping on the breakpoint.

ignore is set to a value above **0** to ignore hitting the breakpoint that number of times before stopping on it.

If a breakpoint already exists here, it is replaced.

Returns **(lineno, err)** where **lineno** is the actual line the breakpoint was placed at and **err** is either **None** or an error string.

CAPIDebugRunState.ClearBreak(filename, lineno)

Clear a breakpoint.

filename is the full path of the file, or for remote files a URL in the form **ssh://hostname/path/to/file.py** where **hostname** is the **Identifier** of a Remote Host.

lineno is the breakpoint's line numbert (0=first line)

CAPIDebugRunState.ClearAllBreaks()

Clear all breakpoints.

Utilities

CAPIDebugRunState.GetProcessID()

Get the process ID of the active debug process. Returns **None** if there is no active process.

CAPIDebugRunState.GetStatus()

Get the status of the debug process. Returns an integer, as follows:

~~

0 -- disconnected (no debug process) 1 -- listening for a connection from an IDE-launched debug process 2 -- connected to a debug process 3 -- the debug process is running 4 -- the debug process is stopped at a breakpoint or paused 5 -- the debug process is stopped on an exception 6 -- listening for a connection from a externally launched debug process

Deprecated Methods

CAPIDebugRunState.GetStackFrame()

Get the currently selected thread ID (None if no thread is paused or stopped at a breakpoint or exception) and stack frame (0=outermost frame, or **None** if the thread is still running).

When there are chained exception stack, the index is in the currently selected stack. Use GetStackIndex() to get the stack number as well.

Deprecated in Wing 7.0. Use **GetStackIndex** instead.

18.7.6. API Reference - Search

Class CAPISearch

API for searching files and directories. One instance of this class should be instantiated for each search. The arguments to the constructor are:

txt -- The text to search for (required)

match_case -- True for case-sensitive search (default=True)

whole_words -- True to match only whole words (default=False)

omit_binary -- True to omit files that appear to be binary files (default=True)

search_styles -- One of 'text' for plain text search, 'wildcard' for wild card matches (unix glob style matching), and 'regex' for regular expression matching (default='text')

include_linenos -- True to include line numbers in the results (when False, line numbers are not computed, which makes for faster searching) (default=False)

use_buffer_content -- True to use the content of edited buffers instead of the disk file when unsaved edits exist (default=True)

regex_flags -- For regex searches: the regex flags from the re module (default=0)

After an instance of is created, use *one* of the following to start searching:

```
SearchDirectory()
SearchFile()
SearchFiles()
```

Signals

Search results and status are reported through the following signals called asynchronously until the search completes. The can be connected to a callback with **Connect(signal, cb)**:

start -- A new search was started. Calls **cb()**.

end -- The search completed or aborted. Calls **cb()**.

match -- One or more matches have been found. Calls **cb(filename, matches)** where **filename** is the full path of the file, and **matches** is list of **(lineno, linestart, line_text, positions)** where **lineno** is the line number in the file (0=first), **linestart** is the position in the file where the line begins, **line_text** is the text for the line, and **positions** is a list of **(start, end)** tuples. The match signal may occur more than once per file or line, to report additional matches found. Line numbers are zero unless **include_lineos** was **True**. All positions are from the start of the file.

dir -- Scanning a new directory. Calls **cb(dirname:str)**.

scanning -- Scanning a new file. Calls **cb(filename)**.

file-done -- Done scanning a file. Calls **cb(filename)**.

File and directory names passed to signal handlers are full paths or for remote files and directories, in the form **ssh://hostname/path/to/item** where **hostname** is the **Identifier** of a Remote Host.

Use **Disconnect(signal_id)** to preemptively disconnect a signal handler, where **signal_id** is the signal ID previously returned from **Connect**.

Example

```
s = CAPISearch("test", match_case=False)
def match(filename, matches):
  print(filename, matches)
def end():
  print("done")
s.Connect('match', match)
s.Connect('end', end)
s.SearchFile('/path/to/myfile.txt')
```

CAPISearch.SearchDirectory(dirname, file_set, recursive)

Start searching the given directory for all files that match the file set, optionally recursively.

The **dirname** is the full path of the directory name or for remote directories in the form **ssh://hostname/path/to/dir** where **hostname** is the **Identifier** of a Remote Host.

The **file_set** can either be a name of a configured file filter stored in the 'main.file-sets' preference or **(includes, excludes)** where **includes** and **excludes** are lists of tuples **(spec_type, text)** in which **spec_type** is one of 'wildcard-filename', 'mime-type', or 'wildcard-directory' and text is the pattern to apply to the file name, mime type, or directory name in order to filter which files are searched.

For example, to search only Python files use **'Python Files'** as the filespec. Or to search ***.py** files other than those within a directory named **'test'**, use the following:

```
[(('wildcard-filename', '*.py'),), (('wildcard-directory', 'test'),)]
```

If the file filter in file_set is a string, an exception will be raised if it is not a valid file filter name.

CAPISearch.SearchFiles(files)

Start searching all the given **files**, which is a list of full path filenames or URLs in the form **ssh://hostname/path/to/file.py** where **hostname** is the **Identifier** of a Remote Host.

This accepts only filenames and not directories. Use **SearchDirectory** to search a directory.

CAPISearch.SearchFile(filename, start_pos=0)

Search a single file for search matches, optionally starting at a given point. The **filename** should be a full path or URL in the form **ssh://hostname/path/to/file.py** where **hostname** is the **Identifier** of a Remote Host.

This can also be used to re-search a file previously searched through **SearchDirectory** if the file changes.

CAPISearch.Stop()

233

Terminate searching, if a search is active.

CAPISearch.Pause()

Pause the search process.

CAPISearch.Continue()

Continue the search process after it was previously paused with **Pause**.

18.7.7. API Reference - Analysis

The static analysis API is used to inspect the structure and contents of Python files. It consists of two parts:

(1) CAPISymbolInfo is used to describe a particular source symbol.

(2) CAPIStaticAnalysis is used to inspect a particular Python file.

Class CAPISymbolInfo

API to describe the inferred type for a particular source symbol. This class should not be instantiated directly. Instances of this class are returned from **CAPIStaticAnalysis.GetSymbolInfo**.

Type information is accessed with the following instance attributes:

generalType: General type of the symbol: One of **'class'**, **'method'**, **'function'**, **'instance'**, **'keyword'**, **'literal'**, or **'module'**.

typeName: The full name of the type.

fileName: The file where the type is defined.

lineStart: The first line of the type definition (0=first line in file).

lineCount: The number of lines taken up by the type definition.

pos: The position of the type definition within the first line.

isCallable: **True** when the symbol is a callable.

args: A list of argument names, if **isCallable** is **True**.

docString: The docstring for the type.

Class CAPIStaticAnalysis

API for inspecting the contents of a Python file, based on Wing's static analysis of that file. This class should not be instantiated directly. Use **CAPIApplication.GetAnalysis** instead.

CAPIStaticAnalysis.GetScopeContents(scope, timeout=0.5)

Get a list of all the symbols defined in the given scope.

scope is the name of the scope to inspect. For example, **MyClass.MyMethod** is the method **MyMethod** in class **MyClass** and **MyClass.MyMethod.nested** is a nested function **nested** within that method.

Use **"** for the top level of the module.

To obtain attributes for an instance, append ':' to the class name. For example, 'MyClass:' provides the attributes for an instance of **MyClass**.

Set **timeout** to specify the maximum computation time in seconds.

Returns a dictionary mapping symbol names to a sequence of one or more strings describing the symbol. The descriptors may be:

```
imported -- The symbol is imported from another module
class -- The symbol is a class or class attrib when 'attrib' is also present
method -- The symbol is a method
function -- The symbol is a function
argument -- The symbol is a function or method argument
module -- The symbol is a module
package -- The symbol is a package
attrib -- The symbol is an instance attribute
```

Use **GetSymbolInfo** to obtain additional information about a symbol, including its inferred type and point of definition.

CAPIStaticAnalysis.FindScopeContainingLine(lineno)

Find the scope containing the given line number. Note that a class line or a def line is in its parent's scope.

CAPIStaticAnalysis.GetSymbolInfo(scope, symbol)

Get extended information for the given symbol within the named scope. A scope of **"** and symbol of **"** obtains type information for the module as a whole. Returns a sequence of **CAPISymbolInfo** instances.

Trouble-shooting Guide

This chapter describes what to do if you are having trouble installing or using Wing.

> ### *Note*
>
> We welcome feedback and bug reports, both of which can be submitted directly from Wing using **Submit Feedback** and **Submit Bug Report** in the **Help** menu, or by emailing us at support at wingware.com.

19.1. Trouble-shooting Failure to Start

If you are having trouble getting Wing to start at all, you can diagnose the problem as follows:

Rule out problems caused by a corrupted project file or preferences by renaming your Settings Directory. If this works, you can copy over items from the renamed directory one at a time to isolate the problem. The most likely files to cause problems are **default.wpr**, **preferences**, and **recent-projects**. Note, however, that Wing may automatically copy over files from the settings directory for an older version of Wing. You may have to move those aside also, to prevent reintroducing problem files.

Check whether anti-virus or security software is blocking Wing from starting. Some anti-virus solutions like Constant Guard have been known to do this, without showing any warnings or messages. On OS X, check the Security & Privacy system control panel for messages.

On Windows, check if the user's temporary directory is full, which prevents Wing from starting. In this case, the directory will contain more than 65,000 files.

On Linux or OS X, check if the cache directory is on a remote file server, which can prevent Wing from starting. This happens if the **~/.cache** directory or the cache directory set by the **$XDG_CACHE_DIR** is located on NFS or other remote file server. In that case, Wing can't obtain a lock on the source analysis database. To use slower dotfile locking, run Wing with the **--use-sqlite-dotfile-locking** command line argument. Note that all Wing processes that use the same cache directory need to either use or not use dotfile locking.

In other cases, refer to Obtaining Diagnostic Output.

19.2. Speeding up Wing

Wing should present a snappy, responsive user interface even on relatively slow hardware. If Wing appears sluggish, you can diagnose the problem as follows:

Wait for source analysis to complete, which may be necessary just after creating a new project, adding files to an existing project, or if many files on disk have changed or moved. In this case, the status area in the lower left of the IDE window will indicate that analysis is running. Wing stores the results of this process in a cache so the problem should not reoccur often.

Increase the source analysis cache allocation with the **Source Analysis > Max Cache Size** preference if the **Cache Directory** in Wing's **About box** exceeds that size. You may also want to press the **Clear Cache** button next to the preference to rule out problems caused by a corrupted source analysis cache.

Try disabling external change checking by setting the **Files > Reloading > External Check Freq** preference to **0**.

On a multi-core virtual machine, set processor affinity if Wing runs slowly. This is done with **schedtool -a 0x1 -e wing7.2** on Linux (the **schedtool** package needs to be installed if not already present) and with **START /AFFINITY 01 "Wing Pro" "C:\Program Files (x86)\Wing Pro 7.2\bin\wing.exe"** on Windows.

In other cases, collect a profile as follows:

- Select **Command by Name** from the **Edit** menu, type **internal-profile-start**, and press **Enter**
- Do something that is slow, or just wait for a while
- Select **Command by Name** again, type **internal-profile-stop**, and press **Enter**

The profile is written to the end of **ide.log** in the Settings Directory. This can be submitted in a bug report from the **Help** menu or by email to support@wingware.com.

19.3. Trouble-shooting Failure to Debug

If you are having trouble debugging with Wing, select whichever of the following most closely describes the problem you are seeing.

19.3.1. Failure to Start Debug

Use the following steps to diagnose failure to start debugging:

Try a simple test case using the following code in a new Python file and **Step Into** in the **Debug** menu:

```
print("test1")
print("test2")
```

This rules out unexpectedly running to completion or running into a fatal error after debug has started successfully.

Set WINGDB_PRINT_ALL_TRACEBACKS=1 in the **Environment** in Project Properties and try restarting the **Python Shell** from its **Options** menu. This is often a simple way to obtain a traceback that shows why Python is failing to start.

Verify that your Python version is supported according to Supported Python Versions. If not, you may need a different version of Wing.

Check that Python Path is valid with **Show Python Environment** in the **Source** menu. If this contains directories inside a Python version that doesn't match the interpreter being run for the

debug process, then Python will fail to start. You can set **Python Path** and **Python Executable** in Project Properties.

Check for environment conflicts, which may occur if you set the **PYTHONHOME** or **PYTHONPATH** environment variables and they do not match the particular Python interpreter that Wing is launching.

Completely remove and reinstall Python if you have installed a newer version over and older one on disk, in the same directory. This may cause debugging to fail even if other Python programs appear to work with the Python installation because the debugger used functionality that isn't used by most other code.

Confirm that TCP/IP is working on your machine since Wing's debugger uses TCP/IP to communicate with the IDE.

Disable PyGame full-screen mode and use window mode instead, since full-screen mode does not work with Python debuggers.

In other cases, collect diagnostics as described in Diagnostic Output.

19.3.2. 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 an exception is reached:

Not saving before you run in the debugger causes the debugger to run with the copy of the file that is on disk, while breakpoints are set using the edited copy of the file in the editor. If lines don't match up, then breakpoints will be missed. To avoid this problem, enable the **File > Auto-Save Files Before Debug or Execute** preference.

Debugging multi-process code can be a problem if child processes are started and not automatically debugged. This is commonly a problem when using Flask, Django, and other frameworks that implement auto-reload by managing and restarting a child process. Debugging child processes is only possible in Wing Pro, and is off by default. To enabled it, set **Debug Child Process** under the **Debug/Execute** tab in **Project Properties** to **Always Debug Child Processes**.

Importing a module before debug has started will appear to miss breakpoints at the top level of the module when it is imported again after debug has started, because the top level of the module is evaluated only during the first import. This occurs with some Python standard library modules that the debugger loads before starting user code, and may occur with any modules loaded before debug is started with **wingdbstub**.

Storing incorrect file names in ``*.pyc`` files prevents the debugger from identifying which breakpoints are relevant. This can be caused by using partial path names on the **Python Path** or when invoking a script from the command line, moving around the ***.pyc** file after they are created, or using **compileall.py** to create ***.pyc** files from source. The easiest way to solve this is to use only full paths on **Python Path** and remove any problematic ***.pyc** files so they can be regenerated.

Failing to send file names to compile() results in code objecs with **co_filename** set to **\<string\>**, which makes it impossible to determine which breakpoints are relevant. This is seen fairly often in embedded Python implementations, where Python acts as a scripting language in a larger application. A work-around is to set **__file__** in the module to the correct full path to the source code, although it's better to fix how **compile()** is being used.

Too many debug processes may cause Wing to fail to stop on breakpoints because it can only debug a finite number of processes at a time. The number of processes that Wing can debug concurrently is **1** in Wing 101 and Wing Personal and set with the **Debugger > Processes > Maximum Process Count** preference in Wing Pro.

Other less common problems include running Python with **psyco** or other optimizer, overriding the Python **__import__** routine, adding breakpoints after you've started debugging an application that spends much of its time in non-Python code, and using symbolic links on Windows.

For more information, see Debugger Limitations.

19.3.3. Failure to Stop on Exceptions

Failure to stop on exceptions is most commonly caused by the same factors that cause failure to stop on breakpoints, although in this case the debugger may stop but fail to show the source code.

Another factor in debugging exceptions is that they may be handled by a catch-all exception handler. Wing doesn't stop on these unless they also print the exception.

The simple work-around for this is to set a breakpoint in the 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 **When Printed** exception handling mode, as set by the **Debugger > Exceptions > Report Exceptions** preference).

The following variant can be used with other exception handling modes:

```
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.

19.3.4. Extra Debugger Exceptions

Wing always stops on **AssertionError**, even if the exception is handled because these are intended to indicate an error in code.

However, since not all programmers use exceptions in the same way, you may find Wing stopping in places that you don't want it to.

To avoid this, you can train Wing to ignore unwanted exception reports with the checkbox in the **Exceptions** tool. Or remove **AssertionError** from the **Debugger > Exceptions > Always Report** preference.

For more information, see Managing Exceptions.

19.4. Trouble-shooting Other Known Problems

Other known problems that can affect some of Wing's functionality include:

Copy/Paste Fails on Windows

Webroot Secure Anywhere v8.0.4.66 blocks Wing and Python's access to the clipboard by default so Copy/Paste will not work. The solution is to remove Wing and Python from the list of applications that Webroot is denying access to the clipboard.

Windows Won't Open File Names with Spaces

File Explorer on some versions of Windows fails to open Python files with Wing if the full path of the file has spaces in it. This is because Windows has 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 be in a form similar to the following, although the actual path will vary according to your installation location for Wing:

```
"C:\Program Files (x86)\Wing Pro 7.2\bin\wing.exe" "%1" %*
```

Failure to Detect HTTP Proxy and Connect to wingware.com

Wing tries to open an http connection to **wingware.com** when you activate a license, check for product updates, or submit a bug report or feedback from the **Help** menu. If you are running in an environment with an http proxy, Wing tries to auto-detect your proxy settings. If this fails you will need to configure your proxy manually using Wing's **Network > HTTP Proxy Server** preference. To

determine the correct settings to use, ask your network administrator or see how to determine proxy settings.

Poor Mouse Wheel Scrolling on Linux

If the mouse wheel does not work right on Linux, the utility **imwheel** may solve it, as described here

Failure to Find Python

Wing scans for Python at startup and in rare cases may 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) or the command that activates your virtualenv or Anaconda environment. 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 can cause this during periods of intensive scanning.

19.5. Obtaining Diagnostic Output

Wing and your debug code run in separate processes, each of which can independently be configured to collect additional diagnostic log information.

Diagnosing IDE Problems

Submit Bug Report in the **Help** menu is a quick way to diagnose problems seen while working with Wing. Please include a description of the problem, your email address so we can follow up, and leave the **Include error log** checkbox checked so we have the information needed to diagnose and fix the problem. The error log is the file **ide.log** in your Settings Directory .

To diagnose failure to start, or if you can't submit a bug report directly from Wing, run **console_wing.exe** (on Windows) or **wing7.2 --verbose** (on Linux and OS X) from the command line to obtain diagnostic output that you can email to support at wingware.com along with your system type and version, version of Wing, version of Python, and a description of the problem you are running into.

If Wing is crashing please provide the file **segfault.log** from the User Settings Directory with any bug reports.

Diagnosing Debugger Problems

To diagnose debugger problems, set the **Debugger > Diagnostics > Debug Internals Log File** preference to the full path of a file that the debugger will be able to create. Then try debugging again.

If the file does not appear, instead set the **Debugger > Diagnostics > Debug Internals Log File** preference to **Log to sys.stderr** and enable the **Debugger > I/O > Use External Console** and **Debugger > I/O > External Console Waits on Exit** preferences. When you try again, Wing should display a debug console with diagnostics.

If you are using **wingdbstub** to start debug, instead set **WINGDB_LOGFILE** environment variable to **<stderr>** (or alter **kLogFile** inside **wingdbstub.py**), and try launching the following script from the command line:

```
import wingdbstub
print("test1")
x = not_defined
print("test2")
```

Never check the **Extremely Verbose Internal Log** preference unless Wingware Technical Support asks you to. When this is enabled, it will drastically slow down the debugger.

Debugger diagnostic logs can be emailed to support at wingware.com together with the file **ide.log** in your User Settings Directory , your system version, version of Wing, version of Python, and a description of the problem you are seeing.

You will want to turn off diagnostic logging again after submitting your report since it slows down debugging considerably.

Diagnosing Debug Process Crashing

If your debug process is crashing entirely while Wing is interacting with it, it may be that Wing is exercising buggy code when it inspects data in the debug process. In this case, it can be useful to capture the Python stack at the moment of the crash. You can do this by installing faulthandler into the Python that runs your debug process, and then adding the following to code that is executed before the crash:

```
import faulthandler
faulthandler.enable()
```

After that is done, the Python stack for each thread is written to **stderr** if the process crashes.

If you can't access output sent to **stderr**, you can send the stack to a file instead, as follows:

