Wing IDE 101 Reference Manual

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

Contents

Introduction

Thanks for choosing Wingware's Wing IDE 101! This manual will help you get started and serves as a reference for the entire feature set.

The manual is organized by major functional area of Wing IDE, including customization, source code editor, search/replace features, python shell, debugger and source code analysis

The rest of this chapter describes how to install and start using Wing IDE 101. See also the quick start guide and tutorial.

1.1. Supported Platforms

Wing IDE 6 is available for Microsoft Windows, Linux, and Mac OS X.

Microsoft Windows

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

Linux/Intel

Wing IDE runs on 64-bit Linux versions with glibc version 2.15 or later (such as Ubuntu 12.04+, CentOS 7+, Kali 1.1+, and Fedora 20+).

Mac OS X

Wing IDE runs on Mac OS X 10.7+ as a native application.

1.2. Supported Python versions

Wing supports CPython 2.5 through 2.7 and 3.2 through 3.6, Stackless Python 2.5 through 3.4, and cygwin Python 2.5 through 2.7. Wing can also be used with PyPy, IronPython, and Jython, but the debugger and Python Shell will not work with these implementations of Python.

Wing's debugger is pre-built for each of these versions of Python with and without **--with-pydebug**. Both 32-bit and 64-bit compilations are supported on Windows and OS X. On Linux only 64-bit Python is supported. CPython **--with-framework** builds are also supported on OS X.

It is also possible to compile Wing's debugger on other operating systems, and against custom versions of Python (requires signed NDA).

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

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

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

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

1.3. Prerequisites for Installation

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

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

1.4. Installing Wing IDE

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

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

Windows

Install Wing IDE by running the downloaded executable. Wing's files are installed by default in **C:\Program Files\Wing IDE 101 6.0**, but this location may be modified during installation. Wing will also create a User Settings Directory in the location appropriate for your version of Windows. This is used to store preferences and other settings.

The Windows installer supports a **/silent** command line option that uses the default options, including removing any prior install of version 6.0 of Wing IDE. If a prior install is removed, a dialog with a progress bar will appear. You can also use a **/dir=<dir name>** option to specify an alternate installation directory.

Linux

Use the RPM, Debian package, or tar file installer as appropriate for your system type. Installation from packages is at **/usr/lib/wingide-101-6** or at the selected location when installing from the tar file. Wing will also create a User Settings Directory in **~/.wing101-6**, which is used to store preferences and other settings.

For more information, see the Linux installation details.

Mac OS X

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

1.5. Running Wing IDE

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

On Windows, start Wing IDE from the Program group of the Start menu. You can also start Wing from the command line with **wing-101.exe** located inside the Wing IDE installation directory.

On Linux/Unix, just execute **wing-101-6.0** (which is on the **PATH** by default for RPM and Debian installs) or execute **wing-101** located inside the Wing IDE installation directory.

On Mac OS X, start Wing IDE by double clicking on the app folder or from the command line using **wing-101** inside **Contents/Resources** within the Wing IDE **.app** folder.

1.6. User Settings Directory

The first time you run Wing, it will create your **User Settings Directory** automatically. This directory is used to store your license, preferences, 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 6** 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 6
```

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

