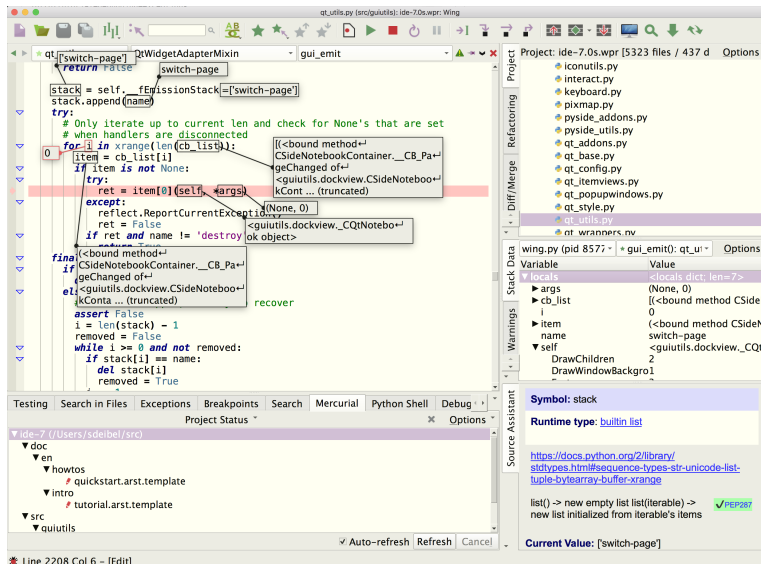




## Wing 101 Reference Manual



This manual documents the entire feature set of Wing 101, which is a free Python IDE designed for teaching entry level programming courses with Python.

It covers installation, customization, editing, searching, using the integrated Python shell, debugging, source code analysis, and trouble-shooting installation and usage problems.

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](#).

---

*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" are trademarks or registered trademarks of Wingware in the United States and other countries.*

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

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

---

Copyright (c) 1999-2020 by Wingware. All rights reserved.

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

# Contents

<b>Wing 101 Reference Manual</b>	<b>1</b>
Introduction	1
1.1. Product Levels	1
1.2. Supported Platforms	1
Windows	1
Mac	1
Linux	2
1.3. Supported Python versions	2
1.4. Prerequisites for Installation	2
1.5. Installing Wing	2
1.6. Running Wing	3
1.7. Settings Directory	3
1.8. Upgrading	4
Upgrading Without an Internet Connection	5
Upgrading to a New Major Release	5
Upgrading Your License	5
1.8.1. Fixing a Failed Upgrade	5
1.9. Installation Details and Options	5
1.9.1. Linux Installation Notes	6
1.9.2. Remote Display on Linux	7
1.10. Backing Up and Sharing Settings	8
1.11. Removing Wing	8
Customization	10
2.1. High Level Configuration Options	10
2.2. User Interface Options	10
2.2.1. Display Style and Colors	10
2.2.2. User Interface Layout	11
2.2.3. Text Font and Size	12
2.3. Keyboard Personalities	12

2.4. Preferences	13
Source Code Editor	14
3.1. Opening, Creating, and Closing Files	14
3.2. File Status and Read-Only Files	14
3.3. Transient, Sticky, and Locked Editors	14
3.4. Editor Context Menu	15
3.5. Navigating Source	15
3.6. Syntax Coloring	15
3.7. Selecting Text	15
3.8. Copy/Paste	17
3.9. Auto-completion	17
3.9.1. Auto-completion Icons	18
3.9.2. How Auto-completion Works	19
3.10. Auto-Reformatting	19
3.10.1. PEP 8 Reformatting Options	20
3.10.2. Black Formatting Options	21
3.10.3. YAPF Formatting Options	21
3.11. Indentation	21
3.11.1. How Indent Style is Determined	21
3.11.2. Auto-Indent	21
3.11.3. The Tab Key	22
3.11.4. Adjusting Indentation	23
3.12. Other Editor Features	23
Search and Replace	25
4.1. Toolbar Quick Search	25
4.2. Search Tool	25
Search Type	25
Search Options	26
Special Characters	26
4.3. Wildcard Search Syntax	26
Integrated Python Shell	27
5.1. Python Shell Environment	28