```
import faulthandler
segfault_fn = '/path/to/segfault.log'  # Change this to a valid path
f = open(segfault_fn, 'a')
faulthandler.enable(f)
# IMPORTANT: Leave f open!!!
```

It is *very important* that you leave the file f open for the life of the process. Otherwise faulthandler may corrupt another file opened later under the same file descriptor, by writing the stack there instead. This is a design limitation imposed by the nature of post-segfault processing.

Please send details of debugger crashes, including the Python stacks obtained by this method, to support@wingware.com. We will try to change Wing's data inspection to avoid the crash that you are seeing, and we may be able to offer a work-around.

See also Problems Handling Values.

Preferences Reference

This chapter documents the entire set of available preferences for Wing Pro. Note that this includes preferences that are ignored and unused in Wing Personal and Wing 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 (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: **[ru, de, None, en, fr]**

Default Value: **None**

Color Palette

The color palette used by Wing. All color preferences default to using colors from the palette, but can be overridden individually. The palette only applies to the editor, unless the Use Editor Palette Throughout the UI preference is enabled. 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>, <type unicode>**

Default Value: **wing-classic**

Use Color Palette Throughout the UI

Controls whether the color palette is used throughout the user interface

Internal Name: **gui.use-palette-throughout-ui**

Data Specification: **<boolean: 0 or 1>**

Default Value: **False**

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**

Tooltip Delay

The time in seconds to wait after the mouse cursor stops moving before any tooltips are displayed.

Internal Name: **gui.tooltips-delay**

Data Specification: **<type int>**

Default Value: **0.5**

• Layout

Windowing Policy

Policy to use for window creation: Combined Toolbox and Editor mode places toolboxes into editor windows, and Separate Toolbox mode creates separate toolbox windows.

Internal Name: **gui.windowing-policy**

Data Specification: **[combined-window, 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**

Presentation Mode

Controls whether Wing runs in presentation mode, which magnifies the user interface. Wing must be restarted before this value takes effect.

Internal Name: **main.presentation-mode**

Data Specification: **<boolean: 0 or 1>**

Default Value: **False**

Presentation Mode Scale Factor

The amount by which to increase UI size when presentation mode is enabled. Wing must be restarted before this value takes effect.

Internal Name: **main.presentation-scale-factor**

Data Specification: **<type float>**

Default Value: **2.0**

• 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: **[text-only, icon-only, text-right, default, auto, text-below]**

Default Value: **auto**

Groups Shown

Controls which groups of tools will be shown in the toolbar.

Internal Name: **guimgr.toolbar-groups**

Data Specification:
[list of: [search, indent, clip, bookmark, debug, vcs, proj, file, diff, test, batch-search, select]]

Default Value: **['file', 'clip', 'select', '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 unicode>, <type str>, <type unicode>]]**

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>, <type unicode>]**

Default Value: **None**

Editor Font/Size

The base font and size to use for source code shown in the editor, Python Shell, Debug Console, Source Assistant, and other tools that display source code.

Internal Name: **edit.qt-display-font**

Data Specification: **[None or <type str>, <type unicode>]**

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, xcode, eclipse, brief, emacs, matlab, 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: **[3, 1, 2, None, 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**

Fallback to Mac OS key bindings

Use key bindings from OS X / macOS keymap for keys not defined in currently selected keymap

Internal Name: **guimgr.fallback-to-macos-keymap**

Data Specification: **<boolean: 0 or 1>**

Default Value: **True**

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>, <type unicode>, values: <type str>, <type unicode>]**

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**

When Launching Wing

Controls whether Wing tries to reuse an existing running instance of the IDE when it is launched again.

Internal Name: **main.instance-reuse-policy**

Data Specification: **[new, None, reuse]**

Default Value: **None**

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**

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 IDE log file (USER_SETTINGS_DIR/ide.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 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**

Auto-check for Product Updates

Automatically check for updates at startup by connecting to wingware.com. Updates are checked every three days, or more often for prerelease versions.

Internal Name: **main.auto-check-updates**

Data Specification: **<boolean: 0 or 1>**

Default Value: **1**

Submit Usage Stats

Allow submitting a log of which features you use to Wingware. This is done periodically at startup and also when you submit bug reports, feedback, or check for updates. The data provided is held confidential, used only for technical support and planning future development, and can be seen in the file USER_SETTINGS_DIR/stats.log

Internal Name: **main.submit-usage-stats**

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**

Show Discount Offers

Controls whether Wing will periodically show discount offers.

Internal Name: **main.show-offers**

Data Specification: **<boolean: 0 or 1>**

Default Value: **1**

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.

Internal Name: **gui.show-report-error-dialog**

Data Specification: **<boolean: 0 or 1>**

Default Value: **False**

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, recent-directory, current-focus, selected-directory], <type str>]**

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-fullpath, prepend-relative]**

Default Value: **append-relative**

Show Hostname in Titles

Show the remote host name in all basename-only filenames used in titles.

Internal Name: **gui.include-host-in-titles**

Data Specification: **<boolean: 0 or 1>**

Default Value: **True**

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 [cp1258, gb18030, cp500, iso_2022_jp_1, cp1026, iso_2022_jp, euc_jis_2004, cp857, cp1140, mac_latin2, gb2312, euc_jp, cp424, shift_jis_2004, mac_cyrillic, cp932, mac_roman, ascii, utf_16, iso8859_15, iso8859_14, iso8859_10, iso8859_13, johab, cp950, koi8_u, gbk, cp869, cp1254, utf_16_le, mac_iceland, cp860, iso8859_9, cp862, cp863, cp864, cp865, euc_jisx0213, hz, utf_8, big5hkscs, shift_jisx0213, mac_greek, cp775, iso_2022_jp_2, utf_7, cp861, mac_turkish, cp1255, latin_1, cp1257, cp1256, cp1251, cp1250, cp1253, cp1252, cp437, iso_2022_jp_2004, None, cp875, iso_2022_jp_ext, iso_2022_kr, cp855, cp1006, cp737, cp874, cp949, cp037, big5, koi8_r, cp850, shift_jis, cp852, utf_16_be, iso8859_8, iso_2022_jp_3, cp856, iso8859_5, iso8859_4, iso8859_7, iso8859_6, iso8859_3, iso8859_2]]**

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. 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, text/x-pov, text/x-ave, text/x-less, text/x-pl-sql, text/x-bash, text/x-java-source, text/x-lua-source, text/x-eiffel, text/x-vxml, text/x-lot, text/x-errorlist, text/x-caml, text/xml, application/json, text/x-octave, text/x-asn1, text/x-php-source, text/x-cython, application/x-tex, text/x-dos-batch, text/x-bullant, text/x-baan, text/x-python, text/x-mako, text/x-matlab, text/x-metaport, text/x-mmixal, text/x-ncrontab, text/postscript, text/x-django, text/x-cmake, text/x-erlang, text/x-javascript, text/x-scss, text/x-fortran, text/x-mysql, text/x-vhdl, text/x-escript, text/x-lisp, text/x-makefile, text/x-diff, text/x-haskell, text/x-ms-idl, text/x-cpp-source, text/x-asm, text/x-ruby, text/x-abaqus, text/x-ada, text/x-d, text/x-idl, text/x-nsis, text/x-scriptol, text/x-markdown, text/x-po, text/x-docbook, text/x-rc, text/x-coffee, text/x-verilog, text/x-xcode, text/x-c-source, text/plain, text/x-spice**

, text/x-zope-pt, text/x-lout, text/x-hss, text/x-inno-setup, text/html, text/x-forth, text/x-perl, text/x-tcl, text/x-qss, text/x-vb-source, text/x-pascal, text/x-yaml, text/x-conf, text/x-ms-makefile, text/x-properties, text/css, text/x-r]]

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: **{'All Source Files': (set([]), set([('wildcard-filename', '*.pyo'), ('wildcard-filename', '*$py.class'), ('wildcard-filename', '*.exe'), ('wildcard-filename', '*.dsw'), ('wildcard-filename', '*.bsc'), ('wildcard-filename', '/core'), ('wildcard-directory', '__pycache__'), ('wildcard-filename', '.#*'), ('wildcard-filename', '*.lib'), ('wildcard-filename', '*.bak'), ('wildcard-filename', '*.tgz'), ('wildcard-filename', '*.sln'), ('wildcard-directory', '.git'), ('wildcard-filename', '*.obj'), ('wildcard-directory', '.hg'), ('wildcard-filename', '*.ilk'), ('wildcard-filename', '*.temp'), ('wildcard-filename', '*~'), ('wildcard-filename', '*.manifest'), ('wildcard-filename', '*.pdb'), ('wildcard-filename', '*.old'), ('wildcard-filename', '*.wpr'), ('wildcard-filename', '*.log'), ('wildcard-filename', '*.ncb'), ('wildcard-filename', '*.user'), ('wildcard-directory', '_svn'), ('wildcard-filename', '*.orig'), ('wildcard-filename', '*.so'), ('wildcard-filename', '*.tmp'), ('wildcard-directory', '.xvpics'), ('wildcard-filename', '#*#'), ('wildcard-filename', '*.pyd'), ('wildcard-filename', '*.zip'), ('wildcard-filename', '*.wpu'), ('wildcard-filename', '*.a'), ('wildcard-directory', '.svn'), ('wildcard-filename', '/.coverage'), ('wildcard-filename', '*.dll'), ('wildcard-directory', '.pytest_cache'), ('wildcard-filename', '*.dsp'), ('wildcard-filename', '*.pyc'), ('wildcard-filename', '*.tar.gz'), ('wildcard-directory', '.bzr'), ('wildcard-directory', 'CVS'), ('wildcard-filename', '*.sbr'), ('wildcard-filename', '*.o'), ('wildcard-filename', '*-old'), ('wildcard-filename', '*.suo'), ('wildcard-filename', '*.svn-base'), ('wildcard-filename', '*.vcproj')])), 'HTML and XML Files': (set([('mime-type', 'text/html'), ('mime-type', 'text/xml'), ('mime-type', 'text/x-zope-pt')]), set([('wildcard-filename', '*.orig'), ('wildcard-directory', '_svn'), ('wildcard-directory', '.xvpics'), ('wildcard-filename', '*~'), ('wildcard-directory', '.bzr'), ('wildcard-directory', 'CVS'), ('wildcard-filename', '#*#'), ('wildcard-filename', '/.coverage'), ('wildcard-directory', '__pycache__'), ('wildcard-filename', '*.svn-base'), ('wildcard-directory', '.git'), ('wildcard-filename', '.#*'), ('wildcard-directory', '.svn'), ('wildcard-directory', '.pytest_cache'), ('wildcard-directory', '.hg')])), 'C/C++ Files': (set([('mime-type', 'text/x-c-source'), ('mime-type', 'text/x-cpp-source')]), set([('wildcard-filename', '*.orig'), ('wildcard-directory', '_svn'), ('wildca**

rd-directory', '.xvpics'), ('wildcard-filename', '*~'), ('wildcard-directory', '.bzr'), ('wildcard-direc
tory', 'CVS'), ('wildcard-filename', '#*#'), ('wildcard-filename', '/.coverage'), ('wildcard-directory
', '__pycache__'), ('wildcard-filename', '*.svn-base'), ('wildcard-directory', '.git'), ('wildcard-file
name', '.#*'), ('wildcard-directory', '.svn'), ('wildcard-directory', '.pytest_cache'), ('wildcard-dir
ectory', '.hg')])), 'Hidden & Temporary Files': (set([('wildcard-filename', '*.pyo'), ('wildcard-file
name', '*$py.class'), ('wildcard-filename', '*.exe'), ('wildcard-filename', '*.bsc'), ('wildcard-filen
ame', '/core'), ('wildcard-filename', '.#*'), ('wildcard-filename', '*.lib'), ('wildcard-filename', '*.ba
k'), ('wildcard-filename', '*.tgz'), ('wildcard-directory', '.git'), ('wildcard-filename', '*.obj'), ('wild
card-filename', '*.ilk'), ('wildcard-filename', '*.temp'), ('wildcard-filename', '*~'), ('wildcard-dire
ctory', '__pycache__'), ('wildcard-filename', '*.pdb'), ('wildcard-filename', '*.old'), ('wildcard-fil
ename', '*.wpr'), ('wildcard-filename', '*.ncb'), ('wildcard-directory', '_svn'), ('wildcard-filename
', '*.orig'), ('wildcard-filename', '*.so'), ('wildcard-filename', '*.tmp'), ('wildcard-directory', '.xvpi
cs'), ('wildcard-filename', '#*#'), ('wildcard-filename', '*.pyd'), ('wildcard-filename', '*.zip'), ('wil
dcard-filename', '*.wpu'), ('wildcard-filename', '*.a'), ('wildcard-filename', '/.coverage'), ('wildca
rd-filename', '*.dll'), ('wildcard-directory', '.pytest_cache'), ('wildcard-directory', '.hg'), ('wildca
rd-filename', '*.pyc'), ('wildcard-filename', '*.tar.gz'), ('wildcard-directory', '.bzr'), ('wildcard-dir
ectory', 'CVS'), ('wildcard-filename', '*.sbr'), ('wildcard-filename', '*.o'), ('wildcard-filename', '*-
old'), ('wildcard-filename', '*.svn-base'), ('wildcard-directory', '.svn')]), set([])), 'Python Files': (
set([('mime-type', 'text/x-cython'), ('mime-type', 'text/x-python')]), set([('wildcard-filename', '*.
orig'), ('wildcard-directory', '_svn'), ('wildcard-directory', '.xvpics'), ('wildcard-filename', '*~'), (
'wildcard-directory', '.bzr'), ('wildcard-directory', 'CVS'), ('wildcard-filename', '#*#'), ('wildcard-
filename', '/.coverage'), ('wildcard-directory', '__pycache__'), ('wildcard-filename', '*.svn-base'
), ('wildcard-directory', '.git'), ('wildcard-filename', '.#*'), ('wildcard-directory', '.svn'), ('wildcar
d-directory', '.pytest_cache'), ('wildcard-directory', '.hg')]))}

- • **Reloading**

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, Request 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**

Reloading Deleted Disk Files

Specifies the behavior of reload when a file that is open in an editor disappears on disk. The default Closes Editor is recommended if using revision control. Otherwise, retaining the current editor content reduces the chances of entirely losing a file if it is accidentally deleted on disk.

Internal Name: **guimgr.deleted-disk-file-policy**

Data Specification: **[list of: [close, prompt, blank]]**

Default Value: **close**

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**

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

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**

Space Indicator Size

Sets the size of the indicator to use for a space character when white space is being shown on the editor. This may be set to zero to show only tab characters.

Internal Name: **edit.space-indicator-size**

Data Specification: **<type int>**

Default Value: **1**

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**

Other Split Type

The type of split to create with commands that display in other split. The default is to split horizontally if the window width is greater than the height and to split vertically otherwise.

Internal Name: **edit.other-split-type**

Data Specification: **[default, horizontal, vertical]**

Default Value: **default**

Show All Files 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**

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, block-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: **False**

- **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 (to control transparency) can be set.

Internal Name: **edit.caret-line-highlight**

Data Specification: **[None or [tuple length 2 of: [None or [tuple length 3 of: [from 0 to 255], [from 0 to 255], [from 0 to 255]]], <type int>]]**

Default Value: **None**

Scrolling Context Lines

The number of lines of context to show above or below the caret when auto-scrolling the editor to a new position

Internal Name: **edit.scroll-context-lines**

Data Specification: **<type int>**

Default Value: **5**

Display Selections Popup

When to display multiple selections popup window

Internal Name: **edit.display-selection-popup**

Data Specification: **[always, never, multiple]**

Default Value: **multiple**

- **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**

• Callouts

Enable Callouts

Whether to enable display of callouts that are briefly displayed to indicate textvisited in the editor by search, goto-definition, and other navigation features.

Internal Name: **edit.callout-enable**

Data Specification: **<boolean: 0 or 1>**

Default Value: **True**

Callout Color

The color used for callouts on the editor.

Internal Name: **edit.callout-color**

Data Specification: **[None or [tuple length 3 of: [from 0 to 255], [from 0 to 255], [from 0 to 255]]]**

Default Value: **None**

Persistence Time (ms)

The time in milliseconds that callouts persist after display on the editor.

Internal Name: **.edit.callout-persistence**

Data Specification: **<type int>**

Default Value: **500**

• Indentation

Use Indent Analysis

Select when to use indent analysis (examination of current file contents) in order to determine indentation type and size. Either always in all files, only in Python files, or never. When disabled, the Preferred Indent Style from Project Properties and Default Indent Size and Default Tab Size from preferences will be used.

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 [tuple length 3 of: [from 0 to 255], [from 0 to 255], [from 0 to 255]]]]**

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**

Convert Indent Style On Paste

Controls whether Wing automatically converts indent style and size on text that is pasted into an editor.

Internal Name: **edit.convert-indents-on-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 Console, 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: <type str>, values: [dict; keys: [italic, back, fore, bold], values: [one of: None, <type str>, <boolean: 0 or 1>]]]]**

Default Value: **{}**

Highlight Builtins

Highlight Python builtins

Internal Name: **edit.highlight-builtins**

Data Specification: **<boolean: 0 or 1>**

Default Value: **True**

- ### Brace Matching

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**

Brace Highlight Color

The color used to highlight matching braces.

Internal Name: **edit.brace-highlight-color**

Data Specification: **[None or [tuple length 3 of: [from 0 to 255], [from 0 to 255], [from 0 to 255]]]**

Default Value: **None**

Brace Highlight Background Color

The background color used to highlight matching braces.

Internal Name: **edit.brace-highlight-backcolor**

Data Specification: **[None or [tuple length 3 of: [from 0 to 255], [from 0 to 255], [from 0 to 255]]]**

Default Value: **None**

Brace Badlight Color

The color used to highlight bad braces.

Internal Name: **edit.brace-badlight-color**

Data Specification: **[None or [tuple length 3 of: [from 0 to 255], [from 0 to 255], [from 0 to 255]]]**

Default Value: **None**

Brace Badlight Background Color

The background color used to highlight bad braces.

Internal Name: **edit.brace-badlight-backcolor**

Data Specification: **[None or [tuple length 3 of: [from 0 to 255], [from 0 to 255], [from 0 to 255]]]**

Default Value: **None**

• **Code Warnings**

Enable Code Warnings

Whether to enable the code warnings system as a whole. When this is disabled, no code warnings are displayed and external code warnings systems will not be launched even if enabled.

Internal Name: **codewarnings.enable**

Data Specification: **<boolean: 0 or 1>**

Default Value: **True**

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**

Indicator Style

Visual display style to use for code errors and warnings shown on the editor.

Internal Name: **edit.indicator-style**

Data Specification: **[squiggle, filled-box, thick-underline, underline, plain-box]**

Default Value: **squiggle**

Error Color

Color to use to indicate code errors in the editor.

Internal Name: **edit.code-error-color**

Data Specification: **[None or [tuple length 3 of: [from 0 to 255], [from 0 to 255], [from 0 to 255]]]**

Default Value: **None**

Warning Color

Color to use to indicate code warnings in the editor.

Internal Name: **edit.code-warning-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**

Confirm Deletion

Show a confirmation dialog when deleting bookmarks

Internal Name: **edit.bookmark-confirm-delete**

Data Specification: **<boolean: 0 or 1>**

Default Value: **True**

- **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**

Python Turbo Mode

When enabled, the Python auto-completer enters the completion automatically whenever a key other than a valid symbol name key is pressed. Press a modifier key (Shift, Alt, or Ctrl) by itself to exit the completer without entering a completion. 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**

Completion Keys

Controls which keys will enter selected completion value into the editor.

Internal Name: **edit.autocomplete-keys**

Data   Specification:
[list of: [f1, f3, return, space, period, bracketleft, tab, f12, colon, f10, parenleft]]

Default Value: **['tab']**

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 Mode

Selects how completion is done in the editor: Either insert the completion at the cursor, replace any symbols that heuristically match the selected 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**

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**

Non-Latin Script Display

Whether to display autocompleter for non-latin scripts

Internal Name: **.edit.display-autocompleter-for-non-latin-scripts**

Data Specification: **[auto, yes, no]**

Default Value: **auto**

- **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**

** Auto-wrap Arguments**

Enable auto-wrapping of arguments during auto-invocation.

Internal Name: **edit.auto-edit-invoke-wraps**

Data Specification: **<boolean: 0 or 1>**

Default Value: **0**

** Invoke After Completion**

Enable auto-invocation to occur automatically after a callable symbol is entered by the auto-completer.

Internal Name: **edit.auto-edit-invoke-after-complete**

Data Specification: **<boolean: 0 or 1>**

Default Value: **0**

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 PEP 8 Style Spacing**

When auto-entering spaces is enabled, enforce PEP 8 style spacing by preventing redundant spaces.

Internal Name: **edit.auto-edit-spaces-enforce**

Data Specification: **<boolean: 0 or 1>**

Default Value: **0**

** Spaces Around = in Argument Lists**

Add spaces around = in argument lists.

Internal Name: **edit.auto-edit-spaces-args**

Data Specification: **<boolean: 0 or 1>**

Default Value: **0**

** Spaces Elsewhere in Argument Lists**

Add spaces around characters other than = in argument lists.

Internal Name: **edit.auto-edit-spaces-args-override**

Data Specification: **<boolean: 0 or 1>**

Default Value: **0**

** Spaces After : in Type Annotations**

When auto-entering spaces is enabled, also auto-enter spaces around ":" in type annotations.

Internal Name: **edit.auto-edit-spaces-types**

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: **True**

** Prefer Block Management Over :=**

In Python 3.8+, prioritize block management with : over the possibility of entering := in the current editing context.

Internal Name: **edit.auto-edit-colon-prioritize-blocks**

Data Specification: **<boolean: 0 or 1>**

Default Value: **False**

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**

- **Auto-formatting**

Auto-Reformat

Controls when Wing automatically reformats code as you edit it. May be disabled, limited to only edited lines after the caret leaves that line, or performed on whole files when they are saved to disk.

Internal Name: **edit.pep8-autoformat**

Data Specification: **[disabled, files, lines]**

Default Value: **disabled**

Reformatter

Selects the reformatter to use when reformatting code automatically.

Internal Name: **edit.pep8-reformatter**

Data Specification: **[yapf, pep8, black]**

Default Value: **pep8**

Reformat Timeout

Number of seconds to wait for auto-formatting to complete before aborting the reformatting process.

Internal Name: **edit.pep8-timeout**

Data Specification: **<type float>, <type int>**

Default Value: **5**

Enforce Line Length

Whether to enforce line length during auto-formatting. The length is specified with the Editor > Line Wrapping > Reformatting Column preference.

Internal Name: **edit.pep8-enforce-line-length**

Data Specification: **<boolean: 0 or 1>**

Default Value: **0**

PEP 8: Reindent All Lines in Files

Whether to reindent all lines during PEP 8 reformatting. This affects only reformatting of whole files. Lines in a selection are never reindented during reformatting.

Internal Name: **edit.pep8-reindent-all-lines**

Data Specification: **<boolean: 0 or 1>**

Default Value: **0**

PEP 8: Spaces Around = in Argument Lists

Override PEP 8 by adding spaces around = in argument lists.

Internal Name: **edit.pep8-spaces-args**

Data Specification: **<boolean: 0 or 1>**

Default Value: **0**

PEP 8: Spaces After #

When applying PEP 8 rules, follow PEP 8 by enforcing the addition of spaces after # comment start.

Internal Name: **edit.pep8-spaces-comment**

Data Specification: **<boolean: 0 or 1>**

Default Value: **1**

PEP 8: Move Indents to Top

When applying PEP 8 rules, follow PEP 8 by moving all indents to the top of the file.

Internal Name: **edit.pep8-move-indents-to-top**

Data Specification: **<boolean: 0 or 1>**

Default Value: **0**

Black: Skip String Normalization

Whether or not to prevent Black from normalizing string quotes during auto-formatting.

Internal Name: **edit.black-skip-string-normalization**

Data Specification: **<boolean: 0 or 1>**

Default Value: **1**

- **Snippets**

Include Snippets in Auto-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**

Snippets Path

Path to search for code snippets. Later directories on the path override earlier directories for a particular snippet name. Partial paths are interpreted relative to the current user's home directory (/Users/maint/). new snippets will be created in the last directory on the path.

Internal Name: **edit.snippets-path**

Data Specification: **[tuple of: <type str>]**

Default Value: **()**

Include Default Snippets

Whether to include the default snippets set in the Snippets tool. These are found in the User Settings directory (USER_SETTINGS_DIR)

Internal Name: **edit.snippets-include-defaults**

Data Specification: **<boolean: 0 or 1>**

Default Value: **True**

- **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**

Document Font

Font to use when printing.

Internal Name: **edit.print-font**

Data Specification: **[None or <type str>, <type unicode>]**

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**

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 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 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 Footer Font

Font to use in print footer.

Internal Name: **edit.print-header-font**

Data Specification: **[None or <type str>]**

Default Value: **None**

- **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: **[list 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**

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: **5**

Use Custom Mouse Cursor

When to use a custom mouse cursor. The color of the cursor will be the color of the caret.

Internal Name: **edit.use-custom-mouse-cursor**

Data Specification: **[always, on-dark-backgrounds, never]**

Default Value: **on-dark-backgrounds**

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', 'indent-to-match'), (u'Uncomment out Region', 'uncomment-out-region'), (u'Outdent Region', 'outdent-region'), (u'Comment out Region', 'comment-out-region')], values: [(u'Never Select', 'never-select'), (u'Retain Select', 'retain-select'), (u'Always Select', 'always-select')]]**

Default Value: **{'uncomment-out-region': 'retain-select', 'outdent-region': 'retain-select', 'comment-out-region': 'retain-select', 'indent-region': 'retain-select', 'indent-to-match': 'retain-select'}**

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 Console.

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: **95**

Show Debug Environment Dialog

Controls whether the Debug Environment 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 launch file's or named entry point's properties.

Internal Name: **debug.show-args-dialog**

Data Specification: **[per-file, never]**

Default Value: **per-file**

Indicate Project Files in Stack

Enable to indicate projects files in the the debug stack, in the stack selector, Stack Data, and Exception tools.

Internal Name: **debug.indicate-project-files**

Data Specification: **\<boolean: 0 or 1\>**

Default Value: **True**

- **Processes**

Enable Multi-Process Debugging

Enable multi-process debugging. When disabled, Wing will only accept one debug connection at a time.

Internal Name: **debug.multi-process-debug**

Data Specification: **\<boolean: 0 or 1\>**

Default Value: **True**

Switch to Stopped Processes

When to automatically switch the currently active debug process to a process that reaches a breakpoint or exception. The default Switch to Most Recently Launched Process Group switches only if there is no other debug process active or if the process belongs to the most recently debug session started from the IDE (this does not include processes that attach using wingdbstub).

Internal Name: **debug.multi-process-switch**

Data Specification: **[always, none, launched]**

Default Value: **launched**

Debug Child Processes

Enable debugging sub-processes. When disabled, Wing will only debug the initially launched parent process.

Internal Name: **debug.multi-process-debug-sub-processes**

Data Specification: **\<boolean: 0 or 1\>**

Default Value: **False**

Replace sys.executable

Enable replacement of sys.executable so that processes launched using that value will be debugged. This must be enabled on Windows in order to debug child processes created with the multiprocessing module.

Internal Name: **debug.multi-process-replace-sys-executable**

Data Specification: **\<boolean: 0 or 1\>**

Default Value: **True**

Maximum Process Count

Maximum number of debug processes that can connect to Wing at once. After the limit is reached, Wing accepts no additional connections until some processes detach or exit.

Internal Name: **debug.multi-process-maximum**

Data Specification: **<type int>**

Default Value: **50**

Debug Multiple Tests at Once

Enable debugging more than one unit test at once. When enabled, the Debug/Abort button in the Testing tool alters according to which test is selected.

Internal Name: **debug.multi-process-multiple-tests**

Data Specification: **<boolean: 0 or 1>**

Default Value: **False**

Debug Multiple Instances of a Named Entry Point

Enable debugging more than one instance of a named entry point. When disabled, any existing debug process for a named entry point will be terminated when it is debugged.

Internal Name: **debug.multi-process-multiple-entry-points**

Data Specification: **<boolean: 0 or 1>**

Default Value: **False**

• **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, Japanese iso-2022-jp-2 004, Hebrew cp856, Japanese euc-jp, Vietnamese cp1258, Greek cp1253, Baltic Languages cp1257, Korean johab, Western European cp1252, Baltic Languages cp775, Japanese iso-2022-jp-ext, Korean iso-2022-kr, Icelandic cp861, Hebrew cp424, Cyrillic Languages cp1251, Turkish iso8859-9, Unicode (UTF-16, little endian) utf-16-le, Western European cp500, Chinese (PRC) gb18030, Greek cp875, Arabic cp864, Icelandic mac-iceland, Chinese (PRC) gbk, Turkish mac-turkish, Greek iso8859-7, Baltic Languages iso8859-13, Cyrillic Languages mac-cyrillic, Greek cp869, Japanese iso-2022-jp-1, Central and Eastern European cp852, Japanese iso-2022-jp-2, Chinese (ROC) big5, Urdu cp1006, Console default (utf-8), Hebrew iso8859-8, Japanese iso-2022-jp-3, Cyrillic Languages iso8859-5, Thai cp874, Cyrillic Languages cp855, Western European iso8859-15, Greek mac-greek, Ukrainian koi8-u, Hebrew cp1255, Danish, Norwegian cp865, Celtic Languages iso8859-14, Turkish cp1026, Western European mac-roman, Western European cp1140, Chinese (PRC) hz, Portuguese cp860, Chinese (ROC) cp950, Unicode (UTF-16, big endian) utf-16-be, Japanese shift-jis-2004, Turkish cp1254, Hebrew cp862, Western European latin-1, Japanese euc-jisx0213, None, Japanese euc-jis-2004, Japanese shift-jisx0213, Central and Eastern European cp1250, Baltic Languages iso8859-4, Chinese (PRC) big5hkscs, English ascii, Japanese shift-jis, Arabic iso8859-6, Canadian English/French cp863, Russian koi8-r, Japanese iso-2022-jp, Unicode (UTF-8) utf-8, Greek cp737, Nordic Languages iso8859-10, Central and Eastern European mac-latin2, Chinese (PRC) gb2312, Unicode**

(UTF-7) utf-7, Arabic cp1256, US, Canada, and Others cp037, Western European cp850, Esperanto and Maltese iso8859-3, Turkish cp857, Korean cp949, US, Australia, New Zealand, S. Africa cp437, Unicode (UTF-16) utf-16, Japanese cp932]]

Default Value: **utf_8**

Flush I/O Periodically

Controls when the debugger periodically flushes I/O sent to sys.stdout and sys.stderr. Doing so may deadlock in some code. Not doing so may not display text that has been output without newline.

Internal Name: **debug.flush-io**

Data Specification: **[Always, Only if Single-Threaded, Never]**

Default Value: **single-thread**

Shell Encoding

Encoding of input/output in the integrated Python Shell and Debug Console

Internal Name: **debug.debug-probe-encoding**

Data Specification: **[None or [Central and Eastern European iso8859-2, Japanese iso-2022-jp-2004, Hebrew cp856, Japanese euc-jp, Vietnamese cp1258, Greek cp1253, Baltic Languages cp1257, Korean johab, Western European cp1252, Baltic Languages cp775, Japanese iso-2022-jp-ext, Korean iso-2022-kr, Icelandic cp861, Hebrew cp424, Cyrillic Languages cp1251, Turkish iso8859-9, Unicode (UTF-16, little endian) utf-16-le, Western European cp500, Chinese (PRC) gb18030, Greek cp875, Arabic cp864, Icelandic mac-iceland, Chinese (PRC) gbk, Turkish mac-turkish, Greek iso8859-7, Baltic Languages iso8859-13, Cyrillic Languages mac-cyrillic, Greek cp869, Japanese iso-2022-jp-1, Arabic iso8859-6, Central and Eastern European cp852, Japanese iso-2022-jp-2, Chinese (ROC) big5, Urdu cp1006, Hebrew iso8859-8, Japanese iso-2022-jp-3, Cyrillic Languages iso8859-5, Thai cp874, Cyrillic Languages cp855, Western European iso8859-15, Greek mac-greek, Ukrainian koi8-u, Hebrew cp1255, Danish, Norwegian cp865, Celtic Languages iso8859-14, Turkish cp1026, Western European mac-roman, Western European cp1140, Chinese (PRC) hz, Portuguese cp860, Chinese (ROC) cp950, Unicode (UTF-16, big endian) utf-16-be, Japanese shift-jis-2004, Turkish cp1254, Hebrew cp862, Western European latin-1, Japanese euc-jisx0213, None, Japanese euc-jis-2004, Japanese shift-jisx0213, Central and Eastern European cp1250, Baltic Languages iso8859-4, Chinese (PRC) big5hkscs, English ascii, Japanese shift-jis, Central and Eastern European mac-latin2, Canadian English/French cp863, Russian koi8-r, Japanese iso-2022-jp, Unicode (UTF-8) utf-8, Greek cp737, Nordic Languages iso8859-10, Use default stdin / stdout encoding, Chinese (PRC) gb2312, Unicode (UTF-7) utf-7, Arabic cp1256, US, Canada, and Others cp037, Western European cp850, Esperanto and Maltese iso8859-3, Turkish cp857, Korean cp949, US, Australia, New Zealand, S. Africa cp437, Unicode (UTF-16) utf-16, Japanese cp932]]**

Default Value: **utf_8**

Pretty Print in Shells

Enable to use pprint.pprint to display values in the Python Shell and Debug Console.

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, Japanese iso-2022-jp-2 004, Hebrew cp856, Japanese euc-jp, Vietnamese cp1258, Greek cp1253, Baltic Languages c p1257, Korean johab, Western European cp1252, Baltic Languages cp775, Japanese iso-202 2-jp-ext, Korean iso-2022-kr, Icelandic cp861, Hebrew cp424, Cyrillic Languages cp1251, Tur kish iso8859-9, Unicode (UTF-16, little endian) utf-16-le, Western European cp500, Chinese (PRC) gb18030, Greek cp875, Arabic cp864, Icelandic mac-iceland, Chinese (PRC) gbk, Turkis h mac-turkish, Greek iso8859-7, Baltic Languages iso8859-13, Cyrillic Languages mac-cyrilli c, Greek cp869, Japanese iso-2022-jp-1, Central and Eastern European cp852, Japanese iso- 2022-jp-2, Chinese (ROC) big5, Urdu cp1006, Console default (utf-8), Hebrew iso8859-8, Japa nese iso-2022-jp-3, Cyrillic Languages iso8859-5, Thai cp874, Cyrillic Languages cp855, Wes tern European iso8859-15, Greek mac-greek, Ukrainian koi8-u, Hebrew cp1255, Danish, Norw egian cp865, Celtic Languages iso8859-14, Turkish cp1026, Western European mac-roman, Western European cp1140, Chinese (PRC) hz, Portuguese cp860, Chinese (ROC) cp950, Unic ode (UTF-16, big endian) utf-16-be, Japanese shift-jis-2004, Turkish cp1254, Hebrew cp862, Western European latin-1, Japanese euc-jisx0213, None, Japanese euc-jis-2004, Japanese sh ift-jisx0213, Central and Eastern European cp1250, Baltic Languages iso8859-4, Chinese (PR C) big5hkscs, English ascii, Japanese shift-jis, Arabic iso8859-6, Canadian English/French c p863, Russian koi8-r, Japanese iso-2022-jp, Unicode (UTF-8) utf-8, Greek cp737, Nordic Lang uages iso8859-10, Central and Eastern European mac-latin2, Chinese (PRC) gb2312, Unicode (UTF-7) utf-7, Arabic cp1256, US, Canada, and Others cp037, Western European cp850, Espe ranto and Maltese iso8859-3, Turkish cp857, Korean cp949, US, Australia, New Zealand, S. Af rica cp437, Unicode (UTF-16) utf-16, Japanese cp932]]**

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. External consoles only work for locally run code. Remote debugging always uses the Debug I/O tool. To debug code running remotely in an external console, use wingdbstub to initiate debug.

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', 'gnome-terminal "--title=Wing Debug Console" -x', 'xterm -T "Wing Debug Console" -e', 'konsole -T "Wing Debug Console" -e', 'rxvt -T "Wing Debug Console" -e']**

- **Data Display**

Show __name Protected Variables

Controls whether the debugger shows protected variables (with one leadingunderscore) in the Stack Data view.

Internal Name: **debug.show-protected-variables**

Data Specification: **<boolean: 0 or 1>**

Default Value: **1**

Show ____name Private Variables

Controls whether the debugger shows private variables (with two leadingunderscores) in the Stack Data view.

Internal Name: **debug.show-private-variables**

Data Specification: **<boolean: 0 or 1>**

Default Value: **1**

Show ____name____ Special Variables

Controls whether the debugger shows special variables (with two leadingand two trailing underscores) in the Stack Data view.

Internal Name: **debug.show-special-variables**

Data Specification: **<boolean: 0 or 1>**

Default Value: **1**

Show Memory Addresses

Controls whether the debugger shows memory addresses as part of the display of object instances.

Internal Name: **debug.show-memory-addresses**

Data Specification: **<boolean: 0 or 1>**

Default Value: **1**

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 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**

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**

- **Data Filters**

Omit Types

Lists types for which values are never shown by the debugger. The strings here are compared with type(value).__name__ and the value is omitted if a match is found.

Internal Name: **debug.omit-types**

Data Specification: **[tuple of: <type str>]**

Default Value: **('function', 'builtin_function_or_method', 'class', 'classobj', 'instance method', 'type', 'module', 'ufunc', 'wrapper_descriptor', 'method_descriptor', 'methoddescriptor', 'member_descriptor', 'classmethod', 'staticmethod')**

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

Lists 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. The strings here are compared with type(value).__name__ and a value is not probed if a match is found.

Internal Name: **debug.no-probe-types**

Data Specification: **[tuple of: <type str>]**

Default Value: **('GdkColormap', 'IOBTree', 'JPackage', 'cython_function_or_method')**

• **Introspection**

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 Calls in Data Inspection

Enable to allow Python code and other dynamic calls to be invoked while inspecting data in the debugger, for display in any part of the IDE's user interface. This should be used with caution

because it can cause the debug process to execute unwanted code, make unexpected changes to runtime state, hang on lengthy computations, deadlock threads, or crash in buggy code.

Internal Name: **debug.allow-dynamic-introspection**

Data Specification: **<boolean: 0 or 1>**

Default Value: **False**

Call Python ____repr____ Methods

Allow __repr__ methods implemented in Python to be invoked. Disable this if the __repr__ methods take too long to complete or fail due to other bugs.

Internal Name: **debug.allow-bytecode-repr**

Data Specification: **<boolean: 0 or 1>**

Default Value: **True**

Inspect Base Classes

Whether to inspect base classes for class attributes. Disable this to work around crashing in packages such as openerp and odoo.

Internal Name: **debug.max-base-classes**

Data Specification: **<boolean: 0 or 1>**

Default Value: **True**

 • **Listening**

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**

Kill Externally Launched Processes

Enable or disable 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**

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**

- **Network**

Use Digests to Identify Files

Controls whether to build an inferred location map from file digest matches that are found locally when debugging files on a remote host. This allows the debugger to find files that are not in the project and were not found to be imported by static analysis, or that are still waiting to be scanned.

Internal Name: **debug.use-digests-to-identify-files**

Data Specification: **<boolean: 0 or 1>**

Default Value: **True**

Warn About Ambigious Digest Matches