```
~/.wing101-6
```

Cache Directory

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

On Windows, the cache directory is located in the **AppData\Local** area. On Linux, it is **~/.cache/wing101-6** and on OS X, it can be found with the symbolic link **~/.wing101-6/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**.

1.7. Upgrading

If you are upgrading within the same minor version number of Wing (for example from 6.0 to 6.x) this will replace your previous installation. Once you have upgraded, your previous preferences and settings should remain and you should immediately be able to start using Wing.

If you are upgrading across major releases (for example from 5.1 to 6.0), this will install the new version along side your old version of Wing.

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

To install an upgrade, follow the steps described in Installing

See also Migrating From Older Versions.

1.7.1. Fixing a Failed Upgrade

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

Windows

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

Mac OS X

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

Linux Debian

If you installed Wing IDE for Linux from Debian package, issue the command **dpkg -r wingide6**. Then go into **/usr/lib/wingide6** and remove any remaining files and directories.

Linux RPM

If you installed Wing IDE for Linux from RPM, issue the command **rpm -e wingide6**. Then go into **/usr/lib/wingide6** and remove any remaining files and directories.

Linux Tar

If you installed Wing IDE for Linux from the tar distribution, run the **wing-uninstall** script located in the install directory listed in Wing's **About** box. Once done, manually remove any remaining files and directories.

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

1.8. Installation Details and Options

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

1.8.1. 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/wingide-101-6**. Only 64-bit Linux is supported.

Installing Wingware's Public Key

Some systems will complain when you try to install Wing IDE 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 wingide-101-6-6.0.2-1.amd64.rpm** as root or use your favorite RPM administration tool to install the RPM. Most files for

Wing are placed under the **/usr/lib/wingide-101-6** directory and the **wing-1016.0** command is placed in the **/usr/bin** directory.

Installing from Debian package:

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

To install, run **dpkg -i wingide-101-6_6.0.2-1_amd64.deb**

as root or use your favorite package administration tool to install. Most files for Wing are placed under the **/usr/lib/wingide-101-6** directory and the **wing-1016.0** 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/wingide-101-6**. Unpacking this archive with **tar -zxvf wingide-101-6.0.2-1-amd64-linux.tar.gz** will create a **wingide-101-6.0.2-1-amd64-linux** directory that contains the **wing-install.py** script and a **binary-package.tar** file.

Running the **wing-install.py** script will prompt for the location to install Wing, and the location in which to place the executable **wing-1016.0**. These locations default to **/usr/local/lib/wingide-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.

1.8.2. Remote Display on Linux

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

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

Speeding up the Connection

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

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

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

Other Options

Other options for displaying Wing remotely from Linux include:

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

1.8.3. Installing Extra Documentation

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

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

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

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

1.9. 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 IDE'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 reactivating your license if moving to a new machine.

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 it is 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 User Settings Directory can be shared to other machines or with other users of Wing IDE. This includes the following files and directories:

- **preferences** -- Wing IDE'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.

1.10. Removing Wing IDE

Windows

On Windows, use the Add/Remove Programs control panel, select **Wing IDE 101 6** and remove it.

Linux/Unix

To remove an RPM installation on Linux, type **rpm -e wingide-101-6**.

To remove an Debian package installation on Linux, type **dpkg -r wingide-101-6**.

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.

Customization

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

> ***Note***
>
> **These are some of the areas of customization that are available:**
>
> - The editor can run with different personalities such as VI/Vim, Emacs, Visual Studio, Eclipse, and Brief emulation
> - The action of the tab key can be configured
> - Many other options are available through preferences

2.1. Keyboard Personalities

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

> ### Note
>
> **Emulation of Other Editors**
>
> The first thing most users will want to do is to set the keyboard personality to emulate their editor of choice. This is done with the **Edit > Keyboard Personality** menu or with the **User Interface > Keyboard > Personality** preference.

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

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

2.2. User Interface Options

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

2.2.1. Display Style and Colors

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

Editor Color Configuration

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

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

UI Color Configuration

To apply the editor color palette also to the UI outside of the editor, enable the **Use Editor Palette Throughout the UI** preference.

Add Color Palettes

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

2.2.2. User Interface Layout

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

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

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

Configuring the Toolbar

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

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

Configuring the Editor Area

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

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

Configuring Toolboxes

The number of tool box splits Wing shows by default depends on your monitor size. Each of the toolboxes can be split or joined into any number of splits along the long axis of the toolbox by clicking on the options drop down icon in the tab area of the toolbox and selecting **Add Toolbox Split** or **Remove Toolbox Split**. This menu is also accessible by right-clicking on the tool tabs.

Toolbox splits can also be added or removed by dragging tools around by their tabs, either within each toolbox, to a different toolbox, or out to a new window. 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.

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.

2.2.3. Altering Text Display

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

2.3. Preferences

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

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