5.2. Active Ranges in the Python Shell	28
5.3. Debugging Code in the Python Shell	28
5.4. Python Shell Options	29
Debugger	30
6.1. Setting Breakpoints	30
6.2. Starting Debug	30
6.3. Debugger Status	30
6.4. Flow Control	31
6.5. Viewing the Stack	31
6.6. Viewing Debug Data	32
6.6.1. Stack Data Tool	32
6.6.1.1. Array, Data Frame, and Textual Data Views	33
6.6.1.2. Stack Data Options Menu	34
6.6.1.3. Stack Data Context Menu	35
6.6.1.4. Advanced Data Display	35
6.6.2. Viewing Data on the Editor	36
6.6.3. Problems Handling Values	36
Managing Value Errors	37
6.7. Debug Process I/O	38
Options	38
6.8. Debugging Multi-threaded Code	38
Source Code Analysis	40
7.1. How Analysis Works	40
7.2. Helping Wing Analyze Code	40
7.2.1. Setting the Correct Python Environment	40
7.2.2. Using Live Runtime State	41
7.2.3. Adding Type Hints	41
7.2.4. Defining Interface Files	42
7.2.5. Helping Wing Analyze Cython Code	43
7.3. Analysis Disk Cache	43
Trouble-shooting Guide	45
8.1. Trouble-shooting Failure to Start	45

8.2. Speeding up Wing	45
8.3. Trouble-shooting Other Known Problems	46
License Information	48
9.1. Wing 101 Software License	48
9.2. Open Source License Information	52
9.3. Privacy Policy	57

## Introduction

This chapter describes how to install and start using Wing 101. See also the [Quick Start Guide](#) and [Tutorial](#).

### **1.1. Product Levels**

This manual is for the Wing 101 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. Supported Platforms**

Wing 7 is available for Microsoft Windows, Linux, and Mac OS X.

#### **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.

### **1.3. Supported Python versions**

Wing 7 supports versions 2.6 to 2.7 and 3.3 to 3.8 of Python from [python.org](https://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](#)).

### **1.4. 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)

### **1.5. 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 101 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.



## Introduction

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/wing-101-7** or at the selected location when installing from the tar file. Wing will also create a **Settings Directory** in **~/.wing101-7**, which is used to store preferences and other settings.

Wing 101 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.

## 1.6. 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 **wing-101-7.2** (which is on the **PATH** by default for RPM and Debian installs) or execute **wing-101** located inside the Wing installation directory. If you installed on Linux from the [Snapcraft Store](#), then the executable is instead named **wing-101-7**.

## 1.7. 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 101 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 101 7
```

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