Controls whether to show a dialog when the debugger detects a remote file that matches more than one local file.

Internal Name: **debug.show-multiple-local-files**

Data Specification: **<boolean: 0 or 1>**

Default Value: **True**

Location Map

Defines a mapping between the remote and local locations of files for host-to-host debugging. This is used only for manual remote debug configuration and is ignored when debug is controlled by a remote host configuration. For each specific IP address or IP address with wildcards (e.g. 10.1.1.*),

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 of: <type str>, <type str>]]]]**

Default Value: **{'127.0.0.1': None}**

Connection Keep Alive

Number of seconds between keep-alive messages sent to the debug process so that the connection doesn't close due to inactivity. Use a value <= 0 to disable the sending of keep-alive messages

Internal Name: **debug.send-keep-alive-seconds**

Data Specification: **<type int>**

Default Value: **0**

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**

Close Connection on Timeout

Controls whether the debugger will close the connection after any data handling timeout. This reduces the potential for hanging on data handling issues, but increases the chances the debug connection will be unnecessarily closed if any inspection of data takes more than the configured timeout to complete.

Internal Name: **debug.close-on-timeout**

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. Host names, specific IP numbers, or IP number dotted quad masks with * to match anything (e.g. 10.1.1.*) may be used. This is used only for manual remote debug configuration and is ignored when debug is controlled by a remote host configuration.

Internal Name: **debug.passive-hosts**

Data Specification: **[tuple of: <type str>]**

Default Value: **('*.*.*.*',)**

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 manual remote debug configuration, and are not necessary when debug is controlled by a remote host configuration. 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),)**

- **Shells**

Enable Debugging

Enables debugging code executed in the Python Shell or Debug Console.

Internal Name: **debug.debug-shells**

Data Specification: **<boolean: 0 or 1>**

Default Value: **0**

Enable Recursive Prompt

Enables recursive debugging in the Python Shell and Debug Console.

Internal Name: **debug.recursive**

Data Specification: **<boolean: 0 or 1>**

Default Value: **0**

Pretty Print

Enable to use pprint.pprint to display values in the Python Shell and Debug Console.

Internal Name: **debug.pretty-print-in-shells**

Data Specification: **<boolean: 0 or 1>**

Default Value: **False**

Auto-Restart when Switch Projects

Auto-restart the Python Shell when changing projects. When this is disabled, the Python Shell will continue to use environment from the previously opened project.

Internal Name: **debug.shell-auto-restart-proj-switch**

Data Specification: **<boolean: 0 or 1>**

Default Value: **1**

Auto-Restart when Evaluate File

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**

Prompt to Confirm Restart

Whether to prompt when restarting the Python Shell as a result of restarting debugging.

Internal Name: **debug.prompt-to-restart-python-shell-debug**

Data Specification: **<boolean: 0 or 1>**

Default Value: **True**

Filter History by Entered Prefix

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**

Evaluate Only 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**

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**

Show Editor on Exceptions in Shells

Controls whether the debugger raises source files to indicate exception locations encountered when working in the Debug Console, 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**

  • **Advanced**

Termination Model

How to terminate debug when a parent process or child process is terminated. A process group includes any all parent and child processes, up to the initially launched process, including also grand-children and any other descendent process.

Internal Name: **debug.multi-process-kill-model**

Data Specification: **[leave-running, auto-kill-group, prompt]**

Default Value: **auto-kill-group**

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**

Step Past importlib Frames

Controls where Wing ignores code inside of Python's importlib machinery when stepping through code. When enabled, Step Into on an import statement continues until it reaches the top level of the module being imported (or results in ImportError or moves past the import if the module was already imported), and Step Out will skip over frames in importlib.

Internal Name: **debug.ignore-import-lib**

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 Console will happen in unmodified stdin instead (even though output will still appear in the Debug Console as always).

Internal Name: **debug.use-stdin-wrapper**

Data Specification: **<boolean: 0 or 1>**

Default Value: **1**

When Build Fails

Controls whether to start debugging if the defined build process fails

Internal Name: **debug.debug-if-build-fails**

Data Specification: **[0, 1, None]**

Default Value: **None**

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**

Move Breakpoints to Valid Lines

Whether to automatically move breakpoints to a valid position when they are placed on a line that will not be reached by the Python interpreter, such as within certain types of multi-line expressions.

Internal Name: **debug.move-breakpoints**

Data Specification: **<boolean: 0 or 1>**

Default Value: **1**

Show Breaks Moved Dialog

Whether to show a dialog when a breakpoint is set on a different line than the selected on.

Internal Name: **debug.show-breaks-moved-message**

Data Specification: **<boolean: 0 or 1>**

Default Value: **1**

Animate Debug Data Tooltips

Whether to animate debug data tips shown when Shift-Space is pressed.

Internal Name: **debug.animate-data-tips**

Data Specification: **<boolean: 0 or 1>**

Default Value: **0**

- **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. The resulting log file can be emailed to support@wingware.com along with your bug report for interpretation. Logging can be disabled, or sent to stderr, stdout, or a file. When enabled, the debugger will run more slowly.

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. This should only be enabled at the request of Wingware Technical Support and will drastically slow down 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. The resulting log file can be emailed to support@wingware.com along with your bug report for interpretation. Logging can be disabled, or sent to stderr, stdout, or a file. When enabled, the Python Shell will run more slowly.

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. This should only be enabled at the request of Wingware Technical Support and will drastically slow down the Python Shell.

Internal Name: **debug.very-verbose-shell-log**

Data Specification: **<boolean: 0 or 1>**

Default Value: **False**

Source Analysis

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 100000]**

Default Value: **2000**

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**

Analyze Function and Method Calls

Whether to analyze function calls and record the types of values passed as arguments to functions. The disk cache should be cleared after this value is changed.

Preferences Reference

Internal Name: **pysource.analyze-function-calls**

Data Specification: **<boolean: 0 or 1>**

Default Value: **False**

- **• 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   from PyQt5 import QtSvg\n except:\n   from PySide import QtSvg\n', 'QtWidgets': 'try:\n from PyQt4 import QtWidgets\nexcept:\n try:\n   from PyQt5 import QtWidgets\n except:\n   from PySide import QtWidgets\n', 'wxpython': 'pass', 'QtHelp': 'try:\n from PyQt4 import QtHelp\nexcept:\n try:\n   from PyQt5 import QtHelp\n except:\n   from PySide import QtHelp\n', 'gdk': 'import pygtk\nvers = pygtk._get_available_versions().keys()\nvers.sort()\nvers.reverse()\nfor v in vers:\n try:\n   pygtk.require(v)\n   break\n except:\n   pass\n', 'QtGui': 'try:\n from PyQt4 import QtGui\nexcept:\n try:\n   from PyQt5 import QtGui\n except:\n   from PySide import QtGui\n', '_gst': 'from gst import _gst', 'gtk': 'import pygtk\nvers = pygtk._get_available_versions().keys()\nvers.sort()\nvers.reverse()\nfor v in vers:\n try:\n   pygtk.require(v)\n   break\n except:\n   pass\n', 'QtXml': 'try:\n from PyQt4 import QtXml\nexcept:\n try:\n   from PyQt5 import QtXml\n except:\n   from PySide import QtXml\n', 'QtWebKit': 'try:\n from PyQt4 import Qt**

WebKit\nexcept:\n try:\n   from PyQt5 import QtWebKit\n except:\n   from PySide import Qt
WebKit\n', 'QtScriptTools': 'try:\n from PyQt4 import QtScriptTools\nexcept:\n try:\n   from
PyQt5 import QtScriptTools\n except:\n   from PySide import QtScriptTools\n', 'QtSql': 'try:\
n from PyQt4 import QtSql\nexcept:\n try:\n   from PyQt5 import QtSql\n except:\n   from
PySide import QtSql\n', 'Qt': 'try:\n from PyQt4 import Qt\nexcept:\n try:\n   from PyQt5 imp
ort Qt\n except:\n   from PySide import Qt\n', 'QtAssistant': 'try:\n from PyQt4 import QtAss
istant\nexcept:\n try:\n   from PyQt5 import QtAssistant\n except:\n   from PySide import Q
tAssistant\n', 'QtXmlPatterns': 'try:\n from PyQt4 import QtXmlPatterns\nexcept:\n try:\n   fr
om PyQt5 import QtXmlPatterns\n except:\n   from PySide import QtXmlPatterns\n', 'QtDecl
arative': 'try:\n from PyQt4 import QtDeclarative\nexcept:\n try:\n   from PyQt5 import QtDe
clarative\n except:\n   from PySide import QtDeclarative\n', 'QtDesigner': 'try:\n from PyQt4
import QtDesigner\nexcept:\n try:\n   from PyQt5 import QtDesigner\n except:\n   from PyS
ide import QtDesigner\n', 'pango': 'import pygtk\nvers = pygtk._get_available_versions().key
s()\nvers.sort()\nvers.reverse()\nfor v in vers:\n try:\n   pygtk.require(v)\n   break\n except:\
n   pass\n', 'QtOpenGL': 'try:\n from PyQt4 import QtOpenGL\nexcept:\n try:\n   from PyQt5
 import QtOpenGL\n except:\n   from PySide import QtOpenGL\n', 'QtUiTools': 'try:\n from
PyQt4 import QtUiTools\nexcept:\n try:\n   from PyQt5 import QtUiTools\n except:\n   from
PySide import QtUiTools\n', 'QSci': 'try:\n from PyQt4 import QSci\nexcept:\n try:\n   from P
yQt5 import QSci\n except:\n   from PySide import QSci\n', 'atk': 'import pygtk\nvers = pygt
k._get_available_versions().keys()\nvers.sort()\nvers.reverse()\nfor v in vers:\n try:\n   pygtk
.require(v)\n   break\n except:\n   pass\n', 'QtTest': 'try:\n from PyQt4 import QtTest\nexcep
t:\n try:\n   from PyQt5 import QtTest\n except:\n   from PySide import QtTest\n', 'QtScript':
 'try:\n from PyQt4 import QtScript\nexcept:\n try:\n   from PyQt5 import QtScript\n except:
\n   from PySide import QtScript\n', 'gobject': 'import pygtk\nvers = pygtk._get_available_ver
sions().keys()\nvers.sort()\nvers.reverse()\nfor v in vers:\n try:\n   pygtk.require(v)\n   break\
n except:\n   pass\n', 'QtCore': 'try:\n from PyQt4 import QtCore\nexcept:\n try:\n   from Py
Qt5 import QtCore\n except:\n   from PySide import QtCore\n', 'QtNetwork': 'try:\n from Py
Qt4 import QtNetwork\nexcept:\n try:\n   from PyQt5 import QtNetwork\n except:\n   from P
ySide import QtNetwork\n'}

Python Docs URL Prefix

Prefix for Python Standard Library Documentation. This should be in the form
https://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**

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 ide.log in the user settings directory.

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: **[not-active, active-if-project-dir, always-active]**

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: **[not-active, active-if-project-dir, always-active]**

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: **[not-active, active-if-project-dir, always-active]**

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: **[not-active, active-if-project-dir, always-active]**

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**

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.hg.dont-find-unregistered**

Data Specification: **<boolean: 0 or 1>**

Default Value: **True**

- ### CVS

Active

When CVS version control support is active

Internal Name: **.versioncontrol.cvs.active**

Data Specification: **[not-active, active-if-project-dir, always-active]**

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: **[not-active, active-if-project-dir, always-active]**

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.

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

Prompt to Update Remote Agent

When enabled, Wing will show a dialog offerring to update any remote agent that does not match Wing's version.

Internal Name: **main.autocheck-remote-agent-version**

Data Specification: **<boolean: 0 or 1>**

Default Value: **True**

SSH Executable

The executable to use in order to run the SSH client on the host where Wing is running. This is used to establish secure SSH tunnels to remote hosts. You must configure SSH and a key server outside of Wing, since Wing assumes it can connect to remote hosts without entering a password.

Internal Name: **main.ssh-executable**

Data Specification: **[None or <type str>]**

Default Value: **None**

SSH Remote TCP/IP Port

The TCP/IP port number to use for the remote end of the SSH tunnel to Wing's remote agent. Using a random port should work in most cases and avoids collisions if there are multiple active sessions, but a fixed port is needed if the random port generated IDE-side is not also available on the remote host.

Internal Name: **main.ssh-remote-port**

Data Specification: **[None or <type int>]**

Default Value: **None**

SSH Timeout

The maximum time in seconds to wait for SSH tunnels to be established.

Internal Name: **main.ssh-timeout**

Data Specification: **<type int>**

Default Value: **10**

Hung Connection Timeout

The maximum time in seconds to wait if a connection to a remote host is not responding. Afterwards the connection is closed and retried.

Internal Name: **main.hung-connection-threshold**

Data Specification: **<type int>**

Default Value: **15**

Warn when Edit Active Remote Configuration

Controls whether to show a warning before editing a remote host configuration tha is currently in use.

Internal Name: **main.show-remote-config-warning**

Data Specification: **<boolean: 0 or 1>**

Default Value: **True**

Use HTTPS to wingware.com

Whether to use secure https (port 443) when accessing wingware.com for license activation, update checks, and submitting feedback or bug reports. When disabled, http (port 80) is used instead.

Internal Name: **main.secure-http-to-wingware**

Data Specification: **<boolean: 0 or 1>**

Default Value: **True**

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.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-update

Used internally to keep track of latest version the user is not interested in

Internal Name: **main.ignored-update**

Data Specification: **[tuple of: <type int>]**

Default Value: **(0, 0, 0, 0)**

main.last-dark-mode

Used internally to remember the most recently used dark mode

Internal Name: **main.last-dark-mode**

Data Specification: **[tuple length 2 of: <type str>, <boolean: 0 or 1>]**

Default Value: **('one-dark', True)**

main.last-light-mode

Used internally to select the most recently used light mode.

Internal Name: **main.last-light-mode**

Data Specification: **[tuple length 2 of: <type str>, <boolean: 0 or 1>]**

Default Value: **('wing-classic', False)**

main.last-prefs-page

Used internally to select the most recently used prefs page.

Internal Name: **main.last-prefs-page**

Data Specification: **[tuple length 2 of: <type int>, <type int>]**

Default Value: **(-1, -1)**

main.last-properties-pages

Used internally to select the most recently used properties dialog pages.

Internal Name: **main.last-properties-pages**

Data Specification: **[dict; keys: <type str>, values: <type int>]**

Default Value: **{}**

main.non-font-scale-factor

Scale factor for icons, windows, and other graphical elements other than fonts. Can either be a single number or a ; (semicolon) separated list of per-screen scale factors in the format used by the QT_SCREEN_SCALE_FACTORS environment variable. This has no effect if the QT_SCREEN_SCALE_FACTORS environment variable is set before Wing is started Wing must be restarted before this value takes effect.

Internal Name: **main.non-font-scale-factor**

Data Specification: **<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.prefs-version

Used internally to identify prefs file version

Internal Name: **main.prefs-version**

Data Specification: **[None or <type int>]**

Default Value: **None**

.main.set-auto-screen-scale-factor

Automatically set scale factor based on screen dpi.

Internal Name: **.main.set-auto-screen-scale-factor**

Data Specification: **<boolean: 0 or 1>**

Default Value: **False**

main.sassist-allow-pep287-errors

Whether to render docstrings even if they contain parse errors at or above the threshold set by Source Assistant PEP 287 Error Threshold. When disabled, failing docstrings are shown as plain

text instead. When enabled, a best effort is made to display the formatted docstring while suppressing errors.

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-pep287-error-level

The error level at or above which the source assistant will display parse errors in PEP287 docstrings (if showing PEP287 errors) or will fall back to showing plain text (if not showing PEP287 errors). For errors below this threshold, a best attempt is made to achieve a reasonable rendering.

Internal Name: **main.sassist-pep287-error-level**

Data Specification: **[0, 1, 2, 3, 4]**

Default Value: **2**

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**

main.suggest-non-font-scale-factor

Whether to suggest per-screen scale factors at startup, based on inspection of font size on each attached display.

Internal Name: **main.suggest-non-font-scale-factor**

Data Specification: **<boolean: 0 or 1>**

Default Value: **True**

main.update-history

History of updates used diagnostically

Internal Name: **main.update-history**

Data Specification: **<type list>**

Default Value: **[]**

User Interface Preferences

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**

guimgr.last-wingtips-size

Internal preference used to remember the last size of the Wing Tips window

Internal Name: **guimgr.last-wingtips-size**

Data Specification: **[any value]**

Default Value: **(500, 450)**

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, editor, scripts, browser], values: [tuple length 3 of: <type str>, [0, 40, 30], <type int>]]**

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.new-tabs-on-left

Whether to add new tabs on the left side instead on the right.

Internal Name: **gui.new-tabs-on-left**

Data Specification: **<boolean: 0 or 1>**

Default Value: **False**

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>, values: <boolean: 0 or 1>]]

Default Value: **{}**

gui.set-win32-foreground-lock-timeout

Controls whether or not to set the foreground lock timeout on Windows, 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**

guimgr.show-menu-bar

Whether to show the menu bar in the window. When this is False, a menu icon is added to the top right.

Internal Name: **guimgr.show-menu-bar**

Data Specification: **<boolean: 0 or 1>**

Default Value: **True**

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.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.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-python-interface', 'text/x-c-source', 'text/x-cpp-source', 'text/x-java-source', 'text/x-javascript', 'text/html', 'text/x-mako', 'text/x-django', 'text/xml', 'text/x-zope-pt', 'text/x-eiffel', 'text/x-lisp', 'text/x-ruby', 'text/x-cython', 'text/x-yaml', 'application/json']**

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.shared-bookmark-categories

Bookmark categories that are shared with all projects.

Internal Name: **edit.shared-bookmark-categories**

Data Specification: **[dict; keys: <type str>, values: <type str>]**

Default Value: **{}**

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.last-anaconda

Used internally to store the last successfully used Anaconda installationfor New Project.

Internal Name: **proj.last-anaconda**

Data Specification: **<type str>**

Default Value: **""**

proj.last-new-project-type

Used internally to store the last used new project type.

Internal Name: **proj.last-new-project-type**

Data Specification: **<type str>**

Default Value: **generic**

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: **first**

debug.array-search-all-columns

Controls whether searching in the debug array view searchs all columns or just the visible columns

Internal Name: **debug.array-search-all-columns**

Data Specification: **<boolean: 0 or 1>**

Default Value: **False**

debug.array-search-case

Selects whether search in the array view is case sensitive

Internal Name: **debug.array-search-case**

Data Specification: **<boolean: 0 or 1>**

Default Value: **False**

debug.array-search-type

Selects the type of search to perform in the array view of debug data: text, wildcard, or regex

Internal Name: **debug.array-search-type**

Data Specification: **[regex, text, wildcard]**

Default Value: **text**

debug.debug-data-vertical

Controls whether the debugger shows value details in data views verticallyor horizontally.

Internal Name: **debug.debug-data-vertical**

Data Specification: **<boolean: 0 or 1>**

Default Value: **1**

debug.debug-io-focus-for-input

Enable to raise the Debug I/O tool and place focus into the I/O buffer whenever the debug process is waiting for keyboard input.

Internal Name: **debug.debug-io-focus-for-input**

Data Specification: **<boolean: 0 or 1>**

Default Value: **False**

debug.debug-io-history

Enable to maintain a history of Debug I/O, up to the number configured in the Files > Max Recent Items preference.

Internal Name: **debug.debug-io-history**

Data Specification: **<boolean: 0 or 1>**

Default Value: **False**

debug.debug-io-history

Enable to include child processes in the process selector popup.

Internal Name: **debug.debug-io-history**

Data Specification: **<boolean: 0 or 1>**

Default Value: **True**

debug.default-python-exec

319

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**

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-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-debug-data-details

Controls whether the debugger shows value details in data views.

Internal Name: **debug.show-debug-data-details**

Data Specification: **<type float>**

Default Value: **0.0**

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 Console.

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**

Source Analysis Preferences

pysource.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**

pysource.use-sqllite-dotfile-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**

Command Reference

This chapter describes the entire top-level command set of Wing. 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).

21.1. 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

apply-update ()

Apply a manually downloaded update

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 Bindings: Wing: Ctrl-Shift-Shift-Tab invokes begin-visited-document-cycle(move_back=False); Brief: Ctrl-Shift-Shift-Tab invokes begin-visited-document-cycle(move_back=False); Eclipse: Ctrl-Shift-Shift-Tab invokes begin-visited-document-cycle(move_back=False); Emacs: Ctrl-Shift-Shift-Tab invokes begin-visited-document-cycle(move_back=False); OS X: Ctrl-Shift-Tab invokes begin-visited-document-cycle(move_back=False); MATLAB: Ctrl-Shift-Shift-Tab invokes begin-visited-document-cycle(move_back=False); VI/VIM: Ctrl-Shift-Shift-Tab invokes begin-visited-document-cycle(move_back=False); Visual Studio: Ctrl-Shift-Shift-Tab invokes begin-visited-document-cycle(move_back=False); XCode: Ctrl-Shift-Tab invokes begin-visited-document-cycle(move_back=False)*

bookmarks-category-menu-items ()

Returns list of menu items for selecting bookmark category

bookmarks-menu-items (names_only=False)

Returns list of menu items for selecting among defined bookmarks

check-for-updates ()

Check for updates to Wing 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: Ctrl-W; Brief: Ctrl-F4; Eclipse: Ctrl-W; Emacs: Ctrl-F4; OS X: Command-Shift-W; MATLAB: Ctrl-W; VI/VIM: Ctrl-W q invokes close(close_window=1); Visual Studio: Ctrl-W; XCode: Command-Shift-W*

close-all (omit_current=False, ignore_changes=False, close_window=False, include_help=True)

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. Also closes documentation views, unless include_help is set to False. *Key Bindings: Eclipse: Ctrl-Shift-W*

close-window ()

Close the current window and all documents and panels in it *Key Bindings: Wing: Alt-F4; Brief: Alt-F4; Eclipse: Alt-F4; Emacs: Ctrl-X 5 0; OS X: Option-F4; MATLAB: Alt-F4; VI/VIM: Alt-F4; Visual Studio: Alt-F4; XCode: Option-F4*

command-by-name (command_name)

Execute given command by name, collecting any args as needed *Key Bindings: Wing: Ctrl-F12; Brief: F10; Eclipse: Ctrl-F12; Emacs: Esc X; OS X: Ctrl-F12; MATLAB: Ctrl-F12; VI/VIM: Ctrl-F12; Visual Studio: Ctrl-/; XCode: Ctrl-F12*

copy-tutorial ()

Prompt user and copy the tutorial directory from the Wing installation to the directory selected by the user

edit-bookmark-categories ()

Edit the defined bookmark categories

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, show=True)

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-!*

export-bookmark-categories (filename)

Export all bookmark categories

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: Ctrl-Alt-G; Eclipse: Ctrl-Alt-G; Emacs: Ctrl-X R B; OS X: Command-Ctrl-B; MATLAB: Ctrl-Alt-G; Visual Studio: Ctrl-Alt-G; XCode: Command-Ctrl-B*

goto-definition (symbol=None, context='selection,path', other_split=None)

Go to the definition of the given symbol, working from the given scope. If symbol is not given then the currently selected symbol is used.

The context can contain one or more of the following in a comma-separated list. They are used in order given and processing stops when a valid definition if found:

- 'selection' to resolve the symbol in the scope of the current editor selection
- 'def' to resolve it in the scope of the point of definition of the current editor selection.
- 'path' to resolve by treating the leading portion as a module Name on the Python Path

If other_split is true, the definition will be displayed if a split other than the current split; if other_split is false, it will be displayed in the current editor; if other_split is not specified or None, the split to be used is determined by the Split Reuse Policy preference value..

goto-next-bookmark (current_file_only=False, category=None)

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. Only bookmarks in the current bookmark category are visited unless a category is passed. *Key Bindings: Wing: Ctrl-Alt-Down invokes goto-next-bookmark(current_file_only=True); Brief: Ctrl-Alt-Down invokes*

goto-next-bookmark(current_file_only=True);   *Eclipse:*   *Ctrl-Alt-Down*   *invokes*
goto-next-bookmark(current_file_only=True);   *Emacs:*   *Ctrl-Alt-Down*   *invokes*
goto-next-bookmark(current_file_only=True); MATLAB: F2; VI/VIM: Ctrl-Alt-Down invokes
goto-next-bookmark(current_file_only=True); Visual Studio: Ctrl-K Ctrl-N

goto-previous-bookmark (current_file_only=False, category=None)

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. Only bookmarks in the current bookmark category are visited unless a category is passed. *Key Bindings: Wing: Ctrl-Alt-Up invokes*
goto-previous-bookmark(current_file_only=True);   *Brief:*   *Ctrl-Alt-Up*   *invokes*
goto-previous-bookmark(current_file_only=True);   *Eclipse:*   *Ctrl-Alt-Up*   *invokes*
goto-previous-bookmark(current_file_only=True);   *Emacs:*   *Ctrl-Alt-Up*   *invokes*
goto-previous-bookmark(current_file_only=True); MATLAB: Shift-F2; VI/VIM: Ctrl-Alt-Up invokes
goto-previous-bookmark(current_file_only=True); Visual Studio: Ctrl-K Ctrl-P

hide-line-numbers ()

Hide line numbers in editors

import-bookmark-categories (filename)

Import bookmark categories

initiate-numeric-modifier (digit)

VI style repeat/numeric modifier for following command *Key Bindings: VI/VIM: 9 invokes*
initiate-numeric-modifier(digit=9)

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-keystroke-logging-start ()

Start logging information about keystroke processing to ide.log

internal-keystroke-logging-stop ()

Stop logging information about keystroke processing to ide.log

internal-logging-start (name=None)

Start logging information about the internal subsystem to ide.log

internal-logging-stop (name=None)

Stop logging information about the internal subsystem to ide.log

internal-profile-start (print_freq=0, print_top_n=40)

Start internal profiling. Profile information is collected for Wing's internals until internal_profile_stop is executed. If the print_freq argument is > 0, stats will be printed to ide.log every print_freq seconds. The print_top_n arg specifies the number of top functions to print.

internal-profile-stop ()

Stop internal profiling after earlier internal_profile_start command. The profile can be found in the ide.log file or submitted to Wingware as part of the error log included with a bug report from the Help menu.

internal-tooltip-logging-start ()

Start logging information about tooltip processing to ide.log

internal-tooltip-logging-stop ()

Stop logging information about tooltip processing to ide.log

maximize-editor-to-window ()

Move the current editor out of the main document window and into its own editor-only window *Key Bindings: MATLAB: Ctrl-Shift-U*

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 3; OS X: Shift-F4; XCode: Shift-F4*

new-file (ext='.py')

Create a new file *Key Bindings: Wing: Ctrl-N; Eclipse: Ctrl-N; OS X: Command-N; MATLAB: Ctrl-N; Visual Studio: Ctrl-N; XCode: Command-T*

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: Ctrl-0; Brief: Alt-N; Eclipse: Ctrl-F6; Emacs: Ctrl-X N; OS X: Command-0; MATLAB: Ctrl-PageDown; VI/VIM: g T; Visual Studio: Ctrl-0; XCode: Command-}*

next-window ()

Switch to the next window alphabetically by title *Key Bindings: Wing: Ctrl-Comma; Eclipse: Ctrl-Comma; Emacs: Ctrl-X 5 O; MATLAB: Ctrl-Comma; 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: Ctrl-K; Eclipse: Ctrl-K; Emacs: Ctrl-X Ctrl-F; MATLAB: Ctrl-K; 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: Ctrl-Shift-O; Eclipse: Ctrl-Shift-R; Emacs: Ctrl-X Ctrl-O; OS X: Command-Shift-O; MATLAB: Ctrl-Shift-F; VI/VIM: Ctrl-Shift-O; Visual Studio: Ctrl-Shift-O; XCode: Command-Shift-O*

open-gui (filename=None)

Open a file from local disk or a remote host, prompting with file selection dialog if necessary. The dialog shown depends on the default starting directory, and may be for local files or remote files. *Key Bindings: Wing: Ctrl-O; Brief: Alt-E; Eclipse: Ctrl-O; OS X: Command-O; MATLAB: Ctrl-O; Visual Studio: Ctrl-O; XCode: Command-O*

open-local (filename=None)

Prompt user to open a file from local disk

open-remote ()

Prompt user to open a file from a remote host

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.

perspective-update-with-current-state (name=None)

Update the perspective with the current state. If no name is given, the active perspective is used.

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: Ctrl-9; Brief: Alt--; Eclipse: Ctrl-9; Emacs: Ctrl-X P; OS X: Command-9; MATLAB: Ctrl-PageUp; VI/VIM: g Shift-T; Visual Studio: Ctrl-9; XCode: Command-{*

previous-window ()

Switch to the previous window alphabetically by title

quit ()

Quit the application. *Key Bindings: Wing: Ctrl-Q; Brief: Alt-X; Eclipse: Ctrl-Q; Emacs: Ctrl-X Ctrl-C; OS X: Command-Q; MATLAB: Alt-F4; Visual Studio: Ctrl-Q; XCode: 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: Ctrl-8; Eclipse: Ctrl-8; Emacs: Ctrl-X D; OS X: Command-8; MATLAB: Ctrl-8; Visual Studio: Ctrl-8; XCode: 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, confirm=False)

Remove the given named bookmark, optionally confirming the removal with the user.

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: Ctrl-S; Brief: Alt-W; Eclipse: Ctrl-S; Emacs: Ctrl-X Ctrl-S; OS X: Command-S; MATLAB: Ctrl-S; VI/VIM: Ctrl-S; Visual Studio: Ctrl-S; XCode: 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: Ctrl-Shift-S; Eclipse: Ctrl-Shift-S; OS X: Command-Shift-S; MATLAB: Ctrl-Shift-S; XCode: Command-Shift-S*

save-as-remote ()

Save active document to a new file on a remote host

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, the category is set to the current bookmark category, and notes are left blank. *Key Bindings: Wing: Ctrl-Alt-M; Brief: Alt-9 invokes set-bookmark(mark="9"); Eclipse: Ctrl-Alt-M; Emacs: Ctrl-X R M; OS X: Command-B; MATLAB: Ctrl-Alt-M; Visual Studio: Ctrl-Alt-M; XCode: Command-B*

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. The bookmark's category is set to the current bookmark category, and notes are left blank.

set-bookmark-dialog ()

Set a bookmark at the current location on the editor using a dialog to set the bookmark name, category, and notes. The default name is auto-generated based on location, and default category is set to the current bookmark category.

set-bookmark-dialog-at-click ()

Set a bookmark at the clicked location on the editor using a dialog to set the bookmark name, category, and notes. The default name is auto-generated based on location, and default category is set to the current bookmark category.

show-bookmarks ()

Show a list of all currently defined bookmarks *Key Bindings: Wing: Ctrl-Alt-K; Brief: Alt-J; Eclipse: Ctrl-Alt-K; Emacs: Ctrl-X R Return; OS X: Command-Shift-K; MATLAB: Ctrl-Alt-K; Visual Studio: Ctrl-Alt-K; XCode: 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-?; XCode: Command-Alt-?*

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 users manual

show-manual-pdf ()

Show the PDF version of the Wing 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-console (**) 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: Alt-Shift-T invokes show-panel(panel_type="refactoring");* MATLAB: *F1* invokes *show-panel(panel_type="source-assistant")*

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-code-warnings (flash=True, grab_focus=None)

Not documented

show-panel-debug-breakpoints (flash=True, grab_focus=None)

Not documented

show-panel-debug-console (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 *Key Bindings: XCode: Command-Alt-Ctrl-2*

show-panel-source-assistant (flash=True, grab_focus=None)

Not documented *Key Bindings: XCode: Command-Alt-Ctrl-/*

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-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; XCode: 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-qa-html ()

Show the Wing Q&A site

show-quickstart ()

Show the quick start guide

show-success-stories-html ()

Show the Python Success Stories page

show-support-html ()

Show the Wing 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 contributed materials area

start-terminal ()

Start a terminal in the OS Commands tool

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*

terminate-os-command (title)

Terminate one of the stored commands in the OS Commands tool, selecting it by its title

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, the category is set to the current bookmark category, and notes are left blank. When removed, the bookmark is removed without confirmation. *Key Bindings: Wing: Ctrl-Alt-T; Eclipse: Ctrl-Alt-T; Emacs: Ctrl-X R T; OS X: Command-Shift-B; MATLAB: Ctrl-F2; Visual Studio: Ctrl-K Ctrl-K; XCode: Command-Shift-B*

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, the category is set to the current bookmark category, and notes are left blank. When removed, the bookmark is removed without confirmation.

toggle-line-numbers ()

Toggle whether or not line numbers are shown in editors

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: Ctrl-Alt-D; Eclipse: Ctrl-Alt-D; MATLAB: Ctrl-Alt-D; Visual Studio: Ctrl-K 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

unmaximize-editors-from-window ()

Move all the editors in the current editor-only window back into the main document window and close the editor-only window. A new main document window is created if none currently exists. *Key Bindings: MATLAB: Ctrl-Shift-D*

validate-install ()

Validate the Wing installation, checking that all files are present and have the expected contents

vi-delete-bookmark (marks)

Remove one or more bookmarks without confirmation (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: '*

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 Bindings: Wing: Shift-F2; Brief: Shift-F2; Eclipse: Ctrl-M;*

Command Reference

Emacs: Shift-F2; OS X: Shift-F2; MATLAB: Ctrl-Shift-M; VI/VIM: Shift-F2; Visual Studio: Shift-F2; XCode: Shift-F2

exit-fullscreen ()

Restore previous non-fullscreen state of all tools and tool bar *Key Bindings: Wing: Shift-F2; Brief: Shift-F2; Eclipse: Ctrl-M; Emacs: Shift-F2; OS X: Shift-F2; MATLAB: Ctrl-Shift-M; VI/VIM: Shift-F2; Visual Studio: Shift-F2; XCode: 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 *Key Bindings: XCode: Command-Shift-Y*

toggle-vertical-tools ()

Show or minimize the vertical tool area *Key Bindings: XCode: Command-0*

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: Ctrl-C; Brief: Ctrl-C; Eclipse: Ctrl-C; Emacs: Alt-W; OS X: Command-C; MATLAB: Ctrl-C; VI/VIM: Ctrl-Insert; Visual Studio: Ctrl-C; XCode: 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: Ctrl-Shift-U; Eclipse: Ctrl-Shift-J; Emacs: Ctrl-R; OS X: Command-Shift-U; MATLAB: Ctrl-Shift-R; Visual Studio: Ctrl-Shift-U; XCode: 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: Emacs: Ctrl-Alt-R; VI/VIM: ?*

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: Ctrl-U; Eclipse: Ctrl-J; Emacs: Ctrl-S; OS X: Command-U; MATLAB: Ctrl-Shift-S; Visual Studio: Ctrl-I; XCode: 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: Emacs: Ctrl-Alt-S; VI/VIM: /*

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: Shift-N invokes isearch-repeat(reverse=1)*

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: Ctrl-Shift-B; Eclipse: Ctrl-Shift-B; Emacs: Ctrl-C R; MATLAB: Ctrl-Shift-B; VI/VIM: # invokes isearch-sel-backward(persist=0, whole_word=1); 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: Ctrl-B; Eclipse: Ctrl-B; Emacs: Ctrl-C S; MATLAB: Ctrl-B; VI/VIM: * invokes isearch-sel-forward(persist=0, whole_word=1); 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 *Key Binding: Ctrl-_*

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: Ctrl-Shift-E; XCode: Command-J*

move-editor-focus (dir=1, wrap=True)

Move focus to next or previous editor split, optionally wrapping when the end is reached. *Key Bindings: Emacs: Ctrl-X O; MATLAB: F6; VI/VIM: Ctrl-W W invokes move-editor-focus(dir=-1)*

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*

next-tool (wrap=True)

Show the next tool, starting with most recently shown tool *Key Bindings: MATLAB: Ctrl-F6*

prev-tool (wrap=True)

Show the previous tool, starting with the move recently shown tool *Key Bindings: MATLAB: Ctrl-Shift-F6*

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

21.2. 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: Ctrl-Shift-I; Brief: Ctrl-Shift-I; Eclipse: Ctrl-Shift-I; Emacs: Ctrl-Shift-I; OS X: Command-Shift-I; MATLAB: Ctrl-Shift-I; VI/VIM: Ctrl-Shift-I; Visual Studio: Ctrl-Shift-I; XCode: 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 ()

Not documented

clear-project-main-entry-point ()

Clear main entry point to nothing, so that debugging and execution starts with the file in the current editor

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-project-remote ()

Open a project file from a remote host

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.

save-project-as-remote (filename=None)

Save current project to a remote host

set-current-as-main-debug-file ()

Not documented

set-current-as-main-entry-point ()

Set current editor file as the main entry point for this project

set-selected-as-main-debug-file ()

Not documented

set-selected-as-main-entry-point ()

Set selected file as the main entry point for this project

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 Personal.

use-single-user-project ()

Store project single-user (one file) format, which can also be read by Wing 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; XCode: 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 ()

Not documented

set-selected-as-main-entry-point ()

Set selected file as the main entry point 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

21.3. 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: Shift-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: Shift-Ctrl-F8; Brief: Shift-Ctrl-F8; Eclipse: Shift-Ctrl-F8; Emacs: Shift-Ctrl-F8; OS X: Shift-Command-F8; MATLAB: Shift-Ctrl-F8; VI/VIM: Ctrl-Q; Visual Studio: Shift-Ctrl-F8; XCode: Shift-Command-F8*

start-select-char ()

Turn on auto-select mode character by character *Key Bindings: Wing: Shift-F8; Brief: Shift-F8; Eclipse: Shift-F8; Emacs: Shift-F8; OS X: Shift-F8; MATLAB: Shift-F8; VI/VIM: v; Visual Studio: Shift-F8; XCode: Shift-F8*

start-select-line ()

Turn on auto-select mode line by line *Key Bindings: Wing: Ctrl-F8; Brief: Ctrl-F8; Eclipse: Ctrl-F8; Emacs: Ctrl-F8; OS X: Command-F8; MATLAB: Ctrl-F8; VI/VIM: Shift-V; Visual Studio: Ctrl-F8; XCode: Command-F8*

vi-command-by-name ()

Execute a VI command by name. This implements ":" commands for the VI/Vim keyboard personality. The following subset of VI/Vim : commands are supported:

```
r[!], e[dit], e!, e#, ene[w], w[rite], up[date], wa[ll], q[uit], q[!], qall, wq,
x[it], xall, wqall, sp[lit], vs[plit], new, on[ly], buffers, files, !, s[ubstitute],
d, delm, reg, marks, n[ext], N, p[revious], rew[ind], last, m[ove], co[py], cl[ose]
(an approximation), and set.
```

The supported directives for 'set' are:

```
ic, ignorecase, noic, noignorecase, ai, autoindent, noai, noautoindent, nu, number,
nonu, nonumber, ro, readonly, noro, noreadonly, sm, showmatch, nosm, and noshowmatch.
    *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: Ctrl-V*

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 Bindings: Wing: Esc; Brief: Esc; Eclipse: Esc; Emacs: Esc; OS X: Esc; MATLAB: Esc; VI/VIM: Ctrl-[; Visual Studio: Esc; XCode: Esc*

start-select-block ()

Turn on auto-select block mode *Key Bindings: Wing: Shift-Ctrl-F8; Brief: Shift-Ctrl-F8; Eclipse: Shift-Ctrl-F8; Emacs: Shift-Ctrl-F8; OS X: Shift-Command-F8; MATLAB: Shift-Ctrl-F8; VI/VIM: Ctrl-Q; Visual Studio: Shift-Ctrl-F8; XCode: Shift-Command-F8*

start-select-char ()

Turn on auto-select mode character by character *Key Bindings: Wing: Shift-F8; Brief: Shift-F8; Eclipse: Shift-F8; Emacs: Shift-F8; OS X: Shift-F8; MATLAB: Shift-F8; VI/VIM: v; Visual Studio: Shift-F8; XCode: Shift-F8*

start-select-line ()

Turn on auto-select mode line by line *Key Bindings: Wing: Ctrl-F8; Brief: Ctrl-F8; Eclipse: Ctrl-F8; Emacs: Ctrl-F8; OS X: Command-F8; MATLAB: Ctrl-F8; VI/VIM: Shift-V; Visual Studio: Ctrl-F8; XCode: 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: Emacs: Ctrl-X 3; VI/VIM: Ctrl-W v*

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; Emacs: Ctrl-X 2; VI/VIM: Ctrl-W n invokes split-vertically(new=1)*

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.  *Key Bindings: Brief: F4; Emacs: Ctrl-X 1; VI/VIM: Ctrl-W o*
```

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: Ctrl-V*

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: Ctrl-1; Brief: Ctrl-1; Eclipse: Ctrl-1; Emacs: Ctrl-1; OS X: Command-1; MATLAB: Ctrl-1; VI/VIM: Ctrl-1; Visual Studio: Ctrl-1; XCode: 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: Forward-button-click; Brief: Forward-button-click; Eclipse: Alt-Right; Emacs: Forward-button-click; OS X: Forward-button-click; MATLAB: Forward-button-click; VI/VIM: Ctrl-I; Visual Studio: Ctrl-_; XCode: Command-Ctrl-Right*

visit-history-previous ()

Move back in history to previous visited editor position *Key Bindings: Wing: Back-button-click; Brief: Back-button-click; Eclipse: Ctrl-Q; Emacs: Back-button-click; OS X: Back-button-click; MATLAB: Back-button-click; VI/VIM: Ctrl-O; Visual Studio: Ctrl--; XCode: Command-Ctrl-Left*

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: Ctrl-V*

enter-insert-mode (pos='delete-sel')

Enter editor insert mode *Key Bindings: VI/VIM: Shift-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 Bindings: Wing: Esc; Brief: Esc; Eclipse: Esc; Emacs: Esc; OS X: Esc; MATLAB: Esc; VI/VIM: Ctrl-[; Visual Studio: Esc; XCode: Esc*

select-inner (extend=False)

Select a text object based on the following key press *Key Bindings: VI/VIM: a invokes select-inner(extend=True)*

vi-command-by-name ()

Execute a VI command by name. This implements ":" commands for the VI/Vim keyboard personality. The following subset of VI/Vim : commands are supported:

```
r[!], e[dit], e!, e#, ene[w], w[rite], up[date], wa[ll], q[uit], q[!], qall, wq,
x[it], xall, wqall, sp[lit], vs[plit], new, on[ly], buffers, files, !, s[ubstitute],
d, delm, reg, marks, n[ext], N, p[revious], rew[ind], last, m[ove], co[py], cl[ose]
(an approximation), and set.
```

The supported directives for 'set' are:

```
ic, ignorecase, noic, noignorecase, ai, autoindent, noai, noautoindent, nu, number,
nonu, nonumber, ro, readonly, noro, noreadonly, sm, showmatch, nosm, and noshowmatch.
    *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 Console, 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: Ctrl-2; Brief: Ctrl-2; Eclipse: Ctrl-2; Emacs: Ctrl-2; OS X: Command-2; MATLAB: Ctrl-2; VI/VIM: Ctrl-2; Visual Studio: Ctrl-2; XCode: Command-2*

activate-symbol-option-menu-2 ()

Activate the 2nd symbol menu for the editor. *Key Bindings: Wing: Ctrl-3; Brief: Ctrl-3; Eclipse: Ctrl-3; Emacs: Ctrl-3; OS X: Command-3; MATLAB: Ctrl-3; VI/VIM: Ctrl-3; Visual Studio: Ctrl-3; XCode: Command-3*

activate-symbol-option-menu-3 ()

Activate the 3rd symbol menu for the editor. *Key Bindings: Wing: Ctrl-4; Brief: Ctrl-4; Eclipse: Ctrl-4; Emacs: Ctrl-4; OS X: Command-4; MATLAB: Ctrl-4; VI/VIM: Ctrl-4; Visual Studio: Ctrl-4; XCode: Command-4*

activate-symbol-option-menu-4 ()

Activate the 4th symbol menu for the editor. *Key Bindings: Wing: Ctrl-5; Brief: Ctrl-5; Eclipse: Ctrl-5; Emacs: Ctrl-5; OS X: Command-5; MATLAB: Ctrl-5; VI/VIM: Ctrl-5; Visual Studio: Ctrl-5; XCode: Command-5*

activate-symbol-option-menu-5 ()

Activate the 5th symbol menu for the editor. *Key Bindings: Wing: Ctrl-6; Brief: Ctrl-6; Eclipse: Ctrl-6; Emacs: Ctrl-6; OS X: Command-6; MATLAB: Ctrl-6; VI/VIM: Ctrl-6; Visual Studio: Ctrl-6; XCode: Command-6*

backward-char (wrap=1, repeat=<numeric modifier; default=1>)

Move cursor backward one character *Key Bindings: Wing: Left; Brief: Left; Eclipse: Left; Emacs: Ctrl-B; OS X: Ctrl-b; MATLAB: Left; VI/VIM: Ctrl-h; Visual Studio: Left; XCode: Ctrl-b*

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: Shift-Alt-Left; Brief: Shift-Alt-Left; Eclipse: Shift-Alt-Left; Emacs: Shift-Alt-Left; OS X: Ctrl-Option-Left; MATLAB: Shift-Alt-Left; VI/VIM: Shift-Alt-Left; Visual Studio: Shift-Alt-Left; XCode: 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: Shift-BackSpace; Brief: Shift-BackSpace; Eclipse: Shift-BackSpace; Emacs: Ctrl-H; OS X: Ctrl-h; MATLAB: Shift-BackSpace; VI/VIM: Ctrl-H; Visual Studio: Shift-BackSpace; XCode: Ctrl-h*

backward-delete-word (delimiters=None, repeat=<numeric modifier; default=1>)

Delete one word behind of the cursor *Key Bindings: Wing: Alt-Delete; Brief: Alt-Delete; Eclipse: Alt-Delete; Emacs: Alt-Delete; OS X: Option-Backspace; MATLAB: Alt-Delete; VI/VIM: Ctrl-W; Visual Studio: Alt-Delete; XCode: Option-Backspace*

backward-page (repeat=<numeric modifier; default=1>)

Move cursor backward one page *Key Bindings: Wing: Ctrl-Prior; Brief: Ctrl-Prior; Eclipse: Ctrl-Prior; Emacs: Alt-V; OS X: Option-Page_Up; MATLAB: Ctrl-Prior; VI/VIM: Ctrl-B; Visual Studio: Ctrl-Prior; XCode: Option-Page_Up*

backward-page-extend (repeat=<numeric modifier; default=1>)

Move cursor backward one page, adjusting the selection range to new position *Key Bindings: Wing: Ctrl-Shift-Page_Up; Brief: Ctrl-Shift-Page_Up; Eclipse: Ctrl-Shift-Page_Up; Emacs: Ctrl-Shift-Page_Up; OS X: Shift-Page_Up; MATLAB: Ctrl-Shift-Page_Up; VI/VIM: Ctrl-Shift-Page_Up; Visual Studio: Ctrl-Shift-Page_Up; XCode: 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: Ctrl-Left; Brief: Ctrl-Left; Eclipse: Ctrl-Left; Emacs: Alt-B; OS X: Ctrl-Left invokes backward-word(delimiters="_`~!@#$%^&*()+-={}[]\|;:'",.<>/? trn"); MATLAB: Ctrl-Left; VI/VIM: Ctrl-W; Visual Studio: Ctrl-Left; XCode: Ctrl-Left invokes backward-word(delimiters="_`~!@#$%^&*()+-={}[]\|;:'",.<>/? trn")*

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: Ctrl-Shift-Left; Brief: Ctrl-Shift-Left; Eclipse: Ctrl-Shift-Left; Emacs: Ctrl-Shift-Left; OS X: Option-Shift-Left; MATLAB: Ctrl-Shift-Left; VI/VIM: Ctrl-Shift-Left; Visual Studio: Ctrl-Shift-Left; XCode: Option-Shift-Left*

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; Emacs: Ctrl-A; OS X: Ctrl-a; VI/VIM: 0 invokes beginning-of-line(toggle=0); XCode: Ctrl-a*

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; XCode: 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: Home; Brief: Home; Eclipse: Home; Emacs: Home; MATLAB: Home; VI/VIM: _; 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: Shift-Home; Brief: Shift-Home; Eclipse: Shift-Home; Emacs: Shift-Home; MATLAB: Shift-Home; VI/VIM: 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: Ctrl-]; Eclipse: Ctrl-Shift-P; Emacs: Ctrl-M; OS X: Command-); MATLAB: Ctrl-]; Visual Studio: Ctrl-]; XCode: Command-)*

cancel ()

Cancel current editor command

cancel-autocompletion ()

Cancel any active autocompletion.

Command Reference

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; Emacs: Ctrl-L; MATLAB: Ctrl-G; VI/VIM: z z*

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 ()

Command Reference

Copy selected text *Key Bindings: Wing: Ctrl-C; Brief: Ctrl-C; Eclipse: Ctrl-C; Emacs: Alt-W; OS X: Command-C; MATLAB: Ctrl-C; VI/VIM: Ctrl-Insert; Visual Studio: Ctrl-C; XCode: 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 b*

cursor-to-top ()

Scroll so cursor is centered at top of display *Key Bindings: VI/VIM: z +*

cut ()

Cut selected text *Key Bindings: Wing: Ctrl-X; Brief: Ctrl-X; Eclipse: Ctrl-X; Emacs: Ctrl-W; OS X: Command-X; MATLAB: Ctrl-X; VI/VIM: Shift-Delete; Visual Studio: Ctrl-X; XCode: 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: Ctrl-Shift-C; Eclipse: Ctrl-D; MATLAB: 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: MATLAB: Ctrl-K; 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; XCode: Command-Backspace*

drop-current-selection ()

Drop current selection when there's 2+ selections

drop-extra-selections ()

Drop all exceptions except the main selection

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: Ctrl-Shift-V; Eclipse: Ctrl-Alt-Down; MATLAB: Ctrl-Shift-V*

duplicate-line-above ()

Duplicate the current line or lines above the selection. *Key Bindings: Wing: Ctrl-Shift-Y; Eclipse: Ctrl-Alt-Up; MATLAB: 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: Ctrl-< invokes enclose(start="<", end=">"); Brief: Ctrl-< invokes enclose(start="<", end=">"); Eclipse: Ctrl-< invokes enclose(start="<", end=">"); Emacs: Ctrl-< invokes enclose(start="<", end=">"); MATLAB:*

Ctrl-< invokes enclose(start="<", end=">"); VI/VIM: Ctrl-< invokes enclose(start="<", end=">"); Visual Studio: Ctrl-< invokes enclose(start="<", end=">")

end-of-document ()

Move cursor to end of document *Key Bindings: Wing: Ctrl-End; Brief: Ctrl-PageDown; Eclipse: Ctrl-End; Emacs: Ctrl-X]; OS X: Command-Down; MATLAB: Ctrl-End; VI/VIM: Ctrl-End; Visual Studio: Ctrl-End; XCode: Command-Down*

end-of-document-extend ()

Move cursor to end of document, adjusting the selection range to new position *Key Bindings: Wing: Ctrl-Shift-End; Brief: Ctrl-Shift-End; Eclipse: Ctrl-Shift-End; Emacs: Ctrl-Shift-End; OS X: Shift-End; MATLAB: Ctrl-Shift-End; VI/VIM: Ctrl-Shift-End; Visual Studio: Ctrl-Shift-End; XCode: Shift-End*

end-of-line (count=<numeric modifier; default=1>)

Move to end of current line *Key Bindings: Wing: End; Brief: Shift-End; Eclipse: End; Emacs: Ctrl-E; OS X: Ctrl-e; MATLAB: Ctrl-E; VI/VIM: $; Visual Studio: End; XCode: Ctrl-e*

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: Shift-End; Brief: Shift-End; Eclipse: Shift-End; Emacs: Shift-End; OS X: Command-Shift-Right; MATLAB: Shift-End; VI/VIM: Shift-End; Visual Studio: Shift-End; XCode: 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: Emacs: Ctrl-X Ctrl-X; VI/VIM: Shift-O*

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 Bindings: Wing: Right; Brief: Right; Eclipse: Right; Emacs: Ctrl-F; OS X: Ctrl-f; MATLAB: Right; VI/VIM: l invokes forward-char(wrap=0); Visual Studio: Right; XCode: Ctrl-f*

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: Shift-Alt-Right; Brief: Shift-Alt-Right; Eclipse: Shift-Alt-Right; Emacs: Shift-Alt-Right; OS X: Ctrl-Option-Right; MATLAB: Shift-Alt-Right; VI/VIM: Shift-Alt-Right; Visual Studio: Shift-Alt-Right; XCode: Ctrl-Option-Right*

forward-delete-char (repeat=<numeric modifier; default=1>)

Delete one character in front of the cursor *Key Bindings: Wing: Delete; Brief: Delete; Eclipse: Delete; Emacs: Ctrl-D; OS X: Ctrl-d; MATLAB: Delete; VI/VIM: Delete; Visual Studio: Delete; XCode: Ctrl-d*

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: Ctrl-Delete; Brief: Ctrl-K; Eclipse: Ctrl-Delete; Emacs: Alt-D; OS X: Option-Delete; MATLAB: Ctrl-Delete; VI/VIM: Ctrl-Delete; Visual Studio: Ctrl-Delete; XCode: 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: Ctrl-Next; Brief: Ctrl-Next; Eclipse: Ctrl-Next; Emacs: Ctrl-V; OS X: Ctrl-v; MATLAB: Ctrl-Next; VI/VIM: Ctrl-F; Visual Studio: Ctrl-Next; XCode: Ctrl-v*

forward-page-extend (repeat=<numeric modifier; default=1>)

Move cursor forward one page, adjusting the selection range to new position *Key Bindings: Wing: Ctrl-Shift-Page_Down; Brief: Ctrl-Shift-Page_Down; Eclipse: Ctrl-Shift-Page_Down; Emacs: Ctrl-Shift-Page_Down; OS X: Shift-Page_Down; MATLAB: Ctrl-Shift-Page_Down; VI/VIM: Ctrl-Shift-Page_Down; Visual Studio: Ctrl-Shift-Page_Down; XCode: 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 Bindings: Wing: Ctrl-T; Brief: Ctrl-T; Eclipse: Ctrl-T; Emacs: Ctrl-T; OS X: Ctrl-T; MATLAB: Tab; VI/VIM: Ctrl-T; Visual Studio: Ctrl-T; XCode: 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: Ctrl-Right; Brief: Ctrl-Right; Eclipse: Ctrl-Right; Emacs: Alt-F; OS X: Option-Right; MATLAB: Ctrl-Right; VI/VIM: Shift-E invokes forward-word(delimiters=" tnr", gravity="endm1"); Visual Studio: Ctrl-Right; XCode: Option-Right*

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: Ctrl-Shift-Right; Brief: Ctrl-Shift-Right; Eclipse: Ctrl-Shift-Right; Emacs: Ctrl-Shift-Right; OS X: Ctrl-Shift-Right invokes forward-word-extend(delimiters="_`~!@#$%^&*()+-={}[]\|;:'",.<>/? trn"); MATLAB: Ctrl-Shift-Right; VI/VIM: Ctrl-Shift-Right; Visual Studio: Ctrl-Shift-Right; XCode: Ctrl-Shift-Right invokes forward-word-extend(delimiters="_`~!@#$%^&*()+-={}[]\|;:'",.<>/? trn")*

goto-overridden-method ()

Goes to the method that is overridden by the current method

hide-selection ()

Turn off display of the current text selection

hide-selections-popup ()

Hide the selections popup; this overrides the preference setting for the current file

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 Bindings:*

Wing: Ctrl-=; Brief: Ctrl-=; Eclipse: Ctrl-=; Emacs: Ctrl-=; OS X: Command-; MATLAB: Ctrl-=; VI/VIM: Ctrl-=; Visual Studio: Ctrl-=; XCode: Ctrl-I

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: Ctrl-Shift-U; Eclipse: Ctrl-Shift-J; Emacs: Ctrl-R; OS X: Command-Shift-U; MATLAB: Ctrl-Shift-R; Visual Studio: Ctrl-Shift-U; XCode: 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: Emacs: Ctrl-Alt-R; VI/VIM: ?*

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: Ctrl-U; Eclipse: Ctrl-J; Emacs: Ctrl-S; OS X: Command-U; MATLAB: Ctrl-Shift-S; Visual Studio: Ctrl-I; XCode: 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: Emacs: Ctrl-Alt-S; VI/VIM: /*

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: Shift-N invokes isearch-repeat(reverse=1)*

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: Ctrl-Shift-B; Eclipse: Ctrl-Shift-B; Emacs: Ctrl-C R; MATLAB: Ctrl-Shift-B; VI/VIM: # invokes isearch-sel-backward(persist=0, whole_word=1); 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: Ctrl-B; Eclipse: Ctrl-B; Emacs: Ctrl-C S; MATLAB: Ctrl-B; VI/VIM: * invokes isearch-sel-forward(persist=0, whole_word=1); 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-K; Emacs: Ctrl-K; OS X: Ctrl-k; XCode: 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: Ctrl-Shift-Down; Eclipse: Alt-Down invokes move-line-down(indent=True); MATLAB: Ctrl-Shift-Down; XCode: Command-Alt-]*

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: Ctrl-Shift-Up; Eclipse: Alt-Up invokes move-line-up(indent=True); MATLAB: Ctrl-Shift-Up; XCode: Command-Alt-[*

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 (auto_indent=None)

Place a new line at the current cursor position. Override the auto-indent preference by setting auto_indent to 'never' to avoid indent, 'always' to auto-indent, and 'blank-only' to auto-indent only on blank lines. *Key Bindings: Wing: Alt-Return; Brief: Alt-Return; Eclipse: Alt-Return; Emacs: Alt-Return; OS X: Option-Return; MATLAB: Shift-Return invokes new-line(auto_indent="never"); VI/VIM: Ctrl-J; Visual Studio: Alt-Return; XCode: Option-Return*

new-line-after ()

Place a new line after the current line *Key Bindings: Wing: Ctrl-Return; Brief: Ctrl-Return; Eclipse: Shift-Enter; Emacs: Ctrl-Return; MATLAB: Ctrl-Return; VI/VIM: Ctrl-Return; Visual Studio: Ctrl-Return*

new-line-before ()

Place a new line before the current line *Key Bindings: Wing: Shift-Return; Brief: Shift-Return; Eclipse: Ctrl-Shift-Enter; Emacs: Shift-Return; MATLAB: Shift-Return; VI/VIM: 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. *Key Bindings: MATLAB: Ctrl-Down*

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 Bindings: Wing: Down; Brief: Down; Eclipse: Down; Emacs: Ctrl-N; OS X: Ctrl-n; MATLAB: Down; VI/VIM: Ctrl-N; Visual Studio: Down; XCode: Ctrl-n*

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 Bindings: Wing: Shift-Down; Brief: Shift-Down; Eclipse: Shift-Down; Emacs: Shift-Down; OS X: Shift-Alt-Down invokes next-line-extend(cursor="xcode"); MATLAB: Shift-Down; VI/VIM: Shift-Down; Visual Studio: Shift-Down; XCode: Shift-Alt-Down invokes next-line-extend(cursor="xcode")*

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: Shift-Alt-Down; Brief: Shift-Alt-Down; Eclipse: Shift-Alt-Down; Emacs: Shift-Alt-Down; OS X: Ctrl-Option-Down; MATLAB: Shift-Alt-Down; VI/VIM: Shift-Alt-Down; Visual Studio: Shift-Alt-Down; XCode: 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: Ctrl-M 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: Ctrl-V; Brief: Ctrl-V; Eclipse: Ctrl-V; Emacs: Ctrl-Y; OS X: Command-V; MATLAB: Ctrl-V; VI/VIM: Shift-Insert; Visual Studio: Ctrl-V; XCode: Command-V*

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: g Shift-P invokes paste-register(pos=-1, cursor=1)*

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. *Key Bindings: MATLAB: Ctrl-Up*

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 Bindings: Wing: Up; Brief: Up; Eclipse: Up; Emacs: Ctrl-P; OS X: Ctrl-p; MATLAB: Up; VI/VIM: Ctrl-P; Visual Studio: Up; XCode: Ctrl-p*

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 Bindings: Wing: Shift-Up; Brief: Shift-Up; Eclipse: Shift-Up; Emacs: Shift-Up; OS X: Shift-Alt-Up invokes previous-line-extend(cursor="xcode"); MATLAB: Shift-Up; VI/VIM: Shift-Up; Visual Studio: Shift-Up; XCode: Shift-Alt-Up invokes previous-line-extend(cursor="xcode")*

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: Shift-Alt-Up; Brief: Shift-Alt-Up; Eclipse: Shift-Alt-Up; Emacs: Shift-Alt-Up; OS X: Ctrl-Option-Up; MATLAB: Shift-Alt-Up; VI/VIM: Shift-Alt-Up; Visual Studio: Shift-Alt-Up; XCode: 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: Ctrl-Shift-Z; Brief: Ctrl-U; Eclipse: Ctrl-Shift-Z; Emacs: Ctrl-.; OS X: Command-Shift-Z; MATLAB: Alt-Shift-Backspace; VI/VIM: Ctrl-R; Visual Studio: Ctrl-Shift-Z; XCode: 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: , invokes repeat-search-char(opposite=1)*

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: Ctrl-Shift-Down; Brief: Ctrl-D; Eclipse: Ctrl-Shift-Down; Emacs: Ctrl-Shift-Down; MATLAB: Ctrl-Down; VI/VIM: Ctrl-D invokes scroll-text-down(repeat=0.5); 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 Shift-L invokes scroll-text-left(repeat=0.5)*

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 Shift-H invokes scroll-text-right(repeat=0.5)*

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: Ctrl-Shift-Up; Brief: Ctrl-E; Eclipse: Ctrl-Shift-Up; Emacs: Ctrl-Shift-Up; MATLAB: Ctrl-Up; VI/VIM: Ctrl-U invokes scroll-text-up(repeat=0.5); 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; XCode: 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; XCode: 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: Shift-T invokes search-char(dir=-1, pos=1, single_line=1)*

select-all ()

Select all text in the editor *Key Bindings: Wing: Ctrl-A; Eclipse: Ctrl-A; OS X: Command-A; MATLAB: Ctrl-A; Visual Studio: Ctrl-A; XCode: 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: Ctrl-Down; Brief: Ctrl-Down; Eclipse: Alt-Shift-Down; Emacs: Ctrl-Down; MATLAB: Ctrl-Down; VI/VIM: 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: Ctrl-Up; Brief: Ctrl-Up; Eclipse: Alt-Shift-Up; Emacs: Ctrl-Up; OS X: Option-Up; MATLAB: Ctrl-Up; VI/VIM: Ctrl-Up; Visual Studio: Ctrl-Up; XCode: Option-Up*

select-scope ()

Select the scope the cursor is in.

select-statement ()

Select the statement the cursor is in.

selection-add-all-occurrences-in-block (stop_at_blank=True, match_case=None, whole_words=None)

Add an extra selection for all occurrences of the main selection text in blocks that contain one or more selections

selection-add-all-occurrences-in-class (match_case=None, whole_words=None)

Add an extra selection for all occurrences of the main selection text in classes that contain one or more selections

selection-add-all-occurrences-in-def (match_case=None, whole_words=None)

Add an extra selection for all occurrences of the main selection text in functions / methods that contain one or more selections

selection-add-all-occurrences-in-file (match_case=None, whole_words=None)

Add an extra selection for all occurrences of the main selection text in the file

selection-add-next-occurrence (skip_current=False, reverse=False, match_case=None, whole_words=None, wrap=None)

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: Ctrl-Shift-D invokes selection-add-next-occurrence(skip_current=True); Eclipse: Ctrl-Shift-D invokes selection-add-next-occurrence(skip_current=True); Emacs: Ctrl-Alt-> invokes selection-add-next-occurrence(skip_current=True); OS X: Command-Shift-D invokes selection-add-next-occurrence(skip_current=True); MATLAB: Ctrl-Shift-D invokes selection-add-next-occurrence(skip_current=True); Visual Studio: Ctrl-Shift-D invokes selection-add-next-occurrence(skip_current=True); XCode: Command-Shift-D invokes selection-add-next-occurrence(skip_current=True)*

send-keys (keys)

Send one or more keys to the editor. Send a string to enter each key in the string, or a list of strings and/or (mod, key) tuples where mod is a string containing any of case insensitive 'shift', 'ctrl', or 'alt'. Valid examples:

```
"testme"
"TestMe"
["test", ('Alt', 'X'), "m"]
[('ctrl-Shift', 'X'), ('shift','E'),]  *Key Binding: Shift-Space invokes send-keys(keys=" ")*
```

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-@*

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: Ctrl-space; Eclipse: Ctrl-space; Emacs: Alt-/; OS X: Ctrl-space; MATLAB: Ctrl-space; Visual Studio: Ctrl-space; XCode: Ctrl-Space*

show-selection ()

Turn on display of the current text selection

show-selections-popup ()

Show the selections popup; this overrides the preference setting for the current file

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)

Key Bindings: MATLAB: Ctrl-I

start-of-document ()

Move cursor to start of document *Key Bindings: Wing: Ctrl-Home; Brief: Home Home Home; Eclipse: Ctrl-Home; Emacs: Ctrl-X [; OS X: Command-Up; MATLAB: Ctrl-Home; VI/VIM: Ctrl-Home; Visual Studio: Ctrl-Home; XCode: Command-Up*

start-of-document-extend ()

Move cursor to start of document, adjusting the selection range to new position *Key Bindings: Wing: Ctrl-Shift-Home; Brief: Ctrl-Shift-Home; Eclipse: Ctrl-Shift-Home; Emacs: Ctrl-Shift-Home; OS X: Shift-Home; MATLAB: Ctrl-Shift-Home; VI/VIM: Ctrl-Shift-Home; Visual Studio: Ctrl-Shift-Home; XCode: Shift-Home*

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: Ctrl-Shift-L; Eclipse: Ctrl-Shift-L; Emacs: Ctrl-X Ctrl-T invokes swap-lines(previous=True); MATLAB: Ctrl-Shift-L*

tab-key ()

Implement the tab key, the action of which is configurable by preference *Key Binding: Tab*

toggle-selection-add-match-case ()

Toggle the value of the default flag for whether the selection add commands match case or not when looking for additional occurrences

toggle-selection-add-whole-words ()

Toggle the value of the default flag for whether the selection add commands only add whole words when looking for additional occurrences

toggle-selection-add-wrap ()

Toggle the value of the default flag for whether the selection add commands wrap when looking for additional occurrences

undo ()

Undo last action *Key Bindings: Wing: Ctrl-Z; Brief: Ctrl-Z; Eclipse: Ctrl-Z; Emacs: Ctrl-X U; OS X: Command-Z; MATLAB: Alt-Backspace; VI/VIM: u; Visual Studio: Ctrl-Z; XCode: Command-Z*

yank-line ()

Yank contents of kill buffer created with kill-line into the edit buffer *Key Bindings: Emacs: Ctrl-Y*

yank-range (start_line, end_line, register=None)

Copy given range of lines into given register (or currently selected default register if register is None

General Editor Commands

Editor commands that act on the current (most recently active) source editor, whether or not it currently has the keyboard focus.

black-file (timeout=None)

Reformat the current file with Black, if installed in the active Python. The command will time out after the given number of seconds, or if timeout is None the timeout configured with the Editor > Auto-formatting > Reformat Timeout preference.

black-selection (start=None, end=None)

Reformat the current selection, or current line if there is no selection with Black. Reformats the given range if start and end are given.