Source Code Editor

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

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

3.1. Syntax Colorization

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

All the available colorization document types are listed in the **Source** -> **Current File Properties** dialog's File Attributes tab. If you have a file that is not being recognized automatically, you can use the **File Type** menu found there to alter the way the file is being displayed.

3.2. Right-click Editor Menu

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

3.3. Navigating Source

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

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

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

on OS X) also jumps to the point of definition unless the **Editor > Advanced** preference for this feature is disabled.

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

Moving to other files can also be done with the **Window** menu, which lists all open files.

3.4. File status and read-only files

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

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

3.5. Transient, Sticky, and Locked Editors

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

Transient Mode -- Files opened when debugging or navigating to point of definition are opened in transient mode and will be automatically closed when hidden.

Sticky Mode -- Files opened from the File menu or from the keyboard file selector will be opened in sticky mode, and are kept open until they are explicitly closed.

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

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

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

3.6. Auto-completion

Wing IDE can display an auto-completer in the editor and shells. In Wing 101, this feature is disabled by default but can be enabled with the **Show Auto-completer** preference.

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

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

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

Auto-Completer Icons

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

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

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

	A Python keyword
	A module name
	A class name
	A Python package (a directory with __init__.py in it)
	A method name
	A function name
	An object instance (other than the basic types below)
	A dictionary
	A tuple
	A list
	A string

	An integer
	A float
	An exception
	A Python stack frame

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

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

How Auto-completion Works

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

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

3.7. Indentation

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

3.7.1. How Indent Style is Determined

When an existing file is opened, it is scanned to determine what type of indentation is used in that file. Wing then matches new indentation as the file is edited 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 using the size set in preferences.

3.7.2. Indentation Preferences

The following preferences affect how the indentation features behave:

1. The **Default Indent Size** preference defines the default size of each level of indent, in spaces. This is used in new empty files only. Wing may override this value in files that contain only tabs in indentation, in order to make it a multiple of the configured tab size.
2. The **Show Indent Guides** preference controls whether or not to show indentation guides as light vertical lines. This value can be overridden on a file-by-file basis from Editor tab in File Properties.

3.7.3. Auto-Indent

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

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

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

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

3.7.4. The Tab Key

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

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

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

Default for Personality

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

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

Indent to Match

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

Move to Next Tab Stop

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

Indent Region

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

Insert Tab Character

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

Smart Tab

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

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

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

    b. In other cases, indentation is reduced by one level.

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

3.7.5. Changing Block Indentation

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

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

3.8. Brace Matching

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

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

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

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

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

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

3.10. Notes on Copy/Paste

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

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

It is important to note which actions use the system-wide clipboard, which use the emacs private clipboard or VI registers, and which use the X11 selection (Linux only). Otherwise, these commands are interchangeable in their effects.

Smart Copy

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

Search/Replace

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

4.1. Toolbar Quick Search

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

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

If focus is off the toolbar search area and it already contains a search string, clicking on it will immediately start searching in the current source editor for the next match. If you wish to search for another string instead, delete the text and type the desired search string. As you delete, the match position in the editor will proceed backward until it reaches your original search start position, so that after typing your new search string you will be presented with the first match after the original source editor cursor position.

4.2. Search Tool

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

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

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

The following search options can be selected from the tool:

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

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

- **Show Replace** -- Whether or not the Replace field is visible in the tool.
- **Text Search** -- Select this to do a regular text search without wildcard or regex.

- **Wildcard Search** -- Select this to allow use of special characters for wildcarding in the search string (see Wildcard Search Syntax for details).
- **Regex Search** -- Select this to use regular expression style searching. This is a more powerful variant than wildcard search that allows for more complex specification of search matches and replacement values. For information on the syntax allowed for the search and replace strings, see Python's Regular Expression Syntax documentation. In this mode, the replace string can reference regex match groups with **\1**, **\2**, etc, as in the Python **re.sub()** call.