```
~/ .wing101-7
```

### 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 **AppDataLocal** area. On Linux, it is **~/.cache/wing101-7** and on OS X, it can be found with the symbolic link **~/.wing101-7/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.

## 1.8. 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](#).

#### ***1.8.1. 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](mailto:support@wingware.com) or submit a bug report from Wing's **Help** menu so we can try to fix the problem for you.

## ***1.9. Installation Details and Options***

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

### 1.9.1. 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/wing-101-7**. 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 wing-101-7-7.2.9.0.x86\_64.rpm** as root or use your favorite RPM administration tool. Most files for Wing are placed under the **/usr/lib/wing-101-7** directory and the **wing-101-7.2** 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 wing-101-7\_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/wing-101-7** directory and the **wing-101-7.2** 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/wing-101-7**. Unpacking this archive with **tar -zxvf wing-101-7.2.9.0-linux-x64.tar.gz** will create a **wing-101-7.2.9.0-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 **wing-101-7.2**. These locations default to **/usr/local/lib/wing-101** 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 wing-101-7 --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.

#### 1.9.2. 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.

## 1.10. 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.

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:

- **preferences** -- Wing's **preferences**, as configured in the **Preferences dialog**.

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.

## 1.11. Removing Wing

### Windows

## Introduction

On Windows, use the Add/Remove Programs control panel, select **Wing 101 7** and remove it.

### Linux/Unix

To remove an RPM installation on Linux, type **rpm -e wing-101-7**.

To remove an Debian package installation on Linux, type **dpkg -r wing-101-7**.

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.

## 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.

#### 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.

#### Editor

**Configure Auto-Completion** can be used to control details of how the editor's auto-completer works. See the [Auto-completion](#) 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.

### 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.

### 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. User Interface Layout

The layout and behavior of the toolboxes, toolbar, and editor area are configurable. This configuration is stored and remembered between sessions.

#### 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 reshown 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.

#### 2.2.3. 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.

### 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**.

## **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 **Wing101** menu on OS X. Preferences are organized by category. Documentation for each preference is displayed when the mouse is hovered over it.

## 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](#).

### 3.1. Opening, Creating, and Closing Files

Files can be opened into the editor from the **File** menu or toolbar, or if already open by selecting them from the **Window** menu or the tabs at the top of the editor.

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.

New files can be created from the **File** menu and from the toolbar.

Open files are closed with close icon in the top right of the editor area.

### 3.2. 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.

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.

### 3.3. 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 debugging, or when navigating to a symbol's point of definition.

-  **Sticky Mode** -- Files opened from the **File** menu (including **Open from Keyboard**) or from the keyboard file selector will be opened in sticky mode, and are kept open until they are explicitly closed, even if they are not edited.

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.

### 3.4. 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. This includes items for copy/pasting, navigating code, evaluating code in the **Python Shell**, commenting out code, adjusting indentation, and accessing **File Properties**.

### 3.5. 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.

#### 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.

#### 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.

### 3.6. 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.

All the available document types for syntax coloring can be seen with **Current File Properties** in the **Source** menu, under the **File Attributes** tab. If you have a file that is not being recognized automatically, you can use the **File Type** menu found there to alter the way the file is being displayed.

### 3.7. 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 selection 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.

### 3.8. 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 indentation style, size, and position when pasting lines of text into the editor. See [Auto-Indent](#) for details.

### 3.9. Auto-completion

Wing provides context-appropriate code completion in the editor and , [Python Shell](#). Using the auto-completer decreases the amount of typing needed to write code, and reduces the incidence of typos in symbol names. In Wing 101, this feature is disabled by default.

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. If **:'** is used as a completion key, the auto-completer will reappear immediately with the attributes of the completed symbol.













#### 3.9.1. 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
-  A class
-  A Python package
-  A method
-  A function
-  A dictionary
-  A tuple
-  A list
-  A string
-  An integer
-  A float
-  An exception





 A Python stack frame

 An object instance of some other type

### Symbol Type Annotation

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

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

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

### 3.9.2. 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.

## 3.10. 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 **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 **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.

#### 3.10.1. 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.
- **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.

### 3.10.2. 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.

### 3.10.3. 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.

## 3.11. Indentation

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

### 3.11.1. 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, space-only indents are inserted matching the **Editor > Indentation > Default Indent Size** preference.

### 3.11.2. 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**).

### 3.11.3. 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.

#### **3.11.4. 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.

### **3.12. 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.

#### **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.

### 4.1. 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.

### 4.2. 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.

## Search and Replace

**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.

## **4.3. 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.



## 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. In Wing 101, the auto-completer is disabled by default but can be enabled with the **Editor > Auto-completion > Auto-show Completer** preference.

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.

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.

### 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.

### 5.1. 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, is configured using **Configure Python** in the **Edit** 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 ;

### 5.2. 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.

### 5.3. 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.

### 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.

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](mailto:support@wingware.com).

## 5.4. 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 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**.

**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.

## Debugger

Wing's debugger can be used to locate and fix bugs in Python code, and as a way to step through and better understand how the code works.

### 6.1. 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. 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).

### 6.2. Starting Debug

The following items in the **Debug** menu, or their key bindings, can be used to start debugging:

- **Start / Continue** runs the file from 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.

Other ways to start debug include:

- 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.

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.

### 6.3. 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:

- **Gray** -- There is no debug process.
- **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.