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. Append '-pep8' to the style to conform to PEP 8 comment format rules. 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: Ctrl-/; Eclipse: Ctrl-/; Emacs: Ctrl-C C; OS X: Command-'; MATLAB: Ctrl-/; Visual Studio: Ctrl-K Ctrl-C; XCode: 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. Append '-pep8' to the style to conform to PEP 8 block comment format rules. 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. Append '-pep8' to the style to conform to PEP 8

comment format rules. If not given, the style configured with the Editor / Block Comment Style preference is used. *Key Bindings: Wing: Ctrl-.; Eclipse: Ctrl-.; Emacs: Ctrl-C #; OS X: Command-;; MATLAB: Ctrl-.; Visual Studio: Ctrl-K Ctrl-T; XCode: 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-code-in-debug-console (code)

Evaluate the given code within the Debug Console tool. When invoking this command directly, only one line can be entered. To enter multiple lines at once, invoke this command with CAPIApplication.ExecuteCommand() in the scripting API.

evaluate-code-in-shell (code, restart=False)

Evaluate the given code within the Python Shell tool, optionally restarting the shell first. When invoking this command directly, only one line can be entered. To enter multiple lines at once, invoke this command with CAPIApplication.ExecuteCommand() in the scripting API.

evaluate-file-in-shell (restart_shell=None)

Run or debug the contents of the editor within the Python Shell *Key Bindings: Wing: Ctrl-Alt-V; Eclipse: Ctrl-Alt-V; MATLAB: Ctrl-Alt-V*

evaluate-sel-in-debug-console (whole_lines=None)

Evaluate the current selection from the editor within the Debug Console 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: Ctrl-Alt-D; Eclipse: Ctrl-Alt-D; MATLAB: Ctrl-Alt-D*

evaluate-sel-in-debug-probe (whole_lines=None)

Not documented

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: Ctrl-Alt-E; Eclipse: Ctrl-Alt-E; Emacs: Ctrl-C |; MATLAB: F9; XCode: Command-R*

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: Ctrl-M; Brief: F8; Eclipse: Ctrl-M; Emacs: Ctrl-X E; OS X: Command-M; MATLAB: Ctrl-M; VI/VIM: @ invokes execute-kbd-macro(register=None); Visual Studio: Ctrl-M; XCode: Command-M

fill-paragraph ()

Attempt to auto-justify the paragraph around the current start of selection *Key Bindings: Wing: Ctrl-J; Eclipse: Ctrl-Shift-F; Emacs: Alt-Q; OS X: Command-J; MATLAB: Ctrl-J; VI/VIM: g q; Visual Studio: Ctrl-K Ctrl-F; XCode: 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: Ctrl-Shift-T; Eclipse: Ctrl-O; Emacs: Ctrl-X G; OS X: Command-Shift-T; MATLAB: Shift-F1; VI/VIM: Ctrl-Shift-T; Visual Studio: Ctrl-Shift-T; XCode: Command-Shift-T*

find-symbol-in-project (fragment=None)

Allow user to visit point of definition of a source symbol in the any file in the project by typing a fragment of the name *Key Bindings: Wing: Ctrl-Shift-P; Eclipse: Ctrl-Shift-T; Emacs: Ctrl-X Ctrl-G; OS X: Command-Shift-P; MATLAB: Ctrl-Shift-F1; VI/VIM: Ctrl-Shift-P; Visual Studio: Ctrl-Shift-P; XCode: Command-Shift-P*

fold-collapse-all ()

Collapse all fold points in the current file *Key Bindings: Wing: Alt-Home; Brief: Alt-Home; Eclipse: Alt-Home; Emacs: Alt-Home; OS X: Command-Ctrl--; MATLAB: Ctrl-=; VI/VIM: z Shift-M; Visual Studio: Alt-Home; XCode: 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: Alt-Page_Up; Brief: Alt-Page_Up; Eclipse: Alt-Page_Up; Emacs: Alt-Page_Up; OS X: Command--; MATLAB: Alt-Page_Up; VI/VIM: Alt-Page_Up; Visual Studio: Alt-Page_Up; XCode: Command--*

fold-collapse-current ()

Collapse the current fold point *Key Bindings: Eclipse: Ctrl--; VI/VIM: z c*

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: Alt-Up; Brief: Alt-Up; Eclipse: Alt-Up; Emacs: Alt-Up; OS X: Command-_; MATLAB: Alt-Up; VI/VIM: Alt-Up; Visual Studio: Alt-Up; XCode: Command-Alt-Left*

fold-expand-all ()

Expand all fold points in the current file *Key Bindings: Wing: Alt-End; Brief: Alt-End; Eclipse: Ctrl-*; Emacs: Alt-End; OS X: Command-Ctrl-*; MATLAB: Ctrl-+; VI/VIM: z Shift-R; Visual Studio: Alt-End; XCode: 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: Alt-Page_Down; Brief: Alt-Page_Down; Eclipse: Alt-Page_Down; Emacs: Alt-Page_Down; OS X: Command-*; MATLAB: Alt-Page_Down; VI/VIM: z Shift-O; Visual Studio: Alt-Page_Down; XCode: Command-**

fold-expand-current ()

Expand the current fold point *Key Bindings: Eclipse: Ctrl-+; VI/VIM: z o*

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: Alt-Down; Brief: Alt-Down; Eclipse: Alt-Down; Emacs: Alt-Down; OS X: Command-+; MATLAB: Alt-Down; VI/VIM: Alt-Down; Visual Studio: Alt-Down; XCode: Command-Alt-Right*

fold-toggle ()

Toggle the current fold point *Key Bindings: Wing: Alt-/; Brief: Alt-/; Eclipse: Ctrl-/; Emacs: Alt-; OS X: Command-/; MATLAB: Ctrl-.; VI/VIM: Alt-/; Visual Studio: Alt-/; XCode: 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>)

Command Reference

Position cursor at start of given line number *Key Bindings: Wing: Ctrl-L; Brief: Alt-G; Eclipse: Ctrl-L; Emacs: Alt-g; OS X: Command-L; MATLAB: Ctrl-G; Visual Studio: Ctrl-G; XCode: 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: Eclipse: Ctrl-| invokes indent-lines(lines=1); MATLAB: Ctrl-]; VI/VIM: >*

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: Ctrl->; Eclipse:*

Ctrl->; Emacs: Ctrl-C >; OS X: Command-]; MATLAB: Ctrl->; VI/VIM: Ctrl-T; Visual Studio: Ctrl->; XCode: 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: g Shift-J invokes join-lines(delim="")*

join-selection (delim=' ')

Join together all lines in given selection (replace newlines with the given delimiter (single space by default) *Key Bindings: VI/VIM: g Shift-J invokes join-selection(delim="")*

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: MATLAB: Ctrl-[; 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: Ctrl-<; Eclipse: Shift-Tab; Emacs: Ctrl-C <; OS X: Command-[; MATLAB: Ctrl-<; VI/VIM: Ctrl-D; Visual Studio: Ctrl-<; XCode: Command-[*

page-setup ()

Show printing page setup dialog

pep8-file (indentation=None, timeout=None)

Reformat the current file to comply with PEP 8 formatting conventions. Indentation is left unchanged unless indentation=True or indentation=None and the Editor > PEP 8 > Reindent All Lines preference is enabled. Indentation within logical lines is always updated. The command will time out after the given number of seconds, or if timeout is None the timeout configured with the Editor > Auto-formatting > Reformat Timeout preference.

pep8-selection (start=None, end=None)

Reformat the current selection, or current line if there is no selection, to comply with PEP 8 formatting conventions. Reformats the given range if start and end are given.

print-view ()

Print active editor document *Key Bindings: Wing: Ctrl-P; Eclipse: Ctrl-P; OS X: Command-P; MATLAB: Ctrl-P; Visual Studio: Ctrl-P; XCode: Command-P*

query-replace (search_string, replace_string)

Initiate incremental mini-search query/replace from the cursor position. *Key Bindings: Wing: Alt-comma; Eclipse: Alt-comma; Emacs: Alt-%; OS X: Ctrl-R; MATLAB: Alt-comma; Visual Studio: Alt-comma; XCode: 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: Ctrl-Alt-Comma; Eclipse: Ctrl-Alt-Comma; Emacs: Ctrl-Alt-%; MATLAB: Ctrl-Alt-Comma; 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*

replace-string (search_string, replace_string)

Replace all occurrences of a string from the cursor position to end of file. *Key Bindings: Wing: Alt-.; Eclipse: Alt-.; Emacs: Alt-@; MATLAB: 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: Ctrl-Alt-.; Eclipse: Ctrl-Alt-.; Emacs: Ctrl-Alt-@; MATLAB: 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: Ctrl-(; Brief: F7; Eclipse: Ctrl-(; Emacs: Ctrl-X (; OS X: Command-Shift-M; MATLAB: Ctrl-(; VI/VIM: q invokes start-kbd-macro(register=None); Visual Studio: Ctrl-(; XCode: Command-Shift-M*

stop-kbd-macro ()

Stop definition of a keyboard macro *Key Bindings: Wing: Ctrl-); Brief: Shift-F7; Eclipse: Ctrl-); Emacs: Ctrl-X); OS X: Command-Shift-M; MATLAB: Ctrl-); VI/VIM: q; Visual Studio: Ctrl-); XCode: Command-Shift-M*

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: Insert; Brief: Alt-I; Eclipse: Ctrl-Shift-Insert; Emacs: Insert; MATLAB: Insert; VI/VIM: 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: Ctrl-?; Eclipse: Ctrl-; Emacs: Ctrl-C U; OS X: Command-"; MATLAB: Ctrl-?; Visual Studio: Ctrl-K Ctrl-U; XCode: 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-cmake ()

Force syntax highlighting for CMake file

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: Ctrl-7 C; Eclipse: Ctrl-7 C; Emacs: Ctrl-X L C; OS X: Command-7 C; MATLAB: Ctrl-7 C; Visual Studio: Ctrl-7 C; XCode: 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-hss ()

Force syntax highlighting for HSS CSS extension language

use-lexer-html ()

Force syntax highlighting for HTML *Key Bindings: Wing: Ctrl-7 H; Eclipse: Ctrl-7 H; Emacs: Ctrl-X L H; OS X: Command-7 H; MATLAB: Ctrl-7 H; Visual Studio: Ctrl-7 H; XCode: 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-less ()

Force syntax highlighting for Less CSS extension language

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: Ctrl-7 M; Eclipse: Ctrl-7 M; Emacs: Ctrl-X L M; OS X: Command-7 M; MATLAB: Ctrl-7 M; Visual Studio: Ctrl-7 M; XCode: 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: Ctrl-7 N; Eclipse: Ctrl-7 N; Emacs: Ctrl-X L N; OS X: Command-7 N; MATLAB: Ctrl-7 N; Visual Studio: Ctrl-7 N; XCode: 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: Ctrl-7 P; Eclipse: Ctrl-7 P; Emacs: Ctrl-X L P; OS X: Command-7 P; MATLAB: Ctrl-7 P; Visual Studio: Ctrl-7 P; XCode: 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 ()

Command Reference

Force syntax highlighting for SCSS formatted SASS

use-lexer-sql ()

Force syntax highlighting for SQL *Key Bindings: Wing: Ctrl-7 S; Eclipse: Ctrl-7 S; Emacs: Ctrl-X L S; OS X: Command-7 S; MATLAB: Ctrl-7 S; Visual Studio: Ctrl-7 S; XCode: 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: Ctrl-7 X; Eclipse: Ctrl-7 X; OS X: Command-7 X; MATLAB: Ctrl-7 X; Visual Studio: Ctrl-7 X; XCode: Command-7 X*

use-lexer-yaml ()

Force syntax highlighting for YAML

yapf-file (timeout=None)

Reformat the current file with YAPF, if installed in the active Python. The command will time out after the given number of seconds, or if timeout is None the timeout configured with the Editor > Auto-formatting > Reformat Timeout preference.

yapf-selection (start=None, end=None)

Reformat the current selection, or current line if there is no selection with YAPF. Reformats the given range if start and end are given.

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 *Key Binding: Ctrl-_*

Shell Or Editor Commands

Commands available when working either in the shell or editor

goto-clicked-symbol-defn (other_split=None)

Goto the definition of the source symbol that was last clicked on. If other_split is true, the definition will be displayed if a split other than the current split; if other_split is false, it will be displayed in the current editor; if other_split is not specified or None, the split to be used is determined by the Split Reuse Policy preference value. *Key Bindings: Wing: Ctrl-left-button-click; Brief: Ctrl-left-button-click; Eclipse: Ctrl-left-button-click; Emacs: Ctrl-left-button-click; OS X: Command-left-button-click; MATLAB: Ctrl-left-button-click; VI/VIM: Ctrl-left-button-click; Visual Studio: Ctrl-left-button-click; XCode: Command-left-button-click*

goto-selected-symbol-defn (other_split=None)

Goto the definition of the selected source symbol. If other_split is true, the definition will be displayed if a split other than the current split; if other_split is false, it will be displayed in the current editor; if other_split is not specified or None, the split to be used is determined by the Split Reuse Policy preference value. *Key Bindings: Wing: F4; Brief: Alt-H; Eclipse: Ctrl-G; Emacs: Alt-.; OS X: F4; MATLAB: F4; VI/VIM: g Shift-D; Visual Studio: F4; XCode: F4*

Source Assistant Commands

Commands for source assistant

hide-assistant-resolution-steps (path='')

Hide the steps used to determine likely types in the source assistant

show-assistant-resolution-steps (path='')

Show the steps used to determine likely types in the source assistant

Bookmark View Commands

Commands available on a specific instance of the bookmark manager tool

bookmarks-export-selected (filename)

Export the selected bookmarks

bookmarks-export-visible (filename)

Export all visible bookmarks

bookmarks-import (filename)

Not documented

bookmarks-remove-all (confirm=1)

Remove all bookmarks

bookmarks-selected-edit ()

Edit the selected bookmark

bookmarks-selected-goto ()

Goto the selected bookmarks

bookmarks-selected-remove (confirm=1)

Remove the selected bookmark

bookmarks-show-docs ()

Show the Wing 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-restore-defaults (delete=False)

Restore the factory default snippets. If delete is True, this will completely remove all snippets first so any changes made to to snippets will be lost. If delete is False, only missing snippet files will be restored.

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 documentation section for the snippet manager

21.4. 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 Bindings: Wing: Left; Brief: Left; Eclipse: Left; Emacs: Ctrl-B; OS X: Ctrl-b; MATLAB: Left; VI/VIM: Ctrl-h; Visual Studio: Left; XCode: Ctrl-b*

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: Shift-BackSpace; Brief: Shift-BackSpace; Eclipse: Shift-BackSpace; Emacs: Ctrl-H; OS X: Ctrl-h; MATLAB: Shift-BackSpace; VI/VIM: Ctrl-H; Visual Studio: Shift-BackSpace; XCode: Ctrl-h*

backward-delete-word ()

Delete word behind the cursor *Key Bindings: Wing: Alt-Delete; Brief: Alt-Delete; Eclipse: Alt-Delete; Emacs: Alt-Delete; OS X: Option-Backspace; MATLAB: Alt-Delete; VI/VIM: Ctrl-W; Visual Studio: Alt-Delete; XCode: Option-Backspace*

backward-word ()

Move backward one word *Key Bindings: Wing: Ctrl-Left; Brief: Ctrl-Left; Eclipse: Ctrl-Left; Emacs: Alt-B; OS X: Ctrl-Left invokes backward-word(delimiters="_`~!@#$%^&\*()+-={}[]\|;:'",.<>/? trn"); MATLAB: Ctrl-Left; VI/VIM: Ctrl-W; Visual Studio: Ctrl-Left; XCode: Ctrl-Left invokes backward-word(delimiters="_`~!@#$%^&*()+-={}[]\|;:'",.<>/? trn")*

backward-word-extend ()

Move backward one word, extending the selection *Key Bindings: Wing: Ctrl-Shift-Left; Brief: Ctrl-Shift-Left; Eclipse: Ctrl-Shift-Left; Emacs: Ctrl-Shift-Left; OS X: Option-Shift-Left; MATLAB: Ctrl-Shift-Left; VI/VIM: Ctrl-Shift-Left; Visual Studio: Ctrl-Shift-Left; XCode: Option-Shift-Left*

beginning-of-line ()

Move to the beginning of the toolbar search entry *Key Bindings: Brief: Shift-Home; Emacs: Ctrl-A; OS X: Ctrl-a; VI/VIM: 0 invokes beginning-of-line(toggle=0); XCode: Ctrl-a*

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; XCode: Command-Shift-Left*

copy ()

Cut selection *Key Bindings: Wing: Ctrl-C; Brief: Ctrl-C; Eclipse: Ctrl-C; Emacs: Alt-W; OS X: Command-C; MATLAB: Ctrl-C; VI/VIM: Ctrl-Insert; Visual Studio: Ctrl-C; XCode: Command-C*

cut ()

Cut selection *Key Bindings: Wing: Ctrl-X; Brief: Ctrl-X; Eclipse: Ctrl-X; Emacs: Ctrl-W; OS X: Command-X; MATLAB: Ctrl-X; VI/VIM: Shift-Delete; Visual Studio: Ctrl-X; XCode: Command-X*

end-of-line ()

Move to the end of the toolbar search entry *Key Bindings: Wing: End; Brief: Shift-End; Eclipse: End; Emacs: Ctrl-E; OS X: Ctrl-e; MATLAB: Ctrl-E; VI/VIM: $; Visual Studio: End; XCode: Ctrl-e*

end-of-line-extend ()

Move to the end of the toolbar search entry, extending the selection *Key Bindings: Wing: Shift-End; Brief: Shift-End; Eclipse: Shift-End; Emacs: Shift-End; OS X: Command-Shift-Right; MATLAB: Shift-End; VI/VIM: Shift-End; Visual Studio: Shift-End; XCode: Command-Shift-Right*

forward-char ()

Move forward one character *Key Bindings: Wing: Right; Brief: Right; Eclipse: Right; Emacs: Ctrl-F; OS X: Ctrl-f; MATLAB: Right; VI/VIM: l invokes forward-char(wrap=0); Visual Studio: Right; XCode: Ctrl-f*

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 Bindings: Wing: Delete; Brief: Delete; Eclipse: Delete; Emacs: Ctrl-D; OS X: Ctrl-d; MATLAB: Delete; VI/VIM: Delete; Visual Studio: Delete; XCode: Ctrl-d*

forward-delete-word ()

Delete word in front of the cursor *Key Bindings: Wing: Ctrl-Delete; Brief: Ctrl-K; Eclipse: Ctrl-Delete; Emacs: Alt-D; OS X: Option-Delete; MATLAB: Ctrl-Delete; VI/VIM: Ctrl-Delete; Visual Studio: Ctrl-Delete; XCode: Option-Delete*

forward-word ()

Move forward one word *Key Bindings: Wing: Ctrl-Right; Brief: Ctrl-Right; Eclipse: Ctrl-Right; Emacs: Alt-F; OS X: Option-Right; MATLAB: Ctrl-Right; VI/VIM: Shift-E invokes forward-word(delimiters=" tnr", gravity="endm1"); Visual Studio: Ctrl-Right; XCode: Option-Right*

forward-word-extend ()

Move forward one word, extending the selection *Key Bindings: Wing: Ctrl-Shift-Right; Brief: Ctrl-Shift-Right; Eclipse: Ctrl-Shift-Right; Emacs: Ctrl-Shift-Right; OS X: Ctrl-Shift-Right invokes forward-word-extend(delimiters="_`~!@#$%^&*()+-={}[]\|;:'",.<>/? trn"); MATLAB: Ctrl-Shift-Right; VI/VIM: Ctrl-Shift-Right; Visual Studio: Ctrl-Shift-Right; XCode: Ctrl-Shift-Right invokes forward-word-extend(delimiters="_`~!@#$%^&*()+-={}[]\|;:'",.<>/? trn")*

paste ()

Paste from clipboard *Key Bindings: Wing: Ctrl-V; Brief: Ctrl-V; Eclipse: Ctrl-V; Emacs: Ctrl-Y; OS X: Command-V; MATLAB: Ctrl-V; VI/VIM: Shift-Insert; Visual Studio: Ctrl-V; XCode: Command-V*

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: Ctrl-Shift-H; Eclipse: Ctrl-Shift-H; Emacs: Ctrl-); OS X: Command-Shift-R; MATLAB: Ctrl-Shift-H; VI/VIM: Ctrl-Shift-G; Visual Studio: Ctrl-Shift-H; XCode: Command-Alt-Shift-F*

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: Ctrl-Shift-F; Eclipse: Ctrl-Shift-U invokes batch-search(look_in="Current File"); Emacs: Ctrl-(; OS X: Command-Shift-F; MATLAB: Ctrl-H; VI/VIM: Ctrl-Shift-F; Visual Studio: Ctrl-Shift-F; XCode: 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: Ctrl-H; Brief: Alt-T; Eclipse: Ctrl-H; Emacs: Ctrl-0; OS X: Command-R; MATLAB: Ctrl-H; Visual Studio: Ctrl-H; XCode: Command-Alt-F*

replace-again ()

Replace current selection with the search manager.

replace-and-search ()

Replace current selection and search again. *Key Bindings: Wing: Ctrl-I; Brief: Shift-F6; Eclipse: Ctrl-I; OS X: Command-Ctrl-R; MATLAB: Ctrl-I; XCode: Command-Ctrl-R*

search ()

Bring up the search manager in search mode. *Key Bindings: Wing: Ctrl-F; Brief: F5; Eclipse: Ctrl-F; Emacs: Ctrl-9; OS X: Command-F; MATLAB: Ctrl-F; VI/VIM: Alt-F3; Visual Studio: Ctrl-F; XCode: Command-F*

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 Bindings: Wing: Ctrl-Shift-G; Brief: Shift-F3; Eclipse: Ctrl-Shift-K; Emacs: Shift-F3; OS X: Command-Shift-G; MATLAB: Ctrl-Shift-G; VI/VIM: Shift-F3; Visual Studio: Ctrl-Shift-G; XCode: Command-Shift-G*

search-forward (search_string='')

Search again using the search manager's current settings in forward direction *Key Bindings: Wing: Ctrl-G; Brief: Shift-F5; Eclipse: Ctrl-K; Emacs: F3; OS X: Command-G; MATLAB: Ctrl-G; VI/VIM: F3; Visual Studio: F3; XCode: Command-G*

search-sel ()

Search forward using current selection

search-sel-backward ()

Search backward using current selection *Key Bindings: Wing: Ctrl-Alt-B; Brief: Alt-F5; Eclipse: Ctrl-Alt-B; Emacs: Ctrl-Alt-B; OS X: Command-Shift-F3; MATLAB: Ctrl-Alt-B; VI/VIM: Ctrl-Shift-F3; Visual Studio: Ctrl-Alt-B; XCode: Command-Shift-F3*

search-sel-forward ()

Search forward using current selection *Key Bindings: Wing: Ctrl-Alt-F; Brief: Ctrl-F3; Eclipse: Ctrl-Alt-F; Emacs: Ctrl-Alt-F; OS X: Command-E; MATLAB: Ctrl-Alt-F; VI/VIM: Ctrl-F3; Visual Studio: Ctrl-Alt-F; XCode: Command-E*

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: Ctrl-C; Brief: Ctrl-C; Eclipse: Ctrl-C; Emacs: Alt-W; OS X: Command-C; MATLAB: Ctrl-C; VI/VIM: Ctrl-Insert; Visual Studio: Ctrl-C; XCode: Command-C*

cut ()

Cut selected text *Key Bindings: Wing: Ctrl-X; Brief: Ctrl-X; Eclipse: Ctrl-X; Emacs: Ctrl-W; OS X: Command-X; MATLAB: Ctrl-X; VI/VIM: Shift-Delete; Visual Studio: Ctrl-X; XCode: Command-X*

forward-tab ()

Place a forward tab at the current cursor position in search or replace string *Key Bindings: Wing: Ctrl-T; Brief: Ctrl-T; Eclipse: Ctrl-T; Emacs: Ctrl-T; OS X: Ctrl-T; MATLAB: Tab; VI/VIM: Ctrl-T; Visual Studio: Ctrl-T; XCode: Ctrl-T*

paste ()

Paste text from clipboard *Key Bindings: Wing: Ctrl-V; Brief: Ctrl-V; Eclipse: Ctrl-V; Emacs: Ctrl-Y; OS X: Command-V; MATLAB: Ctrl-V; VI/VIM: Shift-Insert; Visual Studio: Ctrl-V; XCode: Command-V*

21.5. Refactoring Commands

Refactoring Commands

Not documented

extract-def (new_name=None)

Extract selected lines to a new function or method. The new_name argument is used as the default for the name field if specified. *Key Bindings: Eclipse: Alt-Shift-M*

introduce-variable (pos_range=None, new_name=None)

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. *Key Bindings: Eclipse: Alt-Shift-L*

move-symbol (symbol=None, new_filename=None, new_scope_name=None)

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. *Key Bindings: Eclipse: Alt-Shift-V*

move-symbol-clicked ()

Move last symbol clicked to another module, class, or function.

rename-symbol (fully_scoped=None, new_name=None, transform=None)

Rename currently selected symbol. The new_name argument is used as the default for the name field if specified. Aternatively, the transform argument may be set to camel-upper for UpperCamelCase, camel-lower for lowerCamelCase, under-lower for under_scored_name, or under-upper for UNDER_SCORED_NAME. *Key Bindings: Eclipse: Alt-Shift-R; XCode: Command-Ctrl-E*

rename-symbol-clicked (new_name=None, transform=None)

Rename last symbol clicked. See rename_symbol for details on arguments.