- **Wrap Search** -- Uncheck this to avoid wrapping around when the search reaches the top or bottom of a file.
- **Incremental** -- Check this to immediately start or restarted searching as you type or alter search options. When unchecked, use the forward/backward search buttons to initiate searching.
- **Find After Replace** -- Select this to automatically find the next search match after each Replace operation.

4.3. Wildcard Search Syntax

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

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

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

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

Interactive Python Shell

Wing provides an integrated Python Shell for execution of commands and experimental evaluation of expressions. The version of Python used in the Python Shell, and the environment it runs with, is configured using **Configure Python** in the **Edit** menu.

This shell runs a separate Python process that is independent of the IDE and functions without regard to the state of any running debug process.

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

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

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

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

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

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

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

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

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

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

5.1. Active Ranges in the Python Shell

Code in an editor can be set up as the active range on which the **Python Shell** will operate, to make it easier to reevaluate after it is edited. This is done by selecting a range of lines in an editor and pressing the icon at the top right of the **Python Shell** to set the active range.

Once this is done, additional icons appear for executing the active range, jumping to the active range in the code editor, or clearing the active range. The active range is highlighted in the code editor and should adjust its start/end lines as code is added or deleted.

5.2. Python Shell Auto-completion

Wing's Python Shell includes auto-completion, which can be a powerful tool for quickly finding and investigating functionality at runtime, for the purposes of code learning, or in the process of crafting new code. The Python Shell's completer is fueled by introspection of the runtime environment. In Wing 101, this feature is disabled by default but can be enabled with the **Show Auto-completer** preference.

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

5.3. Debugging Code in the Python Shell

Code executed in Wing's **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 the **Enable Debugging** item 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 here as in editors. This works for code previously typed, dragged, or pasted into the shell. Breakpoints set in editors will also be reached, if that code ends up being executed. Wing will copy breakpoints from a source file and stop in the Python Shell itself when **Evaluate Selection** is used on a short enough range of code. However, when using active ranges or evaluating a long selection or whole file Wing instead stops at breakpoints set within the code editor, since in those cases the code is not visible in the shell itself.

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 **Debug Probe** 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 allows other threads to run as well.

In the **Debug Probe**, when debugging is disabled in its **Options** menu, threads are debugged but are halted whenever the main thread is halted in the debugger. Threads are not run even while executing code from the prompt in the **Debug Probe** so that data in all threads can be inspected without any unexpected change in runtime state caused by running of a thread. Threads will only continue running when the main debug program is continued. This is true whether or not the debug program was started from a file, or from within the **Python Shell**. As in the **Python Shell**, when debugging is enabled in the **Debug Probe** child threads will also be allowed to run whenever code is being executed recursively and the **Debug Probe** is not at the prompt. Threads are still halted whenever the **Debug Probe** is at the prompt

These subtle but necessary differences in threading behavior may affect how threaded code performs within the **Python Shell** and **Debug Probe**. 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.

5.4. Python Shell Options

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

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

- **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 cursor or selection is located).

6.2. Starting Debug

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

- Choose **Start / Continue** from the **Debug** menu or push the **Debug** icon in the toolbar. This will run the current file open in the editor. Execution stops at the first breakpoint or exception, or upon program completion.
- Choose **Step Into** from the **Debug** menu or push the **Step Into** icon in the toolbar. This will run the main debug file if one has been defined, or otherwise the file open in the frontmost editor window. Execution stops at the first line of code.
- Use one of the key bindings given in the **Debug** menu.
- Code can also be debugged from the Python Shell 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.

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. Mousing over the bug icon shows expanded debugger status information in a tool tip. The color of the bug icon summarizes the status of the debug process, as follows:

- **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.

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

6.4. Flow Control

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

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

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

Step Over 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

6.5. Viewing the Stack

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

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

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

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

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

6.6. Viewing Debug Data

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

Note

Values Fetched on Demand

The variable data displayed by Wing is fetched from the debug server on the fly as you navigate. Because of this, you may experience a brief delay when a change in an expansion or stack frame results in a large data transfer.

For the same reason, leaving large amounts of debug data visible on screen may slow down stepping through code.

6.6.1. Stack Data View

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

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

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

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

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

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

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

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