## 6.4. 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** 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 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**.

**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.

### 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.

## 6.5. 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.

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.

## 6.6. Viewing Debug Data

Wing allows you to inspect locals and globals using the **Stack Data** tool. This area displays values for the currently selected stack frame.

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.

### 6.6.1. 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 be 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.

For details, see [Problems Handling Values](#).

#### 6.6.1.1. 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.

#### 6.6.1.2. 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.

#### 6.6.1.3. 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.

**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.

#### 6.6.1.4. 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 change 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.

### 6.6.2. Viewing Data on 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.

### 6.6.3. 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.

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

## Debugger

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.

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.

### 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](#).

## 6.7. Debug Process I/O

Debug process output from `print()` or any writes to `stdout` or `stderr` appears in the **Debug I/O** tool. This is also where you enter keyboard input, if your debug program requests any with `input()` or by reading from `stdin`.

### 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.

**Show Debug I/O Documentation** displays this documentation page.

## 6.8. 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()`

## Debugger

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.

## 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**.

### 7.1. How Analysis Works

To analyze your source code, Wing uses the **Python Executable** and **Python Path** that you have specified using **Configure Python** in the **Edit** menu. This environment defines which modules are found by **import** statements and alters some aspects of type inference, according to Python version.

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.

### 7.2. 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.

#### 7.2.1. 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](#).

### 7.2.2. 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.

Working in the [Python Shell](#) also provides access to runtime type analysis.

### 7.2.3. 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.

### 7.2.4. 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 some cases, as for Python bindings for GUI and other toolkits, **\*.pyi** or **\*.pi** 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.

### 7.2.5. Helping Wing Analyze Cython Code

Wing works best with Cython'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.

## 7.3. 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.

## Source Code Analysis

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.

## 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](mailto:support@wingware.com).

### 8.1. 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.

### 8.2. 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 wing-101-7.2** 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](mailto:support@wingware.com).

### 8.3. 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 **Configure Python** (accessed from the **Edit** menu) to your Python. Wing remembers this and the message should go away.

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.

## 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.

### **9.1. Wing 101 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 101" and related software components. All such software is referred to herein as the "Software Product." If you do not agree to the terms of this EULA, then do not install or use the Software Product. By using this software you acknowledge and agree to be bound by the following terms:

#### **1. GRANT OF NON-EXCLUSIVE LICENSE**

Wingware grants you the non-exclusive, non-transferable right to use this Software Product.

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.

All rights not expressly granted to you are retained by Wingware.

#### **2. 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.

#### **3. NO RIGHT 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.

#### **4. 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.

#### **5. THIRD PARTY RIGHTS**

## License Information

Any software provided along with the Software Product that is associated with a separate license agreement is licensed to you under the terms of that license agreement. This license does not apply to those portions of the Software Product. Copies of these third party licenses are included in all copies of the Software Product.

### 6. SUPPORT SERVICES

Wingware may provide you with support services related to the Software Product. Use of any such support services is governed by Wingware policies and programs described in online documentation and/or other Wingware-provided materials.

As part of these support services, Wingware may make available bug lists, planned feature lists, and other supplemental informational materials. WINGWARE MAKES NO WARRANTY OF ANY KIND FOR THESE MATERIALS AND ASSUMES NO LIABILITY WHATSOEVER FOR DAMAGES RESULTING FROM ANY USE OF THESE MATERIALS. FURTHERMORE, YOU MAY NOT USE ANY MATERIALS PROVIDED IN THIS WAY TO SUPPORT ANY CLAIM MADE AGAINST WINGWARE.

Any supplemental software code or related materials that Wingware provides to you as part of the support services, in periodic updates to the Software Product or otherwise, is to be considered part of the Software Product and is subject to the terms and conditions of this EULA.

With respect to any technical information you provide to Wingware as part of the support services, Wingware may use such information for its business purposes without restriction, including for product support and development. Wingware will not use such technical information in a form that personally identifies you without first obtaining your permission.

### 7. 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.

### 8. 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.

### 9. 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.

#### 10. 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.

#### 11. 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.

#### 12. 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



## License Information

Software Product, or any software, tool, process, or service that was developed using the Software Product, in such applications.

### 13. 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.

### 14. 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.

### 15. 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 rédigés 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.

### 16. 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"

### 17. 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](mailto:info@wingware.com).

### 9.2. 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](mailto:info@wingware.com).

## License Information

[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 sqlite.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:

## License Information

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 <IanLeel521@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:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE 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.

### **autopep8**

We are required by the license terms for autopep8 to include the following copyright notice in this documentation:

## License Information

Copyright (C) 2010-2011 Hideo Hattori  
Copyright (C) 2011-2013 Hideo Hattori, Steven Myint  
Copyright (C) 2013-2016 Hideo Hattori, Steven Myint, Bill Wendling

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:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE 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.

## Pygments

We are required by the license terms for Pygments to include the following copyright notice in this documentation:

Copyright (c) 2006-2019 by the respective authors (see AUTHORS file).  
All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are 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

## License Information

(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 "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.

### **getmac**

We are required by the license terms for getmac to include the following copyright notice in this documentation:

Copyright (c) 2017 Christopher Goes

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

## License Information

furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE 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.

### 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:

The MIT License (MIT)

Copyright (c) 2014 Daniel Hill aka RazorX - Identity e-mail: public at RazorX.com

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:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE 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.

### 9.3. 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

## License Information

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

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



## License Information

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.