21.6. 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.

clear-test-results ()

Not documented

debug-all-tests ()

Debug all the tests in testing panel. *Key Bindings: Wing: Ctrl-Shift-F6; Brief: Ctrl-Shift-F6; Eclipse: Ctrl-Shift-F6; Emacs: Ctrl-Shift-F6; OS X: Command-Shift-F6; MATLAB: Ctrl-Shift-F6; VI/VIM: Ctrl-Shift-F6; Visual Studio: Ctrl-Shift-F6; XCode: 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: Ctrl-Shift-F7; Brief: Ctrl-Shift-F7; Eclipse: Ctrl-Shift-F7; Emacs: Ctrl-Shift-F7; OS X: Command-Shift-F7; MATLAB: Ctrl-Shift-F7; VI/VIM: Ctrl-Shift-F7; Visual Studio: Ctrl-Shift-F7; XCode: Command-Shift-F7*

debug-failed-tests ()

Re-run all the previously failed tests in the debugger. *Key Bindings: Wing: Ctrl-Alt-F6; Brief: Ctrl-Alt-F6; Eclipse: Ctrl-Alt-F6; Emacs: Ctrl-Alt-F6; OS X: Command-Option-F6; MATLAB: Ctrl-Alt-F6; VI/VIM: Ctrl-Alt-F6; Visual Studio: Ctrl-Alt-F6; XCode: Command-Option-F6*

debug-last-tests ()

Debug the last group of tests that were run. *Key Bindings: Wing: Ctrl-Alt-F7; Brief: Ctrl-Alt-F7; Eclipse: Ctrl-Alt-F7; Emacs: Ctrl-Alt-F7; OS X: Command-Option-F7; MATLAB: Ctrl-Alt-F7; VI/VIM: Ctrl-Alt-F7; Visual Studio: Ctrl-Alt-F7; XCode: 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.

internal-testing-logging-start ()

Start verbose logging of test results

internal-testing-logging-stop ()

Stop verbose logging of test results

load-test-results (filename)

Load all test results from a file.

remove-individually-added-testing-files ()

Remove all files added individually

run-all-tests (debug=False)

Runs all the tests in testing panel. *Key Bindings: Wing: Shift-F6; Brief: Shift-F6; Eclipse: Shift-F6; Emacs: Shift-F6; OS X: Shift-F6; MATLAB: Shift-F6; VI/VIM: Shift-F6; Visual Studio: Shift-F6; XCode: Command-U*

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: Alt-F6; Brief: Alt-F6; Eclipse: Alt-F6; Emacs: Alt-F6; OS X: Option-F6; MATLAB: Alt-F6; VI/VIM: Alt-F6; Visual Studio: Alt-F6; XCode: 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: Alt-F7; Brief: Alt-F7; Eclipse: Alt-F7; Emacs: Alt-F7; OS X: Option-F7; MATLAB: Alt-F7; VI/VIM: Alt-F7; Visual Studio: Alt-F7; XCode: 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

21.7. Version Control Commands

Subversion Commands

Subversion revision control system commands

svn-update (locs=<selected files>)

Update the selected files from the Subversion repository

svn-add ()

Add the files to %(label)s

svn-commit-project ()

Commit all project files

svn-revert ()

Revert selected files

svn-project-status ()

View status for entire project

svn-blame (locs=<selected files>)

Show blame / praise / annotate for selected files.

svn-diff ()

Show differences between files in working directory and last committed version

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-configure ()

Show preferences page for selected VCS

svn-status ()

View status of the selected files in the working directory

svn-remove ()

Remove files

svn-update-project ()

Update files in project

svn-resolved (locs=<selected files>)

Indicate that any conflicts are resolved

svn-checkout ()

Start the initial checkout from svn repository. Repository and working directory must be entered before the checkout.

svn-commit ()

Not documented

Git Commands

git revision control system commands

git-diff ()

Show differences between files in working directory and last committed version

git-fetch-repository (locs=<selected files>)

Pull from repository.

git-remove ()

Remove files

git-list (locs=<selected files>)

Show the status of the given files in the git repository

git-commit-project ()

Commit all project files

git-project-status ()

View status for entire project

git-pull-branch (locs=<selected files>)

Pull branch from other git repository

git-list-branches (locs=<selected files>)

List all branches

git-add ()

Add the files to %(label)s

git-log (locs=<selected files>)

Show the revision log for the selected files in the git repository

git-push-branch (locs=<selected files>)

Push branch to other git repository

git-commit ()

Not documented

git-status ()

View status of the selected files in the working directory

git-switch-branch (locs=<selected files>)

Switch to another branch

git-configure ()

Show preferences page for selected VCS

git-blame (locs=<selected files>)

Show the annotated blame/praise for the selected files in the git repository

Bazaar Commands

Subversion revision control system commands

bzr-commit ()

Not documented

bzr-add ()

Add the files to %(label)s

bzr-commit-project ()

Commit all project files

bzr-remove ()

Remove files

bzr-project-status ()

View status for entire project

bzr-status ()

View status of the selected files in the working directory

bzr-push-entire-branch (locs=<selected files>)

Update the selected files from the bzr repository

bzr-log (locs=<selected files>)

Show the revision log for the selected files in the bzr repository

bzr-configure ()

Show preferences page for selected VCS

bzr-revert ()

Revert selected files

bzr-merge-entire-branch (locs=<selected files>)

Update the selected files from the bzr repository

bzr-annotate ()

Show blame / praise / annotate for selected files

bzr-diff ()

Show differences between files in working directory and last committed version

C V S Commands

CVS revision control system commands

cvs-revert (locs=<selected files>)

Revert the selected files

cvs-log (locs=<selected files>)

Show the revision log for the selected files in the CVS repository

cvs-diff (locs=<selected files>)

Show the differences between working version of given files and the corresponding revision in the CVS repository

cvs-configure ()

Configure the CVS integration

cvs-project-status ()

Run status for entire project.

cvs-update (locs=<selected files>)

Update the selected files from the CVS repository

cvs-update-project ()

Update files in project

cvs-checkout ()

Start the initial checkout from cvs repository. Repository and working directory must be entered before the checkout.

cvs-add (locs=<selected files>)

Add the files to cvs

cvs-commit (locs=<selected files>)

Commit the selected files to the CVS repository

cvs-status (locs=<selected files>)

View the CVS repository status for the selected files

cvs-commit-project ()

Commit files in project

cvs-remove (locs=<selected files>)

Remove the selected files

Mercurial Commands

Mercurial revision control system commands

hg-diff ()

Show differences between files in working directory and last committed version

hg-status ()

View status of the selected files in the working directory

hg-revert ()

Revert selected files

hg-add ()

Add the files to %(label)s

hg-pull-entire-repository (locs=<selected files>)

Pull all changes from remote repository to local repository

hg-commit ()

Not documented

hg-update (locs=<selected files>)

Update working directory from repository

hg-annotate (locs=<selected files>)

Show user and revision for every line in the file(s)

hg-configure ()

Show preferences page for selected VCS

hg-remove ()

Remove files

hg-resolve (locs=<selected files>)

Indicate that any conflicts have been resolved

hg-log (locs=<selected files>)

Show the revision log for the selected files in the hg repository

hg-push-entire-repository (locs=<selected files>)

Update the selected files from the hg repository

hg-merge (locs=<selected files>)

Merge working directory with changes in repository

hg-commit-project ()

Commit all project files

hg-project-status ()

View status for entire project

Perforce Commands

Perforce revision control system commands

perforce-log (locs=<selected files>)

Show the revision log for the selected files in the Perforce repository

perforce-blame (locs=<selected files>)

Show blame / praise / annotate for selected files.

perforce-status (locs=<selected files>)

View the Perforce repository status for the selected files

perforce-commit (locs=<selected files>)

Commit the selected files to the Perforce repository

perforce-remove (locs=<selected files>)

Remove the selected files

perforce-commit-project ()

Commit files in project

perforce-revert (locs=<selected files>)

Revert the selected files

perforce-add (locs=<selected files>)

Add the files to perforce

perforce-sync-project ()

Update files in project

perforce-sync (locs=<selected files>)

Copy the selected files from the Perforce repository

perforce-configure ()

Show preferences page for selected VCS

perforce-edit (locs=<selected files>)

Copy the selected files from the Perforce repository

perforce-project-status ()

Run status for entire project.

perforce-diff (locs=<selected files>)

Show the differences between working version of given files and the corresponding revision in the Perforce repository

perforce-resolved (locs=<selected files>)

Indicate that any conflicts are resolved

perforce-annotate ()

Show blame / praise / annotate for selected files

21.8. 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 Bindings: Wing: F9; Brief: F9; Eclipse: Ctrl-Shift-B; Emacs: Ctrl-X Space; OS X: F9; MATLAB: F9; VI/VIM: F9; Visual Studio: F9; XCode: F9*

break-clear-all ()

Clear all breakpoints *Key Bindings: Wing: Ctrl-F9; Brief: Ctrl-F9; Eclipse: Ctrl-F9; Emacs: Ctrl-F9; OS X: Command-F9; MATLAB: Ctrl-F9; VI/VIM: Ctrl-F9; Visual Studio: Ctrl-F9; XCode: 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: Ctrl-Shift-F9; Brief: Ctrl-Shift-F9; Eclipse: Ctrl-Shift-F9; Emacs: Ctrl-Shift-F9; MATLAB: Ctrl-Shift-F9; VI/VIM: 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: Ctrl-Shift-F9; Brief: Ctrl-Shift-F9; Eclipse: Ctrl-Shift-F9; Emacs: Ctrl-Shift-F9; MATLAB: Ctrl-Shift-F9; VI/VIM: 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 Bindings: Wing: F9; Brief: F9; Eclipse: Ctrl-Shift-B; Emacs: Ctrl-X Space; OS X: F9; MATLAB: F9; VI/VIM: F9; Visual Studio: F9; XCode: Command-*

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 conditional breakpoint at the current mouse click location

break-set-disabled ()

Set a disabled breakpoint on the current line *Key Bindings: Wing: Shift-F9; Brief: Shift-F9; Eclipse: Shift-F9; Emacs: Shift-F9; MATLAB: Shift-F9; VI/VIM: Shift-F9; Visual Studio: Shift-F9*

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) *Key Bindings: XCode: Command-Y*

clear-debugger-security-tokens ()

Clear the stored security tokens for accepting external debug connections

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.

create-remote-host (name='', shared=False)

Create a new remote host configuration and open the remote host attribute dialog.

debug-attach ()

Attach to an already-running debug process

debug-console-clear ()

Clear the Debug Console.

debug-console-evaluate-active-range ()

Evaluate the active range in the Debug Console, if any is set

debug-console-show-active-range ()

Show the active range set in the Debug Console in the editor.

debug-console-toggle-active-range ()

Toggle the active range in the Debug Console: The active range is cleared if already set, or otherwise set using the current editor selection.

debug-continue (show_dialog=None)

Start or continue debugging to next breakpoint or exception (press Alt to continue all paused debug processes) *Key Bindings: Wing: F5; Brief: F5; Eclipse: F8; Emacs: Ctrl-C Ctrl-C; OS X: F5; MATLAB: F5; VI/VIM: F5; Visual Studio: F5; XCode: Command-R*

debug-continue-all ()

Continue all paused debug processes *Key Bindings: Wing: Shift-Alt-F5; Brief: Shift-Alt-F5; Eclipse: Shift-Alt-F5; Emacs: Shift-Alt-F5; MATLAB: Shift-Alt-F5; VI/VIM: Shift-Alt-F5; Visual Studio: Shift-Alt-F5*

debug-detach ()

Detach from the debug process and let it run

debug-detach-all ()

Detach from all debug processes and let them run

debug-file (show_dialog=None)

Start debugging the current file (rather than the main entry point) *Key Bindings: Wing: Shift-F5; Brief: Shift-F5; Eclipse: Shift-F5; Emacs: Shift-F5; OS X: Shift-F5; MATLAB: Shift-F5; VI/VIM: Shift-F5; Visual Studio: Ctrl-F5; XCode: Shift-F5*

debug-hide-value-tips ()

Hide all the debug value tooltips previously shown with debug_show_value_tips() *Key Binding: Release-Shift-Space*

debug-kill ()

Terminate current debug session (press Alt to terminate all debug processes) *Key Bindings: Wing: Ctrl-F5; Brief: Ctrl-F5; Eclipse: Ctrl-F5; Emacs: Ctrl-C Ctrl-K; OS X: Command-.; MATLAB: Shift-F5; VI/VIM: Ctrl-F5; Visual Studio: Shift-F5; XCode: Command-.*

debug-kill-all ()

Terminate all debug processes *Key Bindings: Wing: Ctrl-Alt-F5; Brief: Ctrl-Alt-F5; Eclipse: Ctrl-Alt-F5; Emacs: Ctrl-Alt-F5; MATLAB: Ctrl-Alt-F5; VI/VIM: Ctrl-Alt-F5; Visual Studio: Ctrl-Alt-F5*

debug-move-counter ()

Move program counter to caret

debug-move-counter-clicked ()

Move program counter to click location

debug-named-entry-point (name)

Debug the named entry point

debug-new-process (show_dialog=None)

Start a new debug process running

debug-probe-clear ()

Clear the Debug Console.

debug-probe-evaluate-active-range ()

Evaluate the active range in the Debug Console, if any is set

debug-probe-toggle-active-range ()

Toggle the active range in the Debug Console: The active range is cleared if already set, or otherwise set using the current editor selection.

debug-rerun ()

Re-run the latest debug session that was launched from the IDE

debug-restart ()

Stop and restart debugging (press Alt to restart all debug processes)

debug-restart-all ()

Stop and restart all debug processes that were launched from the IDE

debug-show-environment ()

Show the debug run arguments and environment configuration dialog for the main entry point or current file

debug-show-value-tips (release_toggle=False)

Show tooltips on all visible editors indicating the current value of all visible symbols. The value of release_toggle controls whether this command is available if the tips are already shown; this can be used to prevent execution of fallback commands on a key binding while the tips are already visible, if the key is pressed again or reported in key repeat events while the key is held down. *Key Binding: Shift-Space invokes debug-show-value-tips(release_toggle=True)*

debug-stack-menu-items ()

Not documented

debug-stop ()

Pause debug at current program counter (press Alt to pause all debug processes) *Key Bindings: Wing: Ctrl-Shift-F5; Brief: Ctrl-Shift-F5; Eclipse: Ctrl-Shift-I; Emacs: Ctrl-C Ctrl-S; OS X: Command-Shift-F5; MATLAB: Ctrl-C; VI/VIM: Ctrl-Shift-F5; Visual Studio: Ctrl-Shift-F5; XCode: Command-.*

debug-stop-all ()

Pause all free-running debug processes at the current program counter *Key Bindings: Wing: Ctrl-Shift-Alt-F5; Brief: Ctrl-Shift-Alt-F5; Eclipse: Ctrl-Shift-Alt-F5; Emacs: Ctrl-Shift-Alt-F5; MATLAB: Ctrl-Shift-Alt-F5; VI/VIM: Ctrl-Shift-Alt-F5; Visual Studio: Ctrl-Shift-Alt-F5*

debug-to-clicked (new_process=False)

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-main ()

Execute the main entry point outside of the debugger, or the current Python file if no main entry point is defined

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: Shift-F11; Brief: Shift-F11; Eclipse: Shift-F11; Emacs: Shift-F11; MATLAB: Shift-F11; VI/VIM: Shift-F11; Visual Studio: Shift-F11*

frame-up ()

Move up the current debug stack *Key Binding: F11*

hide-debug-value-detail ()

Hide the debug 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

interrupt-debugger ()

Interupt debugger execution; equivalent to ctrl-c on command line

manage-launch-configs ()

Display the launch config manager

manage-named-entry-points ()

Display the named entry point manager

manage-remote-hosts ()

Display the remote host configuration manager

python-shell-clear (show=False, focus=False, scope='all')

Clear text in the python shell, according to given scope ('all' for whole shell, 'selection' for selection and 'entry' for text entered since the last prompt). Optionally shows the Python Shell if not already visible and/or sets focus into it.

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 (show=False, focus=False, prompt=False)

Restart python shell, optionally showing the Python Shell tool and/or placing keyboard focus on it. Prompts the user first when prompt is True or when prompt is 'pref' and the user has not asked to bypass the prompt.

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 *Key Bindings: XCode: Command-B*

run-to-cursor (new_process=False)

Run to current cursor position *Key Bindings: Wing: Alt-F5; Brief: Alt-F5; Eclipse: Ctrl-F5; Emacs: Alt-F5; MATLAB: Alt-F5; VI/VIM: 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 *Key Bindings: MATLAB: Shift-Return*

shell-ctrl-up ()

Not documented

show-debug-value-as-array ()

Show the selected value as an array

show-debug-value-as-text ()

Show the selected value as text

step-into (show_dialog=None, new_process=False)

Step into current execution point, or start debugging at first line *Key Bindings: Wing: F7; Brief: F7; Eclipse: F5; Emacs: F7; OS X: F7; MATLAB: F11; VI/VIM: F7; Visual Studio: F11; XCode: F7*

step-out ()

Step out of the current function or method *Key Bindings: Wing: F8; Brief: F8; Eclipse: F7; Emacs: F8; OS X: F8; MATLAB: F8; VI/VIM: F8; Visual Studio: Shift-F11; XCode: F8*

step-out-to-frame (frame_idx=None)

Step out of the given frame (0=outermost) in the primary stack. Frame is None to step out to the currently selected stack frame.

step-over ()

Step over current instruction *Key Bindings: Wing: Ctrl-F6; Brief: Ctrl-F6; Eclipse: Ctrl-F6; Emacs: Ctrl-F6; MATLAB: Ctrl-F6; VI/VIM: 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 Bindings: Wing: F6; Brief: F6; Eclipse: F6; Emacs: F6; OS X: F6; MATLAB: F10; VI/VIM: F6; Visual Studio: F10; XCode: 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 Pro 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: Ctrl-C; Brief: Ctrl-C; Eclipse: Ctrl-C; Emacs: Alt-W; OS X: Command-C; MATLAB: Ctrl-C; VI/VIM: Ctrl-Insert; Visual Studio: Ctrl-C; XCode: 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

21.9. Script-provided Add-on Commands

Django Script

A plugin that provides Django-specific functionality when a project looks like it contains Django files.

django-sync-db ()

Run manage.py migrate (or syncdb in Django 1.6 and earlier)

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 and Django together

django-validate ()

Run manage.py check (or validate in Django 1.5 and earlier)

django-start-project (django_admin, parent_directory, site_name, superuser, superuser_email, superuser_password, pyexec=None)

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/migrate (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 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 8000" as the main entry point, 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, turns on child process debugging (for auto-reload), enables template debugging 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-run-tests ()

Run manage.py unit tests in the Testing tool

django-migrate-app (appname)

Run manage.py makemigrations for given app name and display the output in a scratch buffer.

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-show-migrations ()

Run manage.py showmigrations and display the output in a scratch buffer.

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 8000" as the main entry point, 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, turns on child process debugging (for auto-reload), enables template debugging 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, pyexec=None)

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/migrate (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 project for use with the new Django project.

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*

Experimental Script

Experimental commands that may be removed from Wing in the future.

experimental-install-simple-names-callback (mtype, callback)

Install a callback for using a list of simple names for the autocomplete popup list for editors for the given mime type. This an experimental feature and may change in future releases. The callback needs to take 2 arguments: (internal-editor-object, ctx) and should return a list of strings to use for autocompletion. The ctx object will have the following attributes:

lineno -- line number of the line with the cursor cursor_pos -- position of the cursor in the document start_pos -- position of the word's first character in the document

Currently this cannot be used for python documents

An example of using this from a script is:

```
def _callback(ed, ctx):
  return ['one', 'two']
wingapi.gApplication.ExecuteCommand('experimental_install_simple_names_callback',
                                    mtype='text/plain',
                                    callback=_callback)
```

Editor Extensions Script

Editor extensions that also serve as examples for scripting Wing.

set-executable-bit (set_bit=True)

Set the current file's executable bit in its permissions. If set_bit is true (the default), the executable bit is set; if set_bit is false, the executable bit is cleared. This doesn't do anything on windows.

toggle-case ()

Toggle current selection or current word between common name formats: my_symbol_name, MySymbolName, and mySymbolName

under-to-hyphen ()

Change underscores to hyphens (dashes) in current selection or current word

fold-python-methods ()

Fold up all Python methods, expand all classes, and leave other fold points alone *Key Bindings: Wing: Alt-1; Brief: Alt-1; Eclipse: Alt-1; Emacs: Alt-1; OS X: Command-Alt--; MATLAB: Alt-1; VI/VIM: Alt-1; Visual Studio: Alt-1; XCode: Command-Alt--*

word-list-completion (word)

Provide simple word-list driven auto-completion on the current editor

open-url-from-editor ()

Open the url at caret in the current editor

kill-line-with-eol ()

Variant of emacs style kill-line command that always kills the eol characters

start-of-block ()

Not documented

upper-case ()

Change current selection or current word to all upper case *Key Bindings: Eclipse: Ctrl-Shift-Y; MATLAB: Shift-Ctrl-U*

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 ()

Change hyphens (dashes) to underscores in current selection or current word

batch-search-current-directory ()

Initial batch search for the current editor's directory

lower-case ()

Change current selection or current word to all lower case *Key Bindings: Eclipse: Ctrl-Shift-X; MATLAB: Ctrl-U*

cc-checkout ()

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)

vs-tab ()

Modified tab indentation command that acts like tab in Visual Studio.

vi-fold-more ()

Approximation of zr key binding in vim *Key Bindings: VI/VIM: z r*

fold-python-classes ()

Fold up all Python classes but leave other fold points alone *Key Bindings: Wing: Alt-2; Brief: Alt-2; Eclipse: Alt-2; Emacs: Alt-2; OS X: Command-Ctrl-/; MATLAB: Alt-2; VI/VIM: Alt-2; Visual Studio: Alt-2; XCode: Command-Ctrl-/*

vi-fold-less ()

Command Reference

Approximation of zm key binding in vim *Key Bindings: VI/VIM: z m*

indent-new-comment-line ()

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 ()

Open the filename at the caret in current editor *Key Bindings: MATLAB: Ctrl-D*

open-clicked-url-from-editor ()

Open the url being clicked in the current editor

sort-selected ()

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-L invokes toggle-mark-command(style="line")*

delete-selected-lines ()

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 ()

Insert a print statement to print a selected variable name and value, along with the file and line number.

smart-cut ()

Implement a variant of clipboard cut that cuts the whole current line if there is no selection on the editor.

end-of-block ()

Not documented

smart-paste ()

A variant of paste that inserts line just copied with smart-copy above current line.

remove-prompts-and-paste ()

Paste from clipboard after removing any >>> and ... prompts

convert-to-lf-lineends ()

Convert the current editor to use LF style line endings

close-all-readonly ()

Close all readonly files

insert-text (text)

Insert given text at current caret location, replacing any existing selected text

convert-to-cr-lineends ()

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 ()

Change current selection or current word 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 ()

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-/; MATLAB: Ctrl-T*

surround (char)

Surround selected text with (), [], {}, "", '', <>, or ``. Arg char should be the opening character. If there is no selection, the current word is surrounded.

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 ()

Open the filename being clicked in the current editor

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 ()

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: Alt-3; Brief: Alt-3; Eclipse: Alt-3; Emacs: Alt-3; OS X: Command-=; MATLAB: Alt-3; VI/VIM: Alt-3; Visual Studio: Alt-3; XCode: Command-=*

toggle-toolbox-separate ()

Toggle between moving the toolboxes to a separate window and the default single-window mode

Testapi Script

Tests for Wing's scripting API.

test-api (verbose=0)

Test Wing's scripting API

Debugger Extensions Script

Scripts that extend the debugger in various ways.

debug-run-to-completion ()

Run the current debug process to completion. This disables all breakpoints temporarily until the process exits.

set-breaks-from-markers ()

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.

Key Binding Reference

This chapter documents all the default key bindings found in the keyboard personalities provided by Wing, set by the **User Interface > Keyboard > 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.

22.1. Wing Personality

This section documents all the default key bindings for the **Wing** keyboard personality, set by the **User Interface > Keyboard > 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-Comma: **query-replace** - Initiate incremental mini-search query/replace from the cursor position.

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-Left-button-click: **find-points-of-use-clicked** - Find points of use for last symbol clicked.

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: **replace-string** - Replace all occurrences of a string from the cursor position to end of file.

Alt-Return: **new-line** - Place a new line at the current cursor position. Override the auto-indent preference by setting auto_indent to 'never' to avoid indent, 'always' to auto-indent, and 'blank-only' to auto-indent only on blank lines.

Alt-Right: **visit-history-next** - Move forward in history to next visited editor position

Alt-Shift-A: **diff-merge-a-b**

Alt-Shift-B: **diff-merge-b-a**

Alt-Shift-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.

Alt-Shift-F4: **find-points-of-use(search_project_files=False)** - Find points of use for a symbol. The symbol defaults to the active selection. Finds points of use in the file the symbol is located and in project files by default.

Alt-Shift-F5: **debug-continue-all** - Continue all paused debug processes

Alt-Shift-Left: **backward-char-extend-rect** - Move cursor backward one character, adjusting the rectangular selection range to new position

Alt-Shift-N: **diff-next**

Alt-Shift-P: **diff-previous**

Alt-Shift-Right: **forward-char-extend-rect** - Move cursor forward one character, adjusting the rectangular selection range to new position

Alt-Shift-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.

Alt-Slash: **fold-toggle** - Toggle the current fold point

Alt-Up: **fold-collapse-more-current** - Collapse the current fold point one more level

Back-button-click: **visit-history-previous** - Move back in history to previous visited editor position

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-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-console** - Evaluate the current selection from the editor within the Debug Console 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-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. Only bookmarks in the current bookmark category are visited unless a category is passed.

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-F5: **debug-kill-all** - Terminate all debug processes

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. Only bookmarks in the current bookmark category are visited unless a category is passed.

Ctrl-Alt-M: **set-bookmark** - Set a bookmark at current location on the editor. Mark is the project-wide textual name of the bookmark, the category is set to the current bookmark category, and notes are left blank.

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-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. Only bookmarks in the current bookmark category are visited unless a category is passed.

Ctrl-Alt-Shift-F5: **debug-stop-all** - Pause all free-running debug processes at the current program counter

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, the category is set to the current bookmark category, and notes are left blank. When removed, the bookmark is removed without confirmation.

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. Only bookmarks in the current bookmark category are visited unless a category is passed.

Ctrl-Alt-V: **evaluate-file-in-shell** - Run or debug the contents of the editor within the Python Shell

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-Bracketright: **brace-match** - Match brace at current cursor position, selecting all text between the two and hilighting the braces

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: **selection-add-next-occurrence** - 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-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-Equal: **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-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** - Terminate current debug session (press Alt to terminate all debug processes)

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-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-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-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-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-Left-button-click: **goto-clicked-symbol-defn** - Goto the definition of the source symbol that was last clicked on. If other_split is true, the definition will be displayed if a split other than the current split; if other_split is false, it will be displayed in the current editor; if other_split is not specified or None, the split to be used is determined by the Split Reuse Policy preference value.

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-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-extend** - Move cursor forward one page, adjusting the selection range to new position

Ctrl-O: **open-gui** - Open a file from local disk or a remote host, prompting with file selection dialog if necessary. The dialog shown depends on the default starting directory, and may be for local files or remote files.

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: **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-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. Append '-pep8' to the style to conform to PEP 8 comment format rules. 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. If other_split is true, the definition will be displayed if a split other than the current split; if other_split is false, it will be displayed in the current editor; if other_split is not specified or None, the split to be used is determined by the Split Reuse Policy preference value.

Ctrl-Prior: **backward-page-extend** - Move cursor backward one page, adjusting the selection range to new position

Ctrl-Q: **quit** - Quit the application.

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-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-occurrence(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-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-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 debug at current program counter (press Alt to pause all debug processes)

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-F8: **start-select-block** - Turn on auto-select block mode

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-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-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: **find-symbol-in-project** - Allow user to visit point of definition of a source symbol in the any file in the project by typing a fragment of the name

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-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-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-Shift-Z: **redo** - Redo last action

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. Append '-pep8' to the style to conform to PEP 8 comment format rules. If not given, the style configured with the Editor / Block Comment Style preference is used. Each call adds a level of commenting.

Ctrl-Space: **show-autocompleter** - Show the auto-completer for current cursor position

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-Underscore: **zoom-reset** - Action varies according to focus: *Document Viewer Commands*: Reset documentation font size to default; *General Editor Commands*: Reset font zoom factor back to zero

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

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

Escape: **exit-visual-mode** - Exit visual mode and return back to default mode

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. If other_split is true, the definition will be displayed if a split other than the current split; if other_split is false, it will be displayed in the current editor; if other_split is not specified or None, the split to be used is determined by the Split Reuse Policy preference value.

F5: **debug-continue** - Start or continue debugging to next breakpoint or exception (press Alt to continue all paused debug processes)

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

Forward-button-click: **visit-history-next** - Move forward in history to next visited editor position

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).

Insert: **toggle-overtype** - Toggle status of overtyping mode

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

Next: **forward-page-extend** - Move cursor forward one page, adjusting the selection range to new position

Page_down: **forward-page** - Move cursor forward one page

Page_up: **backward-page** - Move cursor backward one page

Prior: **backward-page-extend** - Move cursor backward one page, adjusting the selection range to new position

Release-Shift-Space: **debug-hide-value-tips** - Hide all the debug value tooltips previously shown with debug_show_value_tips()

Return: **new-line** - Place a new line at the current cursor position. Override the auto-indent preference by setting auto_indent to 'never' to avoid indent, 'always' to auto-indent, and 'blank-only' to auto-indent only on blank lines.

Right: **forward-char** - Action varies according to focus: *Active Editor Commands*: Move cursor forward one character ; *Toolbar Search Commands*: Move forward one character

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-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. Finds points of use in the file the symbol is located and in project files by default.

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-set-disabled** - Set a disabled breakpoint on the current line
- **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-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-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-Space: Multiple commands; first available is executed:

- **debug-show-value-tips(release_toggle=True)** - Show tooltips on all visible editors indicating the current value of all visible symbols. The value of release_toggle controls whether this command is available if the tips are already shown; this can be used to prevent execution of fallback commands on a key binding while the tips are already visible, if the key is pressed again or reported in key repeat events while the key is held down.
- **send-keys(keys=" ")** - Send one or more keys to the editor. Send a string to enter each key in the string, or a list of strings and/or (mod, key) tuples where mod is a string containing any of case insensitive 'shift', 'ctrl', or 'alt'. Valid examples:

```
"testme"
"TestMe"
["test", ('Alt', 'X'), "m"]
[('ctrl-Shift', 'X'), ('shift','E'),]
```

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.

22.2. Emacs Personality

This section documents all the default key bindings for the **Emacs** keyboard personality, set by the **User Interface > Keyboard > Personality** preference.

Alt-0: **initiate-repeat-0** - Enter a sequence of digits indicating number of times to repeat the subsequent command or keystroke.

Alt-1: **initiate-repeat-1** - Enter a sequence of digits indicating number of times to repeat the subsequent command or keystroke.

Alt-2: **initiate-repeat-2** - Enter a sequence of digits indicating number of times to repeat the subsequent command or keystroke.

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-At: **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-Backslash: **fold-toggle** - Toggle the current fold point

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-Braceleft: **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-Braceright: **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.

Alt-C: **title-case** - Change current selection or current word 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-Exclam: **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-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-Greater: **end-of-document** - Move cursor to end of document

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-Left-button-click: **find-points-of-use-clicked** - Find points of use for last symbol clicked.

Alt-Less: **start-of-document** - Move cursor to start of document

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-Percent: **query-replace** - Initiate incremental mini-search query/replace from the cursor position.

Alt-Period: **goto-selected-symbol-defn** - Goto the definition of the selected source symbol. If other_split is true, the definition will be displayed if a split other than the current split; if other_split is false, it will be displayed in the current editor; if other_split is not specified or None, the split to be used is determined by the Split Reuse Policy preference value.

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. Override the auto-indent preference by setting auto_indent to 'never' to avoid indent, 'always' to auto-indent, and 'blank-only' to auto-indent only on blank lines.

Alt-Right: **visit-history-next** - Move forward in history to next visited editor position

Alt-Shift-A: **diff-merge-a-b**

Alt-Shift-B: **diff-merge-b-a**

Alt-Shift-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.

Alt-Shift-F4: **find-points-of-use(search_project_files=False)** - Find points of use for a symbol. The symbol defaults to the active selection. Finds points of use in the file the symbol is located and in project files by default.

Alt-Shift-F5: **debug-continue-all** - Continue all paused debug processes

Alt-Shift-Left: **backward-char-extend-rect** - Move cursor backward one character, adjusting the rectangular selection range to new position

Alt-Shift-N: **diff-next**

Alt-Shift-P: **diff-previous**

Alt-Shift-Right: **forward-char-extend-rect** - Move cursor forward one character, adjusting the rectangular selection range to new position

Alt-Shift-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.

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

Back-button-click: **visit-history-previous** - Move back in history to previous visited editor position

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-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-At: **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. Only bookmarks in the current bookmark category are visited unless a category is passed.

Ctrl-Alt-F: **search-sel-forward** - Search forward using current selection

Ctrl-Alt-F5: **debug-kill-all** - Terminate all debug processes

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-Greater: **selection-add-next-occurrence(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-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. Only bookmarks in the current bookmark category are visited unless a category is passed.

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-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. Only bookmarks in the current bookmark category are visited unless a category is passed.

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-Shift-F5: **debug-stop-all** - Pause all free-running debug processes at the current program counter

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. Only bookmarks in the current bookmark category are visited unless a category is passed.

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-At: **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-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. Append '-pep8' to the style to conform to PEP 8 comment format rules. 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** - Start or continue debugging to next breakpoint or exception (press Alt to continue all paused debug processes)

Ctrl-C Ctrl-K: **debug-kill** - Terminate current debug session (press Alt to terminate all debug processes)

Ctrl-C Ctrl-S: **debug-stop** - Pause debug at current program counter (press Alt to pause all debug processes)

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 M: **isearch-sel**

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. Append '-pep8' to the style to conform to PEP 8 comment format rules. If not given, the style configured with the Editor / Block Comment Style preference is used.

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-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-Equal: **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-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** - Terminate current debug session (press Alt to terminate all debug processes)

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-Greater: **selection-add-next-occurrence** - 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-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-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-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-Left-button-click: **goto-clicked-symbol-defn** - Goto the definition of the source symbol that was last clicked on. If other_split is true, the definition will be displayed if a split other than the

current split; if other_split is false, it will be displayed in the current editor; if other_split is not specified or None, the split to be used is determined by the Split Reuse Policy preference value.

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-extend** - Move cursor forward one page, adjusting the selection range to new position

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: **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-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. If other_split is true, the definition will be displayed if a split other than the current split; if other_split is false, it will be displayed in the current editor; if other_split is not specified or None, the split to be used is determined by the Split Reuse Policy preference value.

Ctrl-Prior: **backward-page-extend** - Move cursor backward one page, adjusting the selection range to new 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-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(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 debug at current program counter (press Alt to pause all debug processes)

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-F8: **start-select-block** - Turn on auto-select block mode

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-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-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-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-Underscore: **undo** - Undo last action

435

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.
```

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-G: **find-symbol-in-project** - Allow user to visit point of definition of a source symbol in the any file in the project by typing a fragment of the name

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 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-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, the category is set to the current bookmark category, and notes are left blank.

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, the category is set to the current bookmark category, and notes are left blank. When removed, the bookmark is removed without confirmation.

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-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

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

Escape: **exit-visual-mode** - Exit visual mode and return back to default mode

Escape 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

Key Binding Reference

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. If other_split is true, the definition will be displayed if a split other than the current split; if other_split is false, it will be displayed in the current editor; if other_split is not specified or None, the split to be used is determined by the Split Reuse Policy preference value.

F5: **debug-continue** - Start or continue debugging to next breakpoint or exception (press Alt to continue all paused debug processes)

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

Forward-button-click: **visit-history-next** - Move forward in history to next visited editor position

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

Insert: **toggle-overtype** - Toggle status of overtyping mode

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

Next: **forward-page-extend** - Move cursor forward one page, adjusting the selection range to new position

Page_down: **forward-page** - Move cursor forward one page

Page_up: **backward-page** - Move cursor backward one page

Prior: **backward-page-extend** - Move cursor backward one page, adjusting the selection range to new position

Release-Shift-Space: **debug-hide-value-tips** - Hide all the debug value tooltips previously shown with debug_show_value_tips()

Return: **new-line** - Place a new line at the current cursor position. Override the auto-indent preference by setting auto_indent to 'never' to avoid indent, 'always' to auto-indent, and 'blank-only' to auto-indent only on blank lines.

Right: **forward-char** - Action varies according to focus: *Active Editor Commands*: Move cursor forward one character ; *Toolbar Search Commands*: Move forward one character

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-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. Finds points of use in the file the symbol is located and in project files by default.

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-set-disabled** - Set a disabled breakpoint on the current line
- **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-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-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-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-Space: Multiple commands; first available is executed:

- **debug-show-value-tips(release_toggle=True)** - Show tooltips on all visible editors indicating the current value of all visible symbols. The value of release_toggle controls whether this command is available if the tips are already shown; this can be used to prevent execution of fallback commands on a key binding while the tips are already visible, if the key is pressed again or reported in key repeat events while the key is held down.
- **send-keys(keys=" ")** - Send one or more keys to the editor. Send a string to enter each key in the string, or a list of strings and/or (mod, key) tuples where mod is a string containing any of case insensitive 'shift', 'ctrl', or 'alt'. Valid examples:

```
"testme"
"TestMe"
["test", ('Alt', 'X'), "m"]
[('ctrl-Shift', 'X'), ('shift','E'),]
```

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.

22.3. VI/VIM Personality

This section documents all the default key bindings for the **VI/VIM** keyboard personality, set by the **User Interface > Keyboard > Personality** preference.

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

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

1: **initiate-numeric-modifier(digit=1)** - VI style repeat/numeric modifier for following command

1: **initiate-numeric-modifier(digit=1)** - VI style repeat/numeric modifier for following command

2: **initiate-numeric-modifier(digit=2)** - VI style repeat/numeric modifier for following command

2: **initiate-numeric-modifier(digit=2)** - VI style repeat/numeric modifier for following command

3: **initiate-numeric-modifier(digit=3)** - VI style repeat/numeric modifier for following command

3: **initiate-numeric-modifier(digit=3)** - VI style repeat/numeric modifier for following command

4: **initiate-numeric-modifier(digit=4)** - VI style repeat/numeric modifier for following command

4: **initiate-numeric-modifier(digit=4)** - VI style repeat/numeric modifier for following command

5: **initiate-numeric-modifier(digit=5)** - VI style repeat/numeric modifier for following command

5: **initiate-numeric-modifier(digit=5)** - VI style repeat/numeric modifier for following command

6: **initiate-numeric-modifier(digit=6)** - VI style repeat/numeric modifier for following command

6: **initiate-numeric-modifier(digit=6)** - VI style repeat/numeric modifier for following command

7: **initiate-numeric-modifier(digit=7)** - VI style repeat/numeric modifier for following command

7: **initiate-numeric-modifier(digit=7)** - VI style repeat/numeric modifier for following command

8: **initiate-numeric-modifier(digit=8)** - VI style repeat/numeric modifier for following command

8: **initiate-numeric-modifier(digit=8)** - VI style repeat/numeric modifier for following command

9: **initiate-numeric-modifier(digit=9)** - VI style repeat/numeric modifier for following command

9: **initiate-numeric-modifier(digit=9)** - VI style repeat/numeric modifier for following command

A: **enter-insert-mode(pos="after")** - Enter editor insert mode

A: **select-inner(extend=True)** - Select a text object based on the following key press

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-Left-button-click: **find-points-of-use-clicked** - Find points of use for last symbol clicked.

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. Override the auto-indent preference by setting auto_indent to 'never' to avoid indent, 'always' to auto-indent, and 'blank-only' to auto-indent only on blank lines.

Alt-Right: **visit-history-next** - Move forward in history to next visited editor position

Alt-Shift-A: **diff-merge-a-b**

Alt-Shift-B: **diff-merge-b-a**

Alt-Shift-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.

Alt-Shift-F4: **find-points-of-use(search_project_files=False)** - Find points of use for a symbol. The symbol defaults to the active selection. Finds points of use in the file the symbol is located and in project files by default.

Alt-Shift-F5: **debug-continue-all** - Continue all paused debug processes

Alt-Shift-Left: **backward-char-extend-rect** - Move cursor backward one character, adjusting the rectangular selection range to new position

Alt-Shift-N: **diff-next**

Alt-Shift-P: **diff-previous**

Alt-Shift-Right: **forward-char-extend-rect** - Move cursor forward one character, adjusting the rectangular selection range to new position

Alt-Shift-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.

Alt-Slash: **fold-toggle** - Toggle the current fold point

Alt-Up: **fold-collapse-more-current** - Collapse the current fold point one more level

Ampersand: **repeat-replace** - Repeat the last query replace or range replace operation on the current line. The first match is replaced without confirmation.

Apostrophe: **vi-goto-bookmark** - Goto bookmark using single character name defined by the next pressed key

Asciicircum: **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).

Asciicircum: **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).

Asciitilde: **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

Asterisk: **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.

At: **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.

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

Back-button-click: **visit-history-previous** - Move back in history to previous visited editor position

Backspace: **backward-char** - Action varies according to focus: *Active Editor Commands*: Move cursor backward one character ; *Toolbar Search Commands*: Move backward one character

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

Bar: **goto-column** - Move cursor to given column

Braceleft: **backward-paragraph** - Move cursor backward one paragraph (to next all-whitespace line).

Braceright: **forward-paragraph** - Move cursor forward one paragraph (to next all-whitespace line).

Bracketleft 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.

Bracketright 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.

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)

C: **enter-insert-mode(pos="delete-sel")** - Enter editor insert mode

Colon: **vi-command-by-name** - Execute a VI command by name. This implements ":" commands for the VI/Vim keyboard personality. The following subset of VI/Vim : commands are supported:

```
r[!], e[dit], e!, e#, ene[w], w[rite], up[date], wa[ll], q[uit], q[!], qall, wq,
x[it], xall, wqall, sp[lit], vs[plit], new, on[ly], buffers, files, !, s[ubstitute],
d, delm, reg, marks, n[ext], N, p[revious], rew[ind], last, m[ove], co[py], cl[ose]
(an approximation), and set.
```

The supported directives for 'set' are:

```
ic, ignorecase, noic, noignorecase, ai, autoindent, noai, noautoindent, nu, number,
nonu, nonumber, ro, readonly, noro, noreadonly, sm, showmatch, nosm, and noshowmatch.
```

Colon: **vi-command-by-name** - Execute a VI command by name. This implements ":" commands for the VI/Vim keyboard personality. The following subset of VI/Vim : commands are supported:

```
r[!], e[dit], e!, e#, ene[w], w[rite], up[date], wa[ll], q[uit], q[!], qall, wq,
x[it], xall, wqall, sp[lit], vs[plit], new, on[ly], buffers, files, !, s[ubstitute],
d, delm, reg, marks, n[ext], N, p[revious], rew[ind], last, m[ove], co[py], cl[ose]
(an approximation), and set.
```

The supported directives for 'set' are:

```
ic, ignorecase, noic, noignorecase, ai, autoindent, noai, noautoindent, nu, number,
nonu, nonumber, ro, readonly, noro, noreadonly, sm, showmatch, nosm, and noshowmatch.
```

Comma: **repeat-search-char(opposite=1)** - Repeat the last search_char operation, optionally in the opposite direction.

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-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. Only bookmarks in the current bookmark category are visited unless a category is passed.

Ctrl-Alt-F5: **debug-kill-all** - Terminate all debug processes

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. Only bookmarks in the current bookmark category are visited unless a category is passed.

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. Only bookmarks in the current bookmark category are visited unless a category is passed.

Ctrl-Alt-Shift-F5: **debug-stop-all** - Pause all free-running debug processes at the current program counter

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. Only bookmarks in the current bookmark category are visited unless a category is passed.

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-Asciicircum: **nth-document** - Move to the nth document alphabetically in the list of documents open in the current window

Ctrl-B: **backward-page** - Move cursor backward one page

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-Bracketleft: **enter-browse-mode** - Enter editor browse mode

Ctrl-Bracketleft: **enter-browse-mode** - Enter editor browse mode

Ctrl-Bracketleft: **exit-visual-mode** - Exit visual mode and return back to default mode

Ctrl-C: **enter-browse-mode** - Enter editor browse mode

Ctrl-C: **vi-ctrl-c**

Ctrl-C: **vi-ctrl-c**

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.

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.

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.

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-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.

Ctrl-End: **end-of-document** - Move cursor to end of document

Ctrl-Equal: **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-F: **forward-page** - Move cursor forward one page

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** - Terminate current debug session (press Alt to terminate all debug processes)

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-H: **backward-char** - Action varies according to focus: *Active Editor Commands*: Move cursor backward one character ; *Toolbar Search Commands*: Move backward one character

Ctrl-H: **backward-char** - Action varies according to focus: *Active Editor Commands*: Move cursor backward one character ; *Toolbar Search Commands*: Move backward one character

Ctrl-H: **backward-char** - Action varies according to focus: *Active Editor Commands*: Move cursor backward one character ; *Toolbar Search Commands*: Move backward 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-Home: **start-of-document** - Move cursor to start of document

Ctrl-I: **visit-history-next** - Move forward in history to next visited editor position

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-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-J: **new-line** - Place a new line at the current cursor position. Override the auto-indent preference by setting auto_indent to 'never' to avoid indent, 'always' to auto-indent, and 'blank-only' to auto-indent only on blank lines.

Ctrl-J: **new-line** - Place a new line at the current cursor position. Override the auto-indent preference by setting auto_indent to 'never' to avoid indent, 'always' to auto-indent, and 'blank-only' to auto-indent only on blank lines.

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.

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-button-click: **goto-clicked-symbol-defn** - Goto the definition of the source symbol that was last clicked on. If other_split is true, the definition will be displayed if a split other than the current split; if other_split is false, it will be displayed in the current editor; if other_split is not specified or None, the split to be used is determined by the Split Reuse Policy preference value.

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: **new-line** - Place a new line at the current cursor position. Override the auto-indent preference by setting auto_indent to 'never' to avoid indent, 'always' to auto-indent, and 'blank-only' to auto-indent only on blank lines.

Ctrl-M: **new-line** - Place a new line at the current cursor position. Override the auto-indent preference by setting auto_indent to 'never' to avoid indent, 'always' to auto-indent, and 'blank-only' to auto-indent only on blank lines.

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.

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-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-extend** - Move cursor forward one page, adjusting the selection range to new position

Ctrl-O: **enter-browse-mode(provisional=True)** - Enter editor browse mode

Ctrl-O: **visit-history-previous** - Move back in history to previous visited editor position

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-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-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. If other_split is true, the definition will be displayed if a split other than the current split; if other_split is false, it will be displayed in the current editor; if other_split is not specified or None, the split to be used is determined by the Split Reuse Policy preference value.

Ctrl-Prior: **backward-page-extend** - Move cursor backward one page, adjusting the selection range to new position

Ctrl-Q: **start-select-block** - Turn on auto-select block mode

Ctrl-Q: **start-select-block** - Turn on auto-select block mode

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: **redo** - Redo last action

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 debug at current program counter (press Alt to pause all debug processes)

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-F8: **start-select-block** - Turn on auto-select block mode

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-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-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: **find-symbol-in-project** - Allow user to visit point of definition of a source symbol in the any file in the project by typing a fragment of the name

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-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-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-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-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.

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.

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: **delete-to-start-of-line** - Delete everything between the cursor and start of line

Ctrl-U: **delete-to-start-of-line** - Delete everything between the cursor and start of line

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.

Ctrl-Underscore: **zoom-reset** - Action varies according to focus: *Document Viewer Commands*: Reset documentation font size to default; *General Editor Commands*: Reset font zoom factor back to zero

Ctrl-Up: **select-more** - Select more code on either the current line or larger multi-line blocks.

Ctrl-V: **enter-browse-mode** - Enter editor browse mode

Ctrl-V: **vi-ctrl-v**

Ctrl-V: **vi-ctrl-v**

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

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

Ctrl-W B: **move-editor-focus-last** - Move focus to last editor split

Ctrl-W 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.
```

Ctrl-W Ctrl-Asciicircum: **vi-split-edit-alternate**

Ctrl-W Ctrl-W: **move-editor-focus** - Move focus to next or previous editor split, optionally wrapping when the end is reached.

Ctrl-W Down: **move-editor-focus(wrap=False)** - Move focus to next or previous editor split, optionally wrapping when the end is reached.

Ctrl-W J: **move-editor-focus(wrap=False)** - Move focus to next or previous editor split, optionally wrapping when the end is reached.

Ctrl-W K: **move-editor-focus(dir=-1, wrap=False)** - Move focus to next or previous editor split, optionally wrapping when the end is reached.

Ctrl-W Minus: **shrink-split-vertically** - Decrease height of this split

Ctrl-W N: **split-vertically(new=1)** - Split current view vertically. Create new editor in new view when new==1.

Ctrl-W 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.
```

Ctrl-W P: **move-editor-focus-previous** - Move focus to previous editor split

Ctrl-W Plus: **grow-split-vertically** - Increase height of this split

Ctrl-W 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.
  ```
- **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.

Ctrl-W S: **split-vertically** - Split current view vertically. Create new editor in new view when new==1.

Ctrl-W T: **move-editor-focus-first** - Move focus to first editor split

Ctrl-W Up: **move-editor-focus(dir=-1, wrap=False)** - Move focus to next or previous editor split, optionally wrapping when the end is reached.

Ctrl-W V: **split-horizontally** - Split current view horizontally.

Ctrl-W W: **move-editor-focus(dir=-1)** - Move focus to next or previous editor split, optionally wrapping when the end is reached.

Ctrl-X: **vi-ctrl-x**

Ctrl-X: **vi-ctrl-x**

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.

D: **delete-next-move** - Delete the text covered by the next cursor move command.

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.

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

Dollar: **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

Dollar: **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

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.

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

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

Equal: **indent-to-match-next-move** - Indent lines spanned by next cursor move to match, based on the preceding line

Escape: **clear-move-command** - Clear any pending move command action, as for VI mode

Escape: **enter-browse-mode** - Enter editor browse mode

Escape: **enter-browse-mode** - Enter editor browse mode

Escape: **exit-visual-mode** - Exit visual mode and return back to default mode

Exclam: **filter-next-move** - Filter the lines covered by the next cursor move command through an external command and replace the lines with the result

Exclam: **filter-selection** - Filter the current selection through an external command and replace the lines with the result

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.

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. If other_split is true, the definition will be displayed if a split other than the current split; if other_split is false, it will be displayed in the current editor; if other_split is not specified or None, the split to be used is determined by the Split Reuse Policy preference value.

F5: **debug-continue** - Start or continue debugging to next breakpoint or exception (press Alt to continue all paused debug processes)

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

Forward-button-click: **visit-history-next** - Move forward in history to next visited editor position

G 0: **beginning-of-screen-line** - Move to beginning of current wrapped line

G 0: **beginning-of-screen-line** - Move to beginning of current wrapped line

G Asciicircum: **beginning-of-screen-line-text** - Move to first non-blank character at beginning of current wrapped line

G Asciicircum: **beginning-of-screen-line-text** - Move to first non-blank character at beginning of current wrapped line

G Asciitilde: **case-swap-next-move** - Change case of text spanned by next cursor movement so each letter is the opposite of its current case

G D: **goto-selected-symbol-defn** - Goto the definition of the selected source symbol. If other_split is true, the definition will be displayed if a split other than the current split; if other_split is false, it will be displayed in the current editor; if other_split is not specified or None, the split to be used is determined by the Split Reuse Policy preference value.

G Dollar: **end-of-screen-line** - Move to end of current wrapped line

G Dollar: **end-of-screen-line** - Move to end of current wrapped line

G 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

G 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.

G 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.

G 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.

G M: **middle-of-screen-line** - Move to middle of current wrapped line

G 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.

G Q: **fill-paragraph** - Attempt to auto-justify the paragraph around the current start of selection

G Q Q: **fill-paragraph** - Attempt to auto-justify the paragraph around the current start of selection

G 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.

G Shift-D: **goto-selected-symbol-defn** - Goto the definition of the selected source symbol. If other_split is true, the definition will be displayed if a split other than the current split; if other_split is false, it will be displayed in the current editor; if other_split is not specified or None, the split to be used is determined by the Split Reuse Policy preference value.

G 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

G Shift-I: **enter-insert-mode(pos="sol")** - Enter editor insert mode

G Shift-J: **join-lines(delim="")** - Join together specified number of lines after current line (replace newlines with the given delimiter (single space by default)

G Shift-J: **join-selection(delim="")** - Join together all lines in given selection (replace newlines with the given delimiter (single space by default)

G 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.

G Shift-T: **previous-document** - Move to the previous document alphabetically in the list of documents open in the current window

G Shift-U: **case-upper-next-move** - Change case of text spanned by next cursor movement to upper case

G T: **next-document** - Move to the next document alphabetically in the list of documents open in the current window

G U: **case-lower-next-move** - Change case of text spanned by next cursor movement to lower case

G V: **previous-select** - Turn on auto-select using previous mode and selection

Grave: **vi-goto-bookmark** - Goto bookmark using single character name defined by the next pressed key

Greater: **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.

Greater: **indent-next-move** - Indent lines spanned by next cursor move

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

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).

I: **enter-insert-mode(pos="before")** - Enter editor insert mode

I: **select-inner** - Select a text object based on the following key press

Insert: **enter-insert-mode(pos="before")** - Enter editor insert mode

Insert: **toggle-overtype** - Toggle status of overtyping mode

Iso_left_tab: **backward-tab** - Outdent line at current position

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.

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.

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

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

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

Less: **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.

Less: **outdent-next-move** - Outdent lines spanned by next cursor move

M: **vi-set-bookmark** - Set a bookmark at current location on the editor using the next key press as the name of the bookmark.

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.

N: **isearch-repeat** - Repeat the most recent isearch, using same string and regex/text. Reverse direction when reverse is True.

Next: **forward-page-extend** - Move cursor forward one page, adjusting the selection range to new position

Numbersign: **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.

O: **enter-insert-mode(pos="new-below")** - Enter editor insert mode

O: **exchange-point-and-mark** - When currently marking text, this exchanges the current position and mark ends of the current selection

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.

Page_down: **forward-page** - Move cursor forward one page

Page_up: **backward-page** - Move cursor backward one page

Percent: **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.

Period: **repeat-command** - Repeat the last editor command

Plus: **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.

Prior: **backward-page-extend** - Move cursor backward one page, adjusting the selection range to new position

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

Question: **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.

Quotedbl: **set-register** - Set the register to use for subsequent cut/copy/paste operations

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.

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.

Release-Shift-Space: **debug-hide-value-tips** - Hide all the debug value tooltips previously shown with debug_show_value_tips()

Return: **new-line** - Place a new line at the current cursor position. Override the auto-indent preference by setting auto_indent to 'never' to avoid indent, 'always' to auto-indent, and 'blank-only' to auto-indent only on blank lines.

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.

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

S: **enter-insert-mode(pos="delete-sel")** - Enter editor insert mode

S: **forward-delete-char-insert** - Delete one char in front of the cursor and enter insert mode (when working in modal key bindings)

Semicolon: **repeat-search-char** - Repeat the last search_char operation, optionally in the opposite direction.

Shift-A: **enter-insert-mode(pos="after")** - Enter editor insert mode

Shift-A: **enter-insert-mode(pos="eol")** - Enter editor insert mode

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

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-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)

Shift-D: **delete-to-end-of-line(post_offset=-1)** - Delete everything between the cursor and end of line

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-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

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-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.

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. Finds points of use in the file the symbol is located and in project files by default.

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-set-disabled** - Set a disabled breakpoint on the current line
  • **break-enable** - Enable the breakpoint on the current line
  • **break-disable** - Disable the breakpoint on current line

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

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

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-I: **enter-insert-mode(pos="before")** - Enter editor insert mode

Shift-I: **enter-insert-mode(pos="fnb")** - Enter editor insert mode

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-J: **join-lines** - Join together specified number of lines after current line (replace newlines with the given delimiter (single space by default)

Shift-J: **join-selection** - Join together all lines in given selection (replace newlines with the given delimiter (single space by default)

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

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-M: **cursor-move-to-center** - Move cursor to center of display (without scrolling)

Shift-N: **isearch-repeat(reverse=1)** - Repeat the most recent isearch, using same string and regex/text. Reverse direction when reverse is True.

Shift-O: **enter-insert-mode(pos="new-above")** - Enter editor insert mode

Shift-O: **exchange-point-and-mark** - When currently marking text, this exchanges the current position and mark ends of the current selection

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.

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-R: **enter-insert-mode(pos="delete-lines")** - Enter editor insert mode

Shift-R: **enter-replace-mode** - Enter editor replace mode

Shift-Return: **new-line-before** - Place a new line before the current line

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-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).

Shift-Space: Multiple commands; first available is executed:

- **debug-show-value-tips(release_toggle=True)** - Show tooltips on all visible editors indicating the current value of all visible symbols. The value of release_toggle controls whether this command is available if the tips are already shown; this can be used to prevent execution of fallback commands on a key binding while the tips are already visible, if the key is pressed again or reported in key repeat events while the key is held down.
- **send-keys(keys=" ")** - Send one or more keys to the editor. Send a string to enter each key in the string, or a list of strings and/or (mod, key) tuples where mod is a string containing any of case insensitive 'shift', 'ctrl', or 'alt'. Valid examples:

```
"testme"
"TestMe"
["test", ('Alt', 'X'), "m"]
[('ctrl-Shift', 'X'), ('shift','E'),]
```

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.

Shift-Tab: **backward-tab** - Outdent line at current position

Shift-Up: **backward-page** - Move cursor backward one page

Shift-V: **enter-browse-mode** - Enter editor browse mode

Shift-V: **start-select-line** - Turn on auto-select mode line by line

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

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

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.

Shift-Z 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.

Shift-Z 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.

Slash: **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.

Space: **forward-char** - Action varies according to focus: *Active Editor Commands*: Move cursor forward one character ; *Toolbar Search Commands*: Move forward one character

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.

Tab: **tab-key** - Implement the tab key, the action of which is configurable by preference

Timeout-J J: **enter-browse-mode** - Enter editor browse mode

Timeout-J K: **enter-browse-mode** - Enter editor browse mode

U: **undo** - Undo last action

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).

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.

V: **enter-browse-mode** - Enter editor browse mode

V: **start-select-char** - Turn on auto-select mode character by character

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

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.

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.

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.

Y: **move-to-register-next-move** - Move the text spanned by the next cursor motion to a register

Z B: **cursor-to-bottom** - Scroll so cursor is centered at bottom of display

Z C: **fold-collapse-current** - Collapse the current fold point

Z 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.

Z 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.

Z M: **vi-fold-less** - Approximation of zm key binding in vim

Z Minus: **cursor-to-bottom** - Scroll so cursor is centered at bottom of display

Z O: **fold-expand-current** - Expand the current fold point

Z Period: **center-cursor** - Scroll so cursor is centered on display

Z Plus: **cursor-to-top** - Scroll so cursor is centered at top of display

Z R: **vi-fold-more** - Approximation of zr key binding in vim

Z Return: **cursor-to-top** - Scroll so cursor is centered at top of display

Z 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.

Z 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.

Z Shift-M: **fold-collapse-all** - Collapse all fold points in the current file

Z Shift-O: **fold-expand-all-current** - Expand the current fold point completely

Z Shift-R: **fold-expand-all** - Expand all fold points in the current file

Z T: **cursor-to-top** - Scroll so cursor is centered at top of display

Z Z: **center-cursor** - Scroll so cursor is centered on display

22.4. Visual Studio Personality

This section documents all the default key bindings for the **Visual Studio** keyboard personality, set by the **User Interface > Keyboard > 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-Comma: **query-replace** - Initiate incremental mini-search query/replace from the cursor position.

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: **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-Left-button-click: **find-points-of-use-clicked** - Find points of use for last symbol clicked.

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: **replace-string** - Replace all occurrences of a string from the cursor position to end of file.

Alt-Return: **new-line** - Place a new line at the current cursor position. Override the auto-indent preference by setting auto_indent to 'never' to avoid indent, 'always' to auto-indent, and 'blank-only' to auto-indent only on blank lines.

Alt-Right: **visit-history-next** - Move forward in history to next visited editor position

Alt-Shift-A: **diff-merge-a-b**

Alt-Shift-B: **diff-merge-b-a**

Alt-Shift-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.

Alt-Shift-F4: **find-points-of-use(search_project_files=False)** - Find points of use for a symbol. The symbol defaults to the active selection. Finds points of use in the file the symbol is located and in project files by default.

Alt-Shift-F5: **debug-continue-all** - Continue all paused debug processes

Alt-Shift-Left: **backward-char-extend-rect** - Move cursor backward one character, adjusting the rectangular selection range to new position

Alt-Shift-N: **diff-next**

Alt-Shift-P: **diff-previous**

Alt-Shift-Right: **forward-char-extend-rect** - Move cursor forward one character, adjusting the rectangular selection range to new position

Alt-Shift-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.

Alt-Slash: **fold-toggle** - Toggle the current fold point

Alt-Up: **fold-collapse-more-current** - Collapse the current fold point one more level

Back-button-click: **visit-history-previous** - Move back in history to previous visited editor position

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-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-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. Only bookmarks in the current bookmark category are visited unless a category is passed.

Ctrl-Alt-F: **search-sel-forward** - Search forward using current selection

Ctrl-Alt-F5: **debug-kill-all** - Terminate all debug processes

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. Only bookmarks in the current bookmark category are visited unless a category is passed.

Ctrl-Alt-M: **set-bookmark** - Set a bookmark at current location on the editor. Mark is the project-wide textual name of the bookmark, the category is set to the current bookmark category, and notes are left blank.

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-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. Only bookmarks in the current bookmark category are visited unless a category is passed.

Ctrl-Alt-Shift-F5: **debug-stop-all** - Pause all free-running debug processes at the current program counter

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, the category is set to the current bookmark category, and notes are left blank. When removed, the bookmark is removed without confirmation.

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. Only bookmarks in the current bookmark category are visited unless a category is passed.

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-Bracketright: **brace-match** - Match brace at current cursor position, selecting all text between the two and hilighting the braces

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: **selection-add-next-occurrence** - 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-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-Equal: **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-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-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-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-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-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-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. Append '-pep8' to the style to conform to PEP 8 comment format rules. 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-D: **toolbar-search-focus** - Move focus to toolbar search entry.

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, the category is set to the current bookmark category, and notes are left blank. When removed, the bookmark is removed without confirmation.

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. Only bookmarks in the current bookmark category are visited unless a category is passed.

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. Only bookmarks in the current bookmark category are visited unless a category is passed.

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. Append '-pep8' to the style to conform to PEP 8 comment format rules. 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-Left-button-click: **goto-clicked-symbol-defn** - Goto the definition of the source symbol that was last clicked on. If other_split is true, the definition will be displayed if a split other than the current split; if other_split is false, it will be displayed in the current editor; if other_split is not specified or None, the split to be used is determined by the Split Reuse Policy preference value.

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-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-N: **new-file** - Create a new file

Ctrl-Next: **forward-page-extend** - Move cursor forward one page, adjusting the selection range to new position

Ctrl-O: **open-gui** - Open a file from local disk or a remote host, prompting with file selection dialog if necessary. The dialog shown depends on the default starting directory, and may be for local files or remote files.

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: **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-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. If other_split is true, the definition will be displayed if a split other than the current split; if other_split is false, it will be displayed in the current editor; if other_split is not specified or None, the split to be used is determined by the Split Reuse Policy preference value.

Ctrl-Prior: **backward-page-extend** - Move cursor backward one page, adjusting the selection range to new position

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-occurrence(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-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 debug at current program counter (press Alt to pause all debug processes)

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-F8: **start-select-block** - Turn on auto-select block mode

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-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-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: **find-symbol-in-project** - Allow user to visit point of definition of a source symbol in the any file in the project by typing a fragment of the name

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-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-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-Z: **redo** - Redo last action

Ctrl-Slash: **command-by-name** - Execute given command by name, collecting any args as needed

Ctrl-Space: **show-autocompleter** - Show the auto-completer for current cursor position

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

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

Escape: **exit-visual-mode** - Exit visual mode and return back to default mode

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: **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. If other_split is true, the definition will be displayed if a split other than the current split; if other_split is false, it will be displayed in the current editor; if other_split is not specified or None, the split to be used is determined by the Split Reuse Policy preference value.

F5: **debug-continue** - Start or continue debugging to next breakpoint or exception (press Alt to continue all paused debug processes)

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

Forward-button-click: **visit-history-next** - Move forward in history to next visited editor position

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).

Insert: **toggle-overtype** - Toggle status of overtyping mode

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

Next: **forward-page-extend** - Move cursor forward one page, adjusting the selection range to new position

Page_down: **forward-page** - Move cursor forward one page

Page_up: **backward-page** - Move cursor backward one page

Prior: **backward-page-extend** - Move cursor backward one page, adjusting the selection range to new position

Release-Shift-Space: **debug-hide-value-tips** - Hide all the debug value tooltips previously shown with debug_show_value_tips()

Return: **new-line** - Place a new line at the current cursor position. Override the auto-indent preference by setting auto_indent to 'never' to avoid indent, 'always' to auto-indent, and 'blank-only' to auto-indent only on blank lines.

Right: **forward-char** - Action varies according to focus: *Active Editor Commands*: Move cursor forward one character ; *Toolbar Search Commands*: Move forward one character

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-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: **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. Finds points of use in the file the symbol is located and in project files by default.

Shift-F5: **debug-kill** - Terminate current debug session (press Alt to terminate all debug processes)

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-set-disabled** - Set a disabled breakpoint on the current line
> • **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-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-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-Space: Multiple commands; first available is executed:

- **debug-show-value-tips(release_toggle=True)** - Show tooltips on all visible editors indicating the current value of all visible symbols. The value of release_toggle controls whether this command is available if the tips are already shown; this can be used to prevent execution of fallback commands on a key binding while the tips are already visible, if the key is pressed again or reported in key repeat events while the key is held down.
- **send-keys(keys=" ")** - Send one or more keys to the editor. Send a string to enter each key in the string, or a list of strings and/or (mod, key) tuples where mod is a string containing any of case insensitive 'shift', 'ctrl', or 'alt'. Valid examples:

```
"testme"
"TestMe"
["test", ('Alt', 'X'), "m"]
[('ctrl-Shift', 'X'), ('shift','E'),]
```

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.

22.5. OS X Personality

This section documents all the default key bindings for the **OS X** keyboard personality, set by the **User Interface > Keyboard > Personality** preference.

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: **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-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-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-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-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

Alt-Left-button-click: **find-points-of-use-clicked** - Find points of use for last symbol clicked.

Alt-Page_down: **forward-page** - Move cursor forward one page

Alt-Page_up: **backward-page** - Move cursor backward one page

Alt-Return: **new-line** - Place a new line at the current cursor position. Override the auto-indent preference by setting auto_indent to 'never' to avoid indent, 'always' to auto-indent, and 'blank-only' to auto-indent only on blank lines.

Alt-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

Alt-Shift-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.

Alt-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

Alt-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

Alt-Shift-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.

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.

Back-button-click: **visit-history-previous** - Move back in history to previous visited editor position

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-F6: **debug-failed-tests** - Re-run all the previously failed tests in the debugger.

Command-Alt-F7: **debug-last-tests** - Debug the last group of tests that were run.

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. Append '-pep8' to the style to conform to PEP 8 comment format rules. 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: **set-bookmark** - Set a bookmark at current location on the editor. Mark is the project-wide textual name of the bookmark, the category is set to the current bookmark category, and notes are left blank.

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-D: **selection-add-next-occurrence** - 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-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** - Terminate current debug session (press Alt to terminate all debug processes)

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-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-Left-button-click: **goto-clicked-symbol-defn** - Goto the definition of the source symbol that was last clicked on. If other_split is true, the definition will be displayed if a split other than the current split; if other_split is false, it will be displayed in the current editor; if other_split is not specified or None, the split to be used is determined by the Split Reuse Policy preference value.

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 local disk or a remote host, prompting with file selection dialog if necessary. The dialog shown depends on the default starting directory, and may be for local files or remote files.

Command-P: **print-view** - Print active editor document

Command-Parenright: **brace-match** - Match brace at current cursor position, selecting all text between the two and hilighting the braces

Command-Period: **debug-kill** - Terminate current debug session (press Alt to terminate all debug processes)

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. If other_split is true, the definition will be displayed if a split other than the current split; if other_split is false, it will be displayed in the current editor; if other_split is not specified or None, the split to be used is determined by the Split Reuse Policy preference value.

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. Override the auto-indent preference by setting auto_indent to 'never' to avoid indent, 'always' to auto-indent, and 'blank-only' to auto-indent only on blank lines.

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. Append '-pep8' to the style to conform to PEP 8 comment format rules. If not given, the style configured with the Editor / Block Comment Style preference is used.

Command-Shift-B: **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, the category is set to the current bookmark category, and notes are left blank. When removed, the bookmark is removed without confirmation.

Command-Shift-D: **selection-add-next-occurrence(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-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 debug at current program counter (press Alt to pause all debug processes)

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-F8: **start-select-block** - Turn on auto-select block mode

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-M: Multiple commands; first available is executed:

- **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.
- **stop-kbd-macro** - Stop definition of a keyboard macro

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-P: **find-symbol-in-project** - Allow user to visit point of definition of a source symbol in the any file in the project by typing a fragment of the name

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-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: **redo** - Redo last action

Command-Z: **undo** - Undo last action

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-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-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.

Ctrl-Alt-Left: **backward-char-extend-rect** - Move cursor backward one character, adjusting the rectangular selection range to new position

Ctrl-Alt-Right: **forward-char-extend-rect** - Move cursor forward one character, adjusting the rectangular selection range to new position

Ctrl-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.

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-Comma: **visit-history-previous** - Move back in history to previous visited editor position

Ctrl-Command-Asterisk: **fold-expand-all** - Expand all fold points in the current file

Ctrl-Command-B: **goto-bookmark** - Goto named bookmark

Ctrl-Command-Minus: **fold-collapse-all** - Collapse all fold points in the current file

Ctrl-Command-R: **replace-and-search** - Replace current selection and search again.

Ctrl-Command-Slash: **fold-python-classes** - Fold up all Python classes but leave other fold points alone

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-Down: **forward-page** - Move cursor forward one page

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-Equal: **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-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-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-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-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-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-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-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. Override the auto-indent preference by setting auto_indent to 'never' to avoid indent, 'always' to auto-indent, and 'blank-only' to auto-indent only on blank lines.

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-Space: **show-autocompleter** - Show the auto-completer for current cursor position

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-Underscore: **zoom-reset** - Action varies according to focus: *Document Viewer Commands*: Reset documentation font size to default; *General Editor Commands*: Reset font zoom factor back to zero

Ctrl-Up: **backward-page** - Move cursor backward one page

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.

Escape: **exit-visual-mode** - Exit visual mode and return back to default mode

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. If other_split is true, the definition will be displayed if a split other than the current split; if other_split is false, it will be displayed in the current editor; if other_split is not specified or None, the split to be used is determined by the Split Reuse Policy preference value.

F5: **debug-continue** - Start or continue debugging to next breakpoint or exception (press Alt to continue all paused debug processes)

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

Forward-button-click: **visit-history-next** - Move forward in history to next visited editor position

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

Page_down: **forward-page** - Move cursor forward one page

Page_up: **backward-page** - Move cursor backward one page

Release-Shift-Space: **debug-hide-value-tips** - Hide all the debug value tooltips previously shown with debug_show_value_tips()

Return: **new-line** - Place a new line at the current cursor position. Override the auto-indent preference by setting auto_indent to 'never' to avoid indent, 'always' to auto-indent, and 'blank-only' to auto-indent only on blank lines.

Right: **forward-char** - Action varies according to focus: *Active Editor Commands*: Move cursor forward one character ; *Toolbar Search Commands*: Move forward one character

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-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. Override the auto-indent preference by setting auto_indent to 'never' to avoid indent, 'always' to auto-indent, and 'blank-only' to auto-indent only on blank lines.

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-Space: Multiple commands; first available is executed:

- **debug-show-value-tips(release_toggle=True)** - Show tooltips on all visible editors indicating the current value of all visible symbols. The value of release_toggle controls whether this command is available if the tips are already shown; this can be used to

> prevent execution of fallback commands on a key binding while the tips are already visible, if the key is pressed again or reported in key repeat events while the key is held down.
>
> • **send-keys(keys=" ")** - Send one or more keys to the editor. Send a string to enter each key in the string, or a list of strings and/or (mod, key) tuples where mod is a string containing any of case insensitive 'shift', 'ctrl', or 'alt'. Valid examples:

```
"testme"
"TestMe"
["test", ('Alt', 'X'), "m"]
[('ctrl-Shift', 'X'), ('shift','E'),]
```

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.

22.6. Eclipse Personality

This section documents all the default key bindings for the **Eclipse** keyboard personality, set by the **User Interface > Keyboard > 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-Comma: **query-replace** - Initiate incremental mini-search query/replace from the cursor position.

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: **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-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-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-Left-button-click: **find-points-of-use-clicked** - Find points of use for last symbol clicked.

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: **replace-string** - Replace all occurrences of a string from the cursor position to end of file.

Alt-Return: **new-line** - Place a new line at the current cursor position. Override the auto-indent preference by setting auto_indent to 'never' to avoid indent, 'always' to auto-indent, and 'blank-only' to auto-indent only on blank lines.

Alt-Right: **visit-history-next** - Move forward in history to next visited editor position

Alt-Shift-A: **diff-merge-a-b**

Alt-Shift-B: **diff-merge-b-a**

Alt-Shift-Down: **select-less** - Select less code; undoes the last select-more command

Alt-Shift-F4: **find-points-of-use(search_project_files=False)** - Find points of use for a symbol. The symbol defaults to the active selection. Finds points of use in the file the symbol is located and in project files by default.

Alt-Shift-F5: **debug-continue-all** - Continue all paused debug processes

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-N: **diff-next**

Alt-Shift-O: **show_preferences_gui(prefname="edit.highlight-occurrences")**

Alt-Shift-P: **diff-previous**

Alt-Shift-R: **rename-symbol** - Rename currently selected symbol. The new_name argument is used as the default for the name field if specified. Aternatively, the transform argument may be set to camel-upper for UpperCamelCase, camel-lower for lowerCamelCase, under-lower for under_scored_name, or under-upper for UNDER_SCORED_NAME.

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-console (**) 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-Up: **move-line-up(indent=True)** - Move the current line or lines up one line, optionally indenting to match the new position

Back-button-click: **visit-history-previous** - Move back in history to previous visited editor position

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-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-console** - Evaluate the current selection from the editor within the Debug Console 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-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-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-F5: **debug-kill-all** - Terminate all debug processes

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. Only bookmarks in the current bookmark category are visited unless a category is passed.

Ctrl-Alt-M: **set-bookmark** - Set a bookmark at current location on the editor. Mark is the project-wide textual name of the bookmark, the category is set to the current bookmark category, and notes are left blank.

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-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. Only bookmarks in the current bookmark category are visited unless a category is passed.

Ctrl-Alt-Shift-F5: **debug-stop-all** - Pause all free-running debug processes at the current program counter

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, the category is set to the current bookmark category, and notes are left blank. When removed, the bookmark is removed without confirmation.

Ctrl-Alt-Up: **duplicate-line-above** - Duplicate the current line or lines above the selection.

Ctrl-Alt-V: **evaluate-file-in-shell** - Run or debug the contents of the editor within the Python Shell

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-Backslash: **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-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-Bar: **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.

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-Bracketright: **brace-match** - Match brace at current cursor position, selecting all text between the two and hilighting the braces

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-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: **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-console (**) 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-Enter: **new-line-before** - Place a new line before the current line

Ctrl-Equal: **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-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: **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-F8: **start-select-line** - Turn on auto-select mode line by line

Ctrl-F9: **break-clear-all** - Clear all breakpoints

Ctrl-G: **goto-selected-symbol-defn** - Goto the definition of the selected source symbol. If other_split is true, the definition will be displayed if a split other than the current split; if other_split is false, it will be displayed in the current editor; if other_split is not specified or None, the split to be used is determined by the Split Reuse Policy preference value.

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-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-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-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-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: **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-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-button-click: **goto-clicked-symbol-defn** - Goto the definition of the source symbol that was last clicked on. If other_split is true, the definition will be displayed if a split other than the current split; if other_split is false, it will be displayed in the current editor; if other_split is not specified or None, the split to be used is determined by the Split Reuse Policy preference value.

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-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-Minus: **fold-collapse-current** - Collapse the current fold point

Ctrl-N: **new-file** - Create a new file

Ctrl-Next: **forward-page-extend** - Move cursor forward one page, adjusting the selection range to new position

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-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: **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-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. Append '-pep8' to the style to conform to PEP 8 comment format rules. 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-Pointer_button1: **goto-clicked-symbol-defn** - Goto the definition of the source symbol that was last clicked on. If other_split is true, the definition will be displayed if a split other than the current split; if other_split is false, it will be displayed in the current editor; if other_split is not specified or None, the split to be used is determined by the Split Reuse Policy preference value.

Ctrl-Prior: **backward-page-extend** - Move cursor backward one page, adjusting the selection range to new position

Ctrl-Q: **visit-history-previous** - Move back in history to previous visited editor position

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-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: **run-to-cursor** - Run to current cursor position

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: 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-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-D: **selection-add-next-occurrence(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-Shift-Delete: **delete-lines**

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-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-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. Also closes documentation views, unless include_help is set to False.

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. Also closes documentation views, unless include_help is set to False.

Ctrl-Shift-F5: **debug-stop** - Pause debug at current program counter (press Alt to pause all debug processes)

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-F8: **start-select-block** - Turn on auto-select block mode

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. Finds points of use in the file the symbol is located and in project files by default.

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: **debug-stop** - Pause debug at current program counter (press Alt to pause all debug processes)

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-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_up: **backward-page-extend** - Move cursor backward one page, adjusting the selection range to new position

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-S: **save-all** - Save all unsaved items, prompting for names for any new items that don't have a filename already.

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-console (**) 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-in-project** - Allow user to visit point of definition of a source symbol in the any file in the project 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: **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-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-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. Also closes documentation views, unless include_help is set to False.

Ctrl-Shift-X: **lower-case** - Change current selection or current word to all lower case

Ctrl-Shift-Y: **upper-case** - Change current selection or current word to all upper case

Ctrl-Shift-Z: **redo** - Redo last action

Ctrl-Slash: **fold-toggle** - Toggle the current fold point

Ctrl-Space: **show-autocompleter** - Show the auto-completer for current cursor position

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: **execute-file** - Execute the file at the given location or use the active view if loc is None.

Ctrl-Underscore: **zoom-reset** - Action varies according to focus: *Document Viewer Commands*: Reset documentation font size to default; *General Editor Commands*: Reset font zoom factor back to zero

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

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

Enter: **new-line-after** - Place a new line after the current line

Escape: **exit-visual-mode** - Exit visual mode and return back to default mode

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** - Start or continue debugging to next breakpoint or exception (press Alt to continue all paused debug processes)

F12: **focus-current-editor** - Move focus back to the current editor, out of any tool, if there is an active editor.

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. If other_split is true, the definition will be displayed if a split other than the current split; if other_split is false, it will be displayed in the current editor; if other_split is not specified or None, the split to be used is determined by the Split Reuse Policy preference value.

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-console (**) 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: **step-into** - Step into current execution point, or start debugging at first line

F6: **step-over-statement** - Step over current statement

F7: **step-out** - Step out of the current function or method

F8: **debug-continue** - Start or continue debugging to next breakpoint or exception (press Alt to continue all paused debug processes)

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

Forward-button-click: **visit-history-next** - Move forward in history to next visited editor position

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).

Insert: **toggle-overtype** - Toggle status of overtyping mode

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

Next: **forward-page-extend** - Move cursor forward one page, adjusting the selection range to new position

Page_down: **forward-page** - Move cursor forward one page

Page_up: **backward-page** - Move cursor backward one page

Prior: **backward-page-extend** - Move cursor backward one page, adjusting the selection range to new position

Release-Shift-Space: **debug-hide-value-tips** - Hide all the debug value tooltips previously shown with debug_show_value_tips()

Return: **new-line** - Place a new line at the current cursor position. Override the auto-indent preference by setting auto_indent to 'never' to avoid indent, 'always' to auto-indent, and 'blank-only' to auto-indent only on blank lines.

Right: **forward-char** - Action varies according to focus: *Active Editor Commands*: Move cursor forward one character ; *Toolbar Search Commands*: Move forward one character

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-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. Finds points of use in the file the symbol is located and in project files by default.

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-set-disabled** - Set a disabled breakpoint on the current line
- **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-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-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-Space: Multiple commands; first available is executed:

- **debug-show-value-tips(release_toggle=True)** - Show tooltips on all visible editors indicating the current value of all visible symbols. The value of release_toggle controls whether this command is available if the tips are already shown; this can be used to

prevent execution of fallback commands on a key binding while the tips are already visible, if the key is pressed again or reported in key repeat events while the key is held down.

- **send-keys(keys=" ")** - Send one or more keys to the editor. Send a string to enter each key in the string, or a list of strings and/or (mod, key) tuples where mod is a string containing any of case insensitive 'shift', 'ctrl', or 'alt'. Valid examples:

```
"testme"
"TestMe"
["test", ('Alt', 'X'), "m"]
[('ctrl-Shift', 'X'), ('shift','E'),]
```

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.

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.

22.7. Brief Personality

This section documents all the default key bindings for the **Brief** keyboard personality, set by the **User Interface > Keyboard > 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, the category is set to the current bookmark category, and notes are left blank.

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, the category is set to the current bookmark category, and notes are left blank.

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, the category is set to the current bookmark category, and notes are left blank.

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, the category is set to the current bookmark category, and notes are left blank.

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, the category is set to the current bookmark category, and notes are left blank.

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, the category is set to the current bookmark category, and notes are left blank.

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, the category is set to the current bookmark category, and notes are left blank.

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, the category is set to the current bookmark category, and notes are left blank.

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, the category is set to the current bookmark category, and notes are left blank.

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, the category is set to the current bookmark category, and notes are left blank.

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: **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 local disk or a remote host, prompting with file selection dialog if necessary. The dialog shown depends on the default starting directory, and may be for local files or remote files.

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: **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. If other_split is true, the definition will be displayed if a split other than the current split; if other_split is false, it will be displayed in the current editor; if other_split is not specified or None, the split to be used is determined by the Split Reuse Policy preference value.

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-Left-button-click: **find-points-of-use-clicked** - Find points of use for last symbol clicked.

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. Override the auto-indent preference by setting auto_indent to 'never' to avoid indent, 'always' to auto-indent, and 'blank-only' to auto-indent only on blank lines.

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-Shift-A: **diff-merge-a-b**

Alt-Shift-B: **diff-merge-b-a**

Alt-Shift-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.

Alt-Shift-F4: **find-points-of-use(search_project_files=False)** - Find points of use for a symbol. The symbol defaults to the active selection. Finds points of use in the file the symbol is located and in project files by default.

Alt-Shift-F5: **debug-continue-all** - Continue all paused debug processes

Alt-Shift-Left: **backward-char-extend-rect** - Move cursor backward one character, adjusting the rectangular selection range to new position

Alt-Shift-N: **diff-next**

Alt-Shift-P: **diff-previous**

Alt-Shift-Right: **forward-char-extend-rect** - Move cursor forward one character, adjusting the rectangular selection range to new position

Alt-Shift-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.

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.

Back-button-click: **visit-history-previous** - Move back in history to previous visited editor position

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-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. Only bookmarks in the current bookmark category are visited unless a category is passed.

Ctrl-Alt-F5: **debug-kill-all** - Terminate all debug processes

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. Only bookmarks in the current bookmark category are visited unless a category is passed.

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. Only bookmarks in the current bookmark category are visited unless a category is passed.

Ctrl-Alt-Shift-F5: **debug-stop-all** - Pause all free-running debug processes at the current program counter

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. Only bookmarks in the current bookmark category are visited unless a category is passed.

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: **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-Equal: **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-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** - Terminate current debug session (press Alt to terminate all debug processes)

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-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-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-Left-button-click: **goto-clicked-symbol-defn** - Goto the definition of the source symbol that was last clicked on. If other_split is true, the definition will be displayed if a split other than the current split; if other_split is false, it will be displayed in the current editor; if other_split is not specified or None, the split to be used is determined by the Split Reuse Policy preference value.

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-Next: **forward-page-extend** - Move cursor forward one page, adjusting the selection range to new position

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-Pagedown: **end-of-document** - Move cursor to end of document

Ctrl-Pageup: **beginning-of-document**

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. If other_split is true, the definition will be displayed if a split other than the current split; if other_split is false, it will be displayed in the current editor; if other_split is not specified or None, the split to be used is determined by the Split Reuse Policy preference value.

Ctrl-Prior: **backward-page-extend** - Move cursor backward one page, adjusting the selection range to new 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-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 debug at current program counter (press Alt to pause all debug processes)

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-F8: **start-select-block** - Turn on auto-select block mode

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-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-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-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-Underscore: **zoom-reset** - Action varies according to focus: *Document Viewer Commands*: Reset documentation font size to default; *General Editor Commands*: Reset font zoom factor back to zero

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 End: **end-of-document** - Move cursor to end of document

Escape: **exit-visual-mode** - Exit visual mode and return back to default mode

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: **split-vertically** - Split current view vertically. Create new editor in new view when new==1.

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.
```

F5: **search** - Bring up the search manager in search mode.

F6: **replace** - Bring up the search manager in replace mode.

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.

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.

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

Forward-button-click: **visit-history-next** - Move forward in history to next visited editor position

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 Home Home: **start-of-document** - Move cursor to start of document

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

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

Next: **forward-page-extend** - Move cursor forward one page, adjusting the selection range to new position

Page_down: **forward-page** - Move cursor forward one page

Page_up: **backward-page** - Move cursor backward one page

Prior: **backward-page-extend** - Move cursor backward one page, adjusting the selection range to new position

Release-Shift-Space: **debug-hide-value-tips** - Hide all the debug value tooltips previously shown with debug_show_value_tips()

Return: **new-line** - Place a new line at the current cursor position. Override the auto-indent preference by setting auto_indent to 'never' to avoid indent, 'always' to auto-indent, and 'blank-only' to auto-indent only on blank lines.

Right: **forward-char** - Action varies according to focus: *Active Editor Commands*: Move cursor forward one character ; *Toolbar Search Commands*: Move forward one character

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-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-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. Finds points of use in the file the symbol is located and in project files by default.

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-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-set-disabled** - Set a disabled breakpoint on the current line
- **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-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-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-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-Space: Multiple commands; first available is executed:

- **debug-show-value-tips(release_toggle=True)** - Show tooltips on all visible editors indicating the current value of all visible symbols. The value of release_toggle controls whether this command is available if the tips are already shown; this can be used to prevent execution of fallback commands on a key binding while the tips are already visible, if the key is pressed again or reported in key repeat events while the key is held down.
- **send-keys(keys=" ")** - Send one or more keys to the editor. Send a string to enter each key in the string, or a list of strings and/or (mod, key) tuples where mod is a string containing any of case insensitive 'shift', 'ctrl', or 'alt'. Valid examples:

```
"testme"
"TestMe"
["test", ('Alt', 'X'), "m"]
[('ctrl-Shift', 'X'), ('shift','E'),]
```

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.

License Information

Wing is a commercial product that is based on a number of open source technologies. Although the product source code is available for Wing Pro 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), provide required legal statements for the incorporated open source components, and describe what information Wingware may collect through this product.

23.1. Wing Pro 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 Pro" 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 using this software you acknowledge and agree to be bound by the following terms:

1. EXPIRING LICENSE WARNING

Some Software Product Licenses for this Software Product have an expiration date that causes most of the features of the Software Product to be disabled after the expiration date. WINGWARE BEARS NO LIABILITY FOR ANY DAMAGES RESULTING FROM USE OR ATTEMPTED USE OF THE SOFTWARE PRODUCT AFTER THE EXPIRATION DATE OF AN EXPIRING SOFTWARE PRODUCT LICENSE AND HAS NO DUTY TO PROVIDE ANY SUPPORT AFTER THE EXPIRATION DATE OF AN EXPIRING SOFTWARE PRODUCT LICENSE.

2. GRANT OF NON-EXCLUSIVE LICENSE

Wingware grants you and your affliates the world-wide, 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. This right does not include the right to distribute substantial portions of the

522

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, (d) other individual users who use the Software Product for unpaid personal use only (for example, unpaid hobby, learning, or entertainment), , and (e) individuals and entities that have received from Wingware express written permission to use the Software Product for other purposes.

Non-Commercial Use licenses purchased by companies; organizations other than publicly funded charities; government divisions, agencies, or offices; or any other individual or entity not described in the preceding paragraph 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 listed in the documentation included with 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.

Wingware will keep confidential and private all technical information that you provide to obtain support services.

8. 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 your copies of the Software Product and Software Product Licenses.

9. 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.

10. 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.

11. 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.

12. 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 A WINGWARE SUPPORT SERVICES AGREEMENT, WINGWARE'S ENTIRE LIABILITY REGARDING SUPPORT SERVICES SHALL BE GOVERNED BY THE TERMS OF THAT AGREEMENT.

13. 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.

14. 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.

15. 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.

16. 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.

17. TRADEMARKS

The following are trademarks or registered trademarks of Wingware: Wingware, the feather logo, Wing Python IDE, Wing Pro, Wing Personal, Wing 101, Wing IDE, Wing IDE 101, Wing IDE Personal, Wing IDE Professional, Wing IDE Pro, Wing Debugger, and "The Intelligent Development Environment for Python Programmers"

18. 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.

23.2. Open Source License Information

Wing incorporates the following open source technologies, most of which are under OSI Certified Open Source licenses except as indicated in the footnotes:

- Python -- The Python programming language by Guido van Rossum, PythonLabs, and many contributors -- Python Software Foundation License version 2 [3]
- Qt5 -- Graphical user interface toolkit by many contributors -- LGPL v. 2.1 [1] [6]
- Python Imaging Library -- Library for image manipulation with Python, written by Secret Labs AB and Fredrik Lundh -- MIT License
- Scintilla -- Source code editor component by Neil Hodgson and contributors -- MIT License
- docutils -- reStructuredText markup processing by David Goodger and contributors-- Public Domain [2]
- Sqlite -- A self-contained, serverless, zero-configuration, transactional SQL database engine -- Public domain [5]
- pysqlite -- Python bindings for sqlite by Gerhard Haering -- BSD-like custom license [4]
- pexpect -- 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, Jacques-Etienne Baudoux, Geoffrey Marshall, Francisco Lourenco, Glen Mabey, Karthik Gurusamy, Fernando Perez, Corey Minyard, Jon Cohen, Guillaume Chazarain, Andrew Ryan, Nick Craig-Wood, Andrew Stone, Jorgen Grahn, John Spiegel, Jan Grant, and Shane Kerr. -- ISC License
- ptyprocess -- Process control library by Noah Spurrier -- ISC License

- pycodestyle -- A simple Python style checker by Johann C. Rocholl, Florent Xicluna, and Ian Lee -- MIT License
- autopep8 -- A PEP 8 Python code formatter by Hideo Hattori, Steven Myint, Bill Wendling, and contributors -- MIT License
- Pygments -- A syntax highlighter by Georg Brandl, Armin Ronacher, Tim Hatch, and contributors -- MIT License
- getmac -- A utility for obtaining MAC addresses written by Christopher Goes -- MIT License
- Positronic, Cherry Blossom, and Sun Steel -- Color palettes by Daniel Hill -- MIT License

Notes

[1] The LGPL requires us to redistribute the source code for all libraries linked into Wing. 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.

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 © 1997-2011 by Secret Labs AB
    Copyright © 1995-2011 by Fredrik Lundh

By obtaining, using, and/or copying this software and/or its associated documentation, you agree
that you have read, understood, and will comply with the following terms and conditions:

Permission to use, copy, modify, and distribute this software and its associated documentation for
any purpose and without fee is hereby granted, provided that the above copyright notice appears in
all copies, and that both that copyright notice and this permission notice appear in supporting
documentation, and that the name of Secret Labs AB or the author not be used in advertising or
publicity pertaining to distribution of the software without specific, written prior permission.

SECRET LABS AB AND THE AUTHOR DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING ALL
IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL SECRET LABS AB OR THE AUTHOR
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.
```

pycodestyle

We are required by the license terms for pycodestyle to include the following copyright notice in this documentation:

```
Copyright © 2006-2009 Johann C. Rocholl <johann@rocholl.net>
Copyright © 2009-2014 Florent Xicluna <florent.xicluna@gmail.com>
Copyright © 2014-2018 Ian Lee <IanLee1521@gmail.com>

Licensed under the terms of the Expat License

Permission is hereby granted, free of charge, to any person
obtaining a copy of this software and associated documentation files
(the "Software"), to deal in the Software without restriction,
including without limitation the rights to use, copy, modify, merge,
publish, distribute, sublicense, and/or sell copies of the Software,
and to permit persons to whom the Software is furnished to do so,
subject to the following conditions:
```

autopep8

We are required by the license terms for autopep8 to include the following copyright notice in this documentation:

Pygments

We are required by the license terms for Pygments to include the following copyright notice in this documentation:

```
met:

* Redistributions of source code must retain the above copyright
  notice, this list of conditions and the following disclaimer.

* Redistributions in binary form must reproduce the above copyright
  notice, this list of conditions and the following disclaimer in the
  documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
```

pexpect

We are required by the license terms for pexpect to include the following copyright notice in this documentation:

```
Copyright (c) 2013-2014, Pexpect development team
Copyright (c) 2012, Noah Spurrier <noah@noah.org>

Permission to use, copy, modify, and/or distribute this software for any
purpose with or without fee is hereby granted, provided that the above
copyright notice and this permission notice appear in all copies.

THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES
WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF
MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR
ANY SPECIAL, DIRECT, 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.
```

ptyprocess

We are required by the license terms for ptyprocess to include the following copyright notice in this documentation:

```
Copyright (c) 2013-2014, Pexpect development team
Copyright (c) 2012, Noah Spurrier <noah@noah.org>

PERMISSION TO USE, COPY, MODIFY, AND/OR DISTRIBUTE THIS SOFTWARE FOR ANY PURPOSE
WITH OR WITHOUT FEE IS HEREBY GRANTED, PROVIDED THAT THE ABOVE COPYRIGHT NOTICE
AND THIS PERMISSION NOTICE APPEAR IN ALL COPIES. THE SOFTWARE IS PROVIDED
```

getmac

We are required by the license terms for getmac to include the following copyright notice in this documentation:

Positronic, Cherry Blossom, and Sun Steel Display Palettes

We are required by the license terms for these colors palettes to include the following copyright notice in this documentation:

```
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.
```

23.3. Privacy Policy

Wingware collects as little information about its customers as is reasonably necessary to conduct business with them, and never rents or sells information about its customers to a third party. However, customer identity and personal information may be used by Wingware to conduct its own demographic research for marketing purposes, to comply with regulatory or legal requirements, or in confidential applications for services such as insurance.

This product may submit certain information to Wingware, using **https** encrypted communication, as follows:

License Activation

In products that require a license, such as Wing Pro, a license activation process takes place before you can use the software. This is true both for trial licenses and when activating a purchased license. The information passed to our servers includes:

1. License number
2. For trial licenses only, an SHA hash of machine identity metrics that include hardware serial number, ethernet number, and file system IDs
3. The request code, which is an SHA hash of the license number, date, and the same machine identity metrics listed above
4. Your IP address
5. The optional user information that you entered into the license activation dialog for the purpose of license recovery

License activation is a requirement and cannot be disabled. However, it can be done manually via https://wingware.com/activate if you don't want Wing to connect directly to wingware.com.

Update Check

Wing periodically checks for updates by contacting **wingware.com**. This check will send the following information:

1. License number
2. Current version and patch level
3. Your IP address

Update checks can be disabled with the **User Interface > Other > Auto-check for Product Updates** preference and by not using **Check for Updates** in the **Help** menu.

Bug Reports and Feedback

License Information

Wing provides a mechanism for submitting bug reports and feedback, which may send the following information to our servers:

1. Any text entered into the bug report and feedback dialogs
2. License number
3. Your IP address
4. Basic host and installation data including product version and patch level, installation location, settings directory, cache directory, OS type and version, CPU type, memory size, local IP addresses, currently open project, and active Python installation location and version
5. In bug reports, you may optionally include a log of recent IDE activity, the filename of which is given in the bug submission dialog

To avoid submitting this information to Wingware, simply refrain from submitting any bug reports or feedback from the **Help** menu.

Usage Statistics

Wing periodically submits usage statistics to help us understand which features are most used and to help provide support. The data sent consists of:

1. Product type, version, and patch level
2. License number
3. Your IP address

3. Usage statistics consisting of (name, value) pairs where name is a code identifying an IDE feature, such as 'debug.start', 'testing.run', and 'minutes-used', and value is an integer count

To avoid submitting usage statistics, disable the **User Interface > Other > Submit Usage Stats** preference.

Special Offers

Wing periodically checks for special offers posted by Wingware and presents these to the user. This is done as part of the product update check and sends no additional information to **wingware.com**.

To turn off display of special offers without disabling update checks, uncheck the **User Interface > Other > Show Discount Offers** preference.