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

> ***Note***
>
> **Opaque Data**

Some data types, such as those defined only within C/C++ code, or those containing certain Python language internals, cannot be transferred over the network. These are denoted with **Value** entries in the form **<opaque 0x80ce784>** and cannot be expanded further.

6.6.1.1. Popup Menu Options

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

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

6.6.2. Problems Handling Values

The Wing debugger tries to handle debug data as gently as possible to avoid entering into lengthy computations or triggering errors in the debug process while it is packaging debug data for transfer. Even so, not all debug data can be shown on the display. This section describes each of the reasons why this may happen:

Wing may time out handling a value -- Large data values may hang up the debug server process during packaging. Wing tries to avoid this by carefully probing an object's size before packing it up. In some cases, this does not work and Wing will wait for the data for the duration set by the **Network Timeout** preference and then will display the variable value as **<network timeout during evaluate>**.

Wing may encounter values too large to handle -- Wing will not package and transfer large sequences, arrays or strings that exceed the size limits set by **Huge List Threshold** and **Huge String Threshold** preferences. On the debugger

display, oversized sequences and arrays are annotated as **huge** and **<truncated>** is prepended to large truncated strings.

To avoid this, increase the value of the threshold preferences, but be prepared for longer data transfer times. Note that setting these values too high will cause the debugger to time out if the **Network Timeout** value isn't also increased.

Wing may encounter errors during data handling -- Because Wing makes assignments and comparisons during packaging of debug data, and because it converts debug data into string form, it may execute special methods such as **__cmp__** and **__str__** in your code. If this code has bugs in it, the debugger may reveal those bugs at times when you would otherwise not see them.

The rare worst case scenario is crashing of the debug process if flawed C or C++ extension module code is invoked. In this case, the debug session is ended.

More common, but still rare, are cases where Wing encounters an unexpected Python exception while handling a debug data value. When this happens, Wing displays the value as **<error handling value>**.

These errors are not reported as normal program errors in the Exceptions tool. However, extra output that may contain the exception being raised can be obtained by setting the **Debug Internals Log File** preference.

Stored Value Errors

Wing remembers errors it encounters on debug values and stores these in the project file. These values will not be refetched during subsequent debugging, even if Wing is quit and restarted.

To override this behavior for an individual value, use the **Force Reload** item in the right-click context menu on a data value.

To clear the list of all errors previously encountered so that all values are reloaded, use the **Clear Stored Value Errors** item in the **Debug** menu. This operates only on the list of errors known for the current debug file, if a debug session is active, or for the main debug file, if any, when no debug process is running.

6.7. Debug Process I/O

While running under the Wing debugger, any output from **print** or any writes to **stdout** or **stderr** will be seen in the **Debug I/O** tool. This is also where you enter keyboard input, if your debug program requests any with **input()** or **raw_input()** or by reading from **stdin**.

6.8. Debugging Multi-threaded Code

Wing's debugger can debug multi-threaded code, as well as single-threaded code. By default, Wing will debug all threads and will stop all threads if a single thread stops. If multiple threads are present in the debug process, the Stack Data tool (and in Wing Pro the Debug Probe and Watch tools) will add a thread selector popup to the stack selector.

Even though Wing tries to stop all threads, some may continue running if they do not enter any Python code. In that case, the thread selector will list the thread as running. It also indicates which thread was the first one to stop.

When moving among threads in a multi-threaded program, the Show Position icon shown in the toolbar during debugging (between the up/down frame icons) is a convenient way to return to the original thread and stopping position.

Whenever debugging threaded code, please note that the debugger's actions may alter the order and duration that threads are run. This is a result of the small added overhead, which may influence timing, and the fact that the debugger communicates with the IDE through a TCP/IP connection.

Selecting Threads to Debug

Currently, the only way to avoid stopping all threads in the debugger is to launch your debug process from outside Wing, import wingdbstub, and use the debugger API's **SetDebugThreads()** call to specify which threads to debug. All other threads will be entirely ignored. This is documented in Debugging Externally Launched Code and the API is described in Debugger API

An example of this can be seen in the file **DebugHttpServer.py** that ships with Wing's support for Zope and Plone. To see this, unpack the WingDBG archive found inside the **zope** directory in your Wing installation.

Note, however, that specifying a subset of threads to debug may cause problems in some cases. For example, if a non-debugged thread starts running and does not return control to any other threads, then Wing's debugger will cease to respond to the IDE and the connection to the debug process will eventually be closed. This is unavoidable as there is no way to preemptively force the debug-enabled threads to run again.

6.9. OS X Debugging Notes

System-Provided Python

The copy of Python in **/Library/Python** on OS X does not include source files for the standard libraries, so Wing's editor will not offer autocompletion values for those modules. To work around this, use Python from within **/Library/Frameworks/Python.frameworks** instead or copy of Python installed from the standard source distribution.

MacPorts Python

At least some versions of the MacPorts packaging of Python are known not to work with Wing's debugger because it contains an **_md5** module that won't load. To work around this, use a different distribution of Python instead.

Debugging 32-bit Python on a 64-bit System

On 64-bit OS X systems, you can set up a shell script with the following contents and set it as the Python Executable in Project Properties, in order to facilitate debugging Python in 32-bit mode:

```
#!/bin/bash
arch -i386 python "$@"
```

This should only be necessary if your code needs 32-bit libraries. Wing's debugger works in either 64-bit or 32-bit mode.

Source Code Analysis

Wing's auto-completer, source assistant, source index menu, goto-definition capability, find uses, refactoring, and other features all rely on a central engine that reads and analyzes your source code in the background as you add files to your project or alter your code in the source code editor. This engine can also load and inspect extension modules used by your code, can make use of live runtime state when available in a debug process or in the integrated Python Shell, and can read user-provided interface description files.

7.1. How Analysis Works

In analysing your source, Wing will use the Python interpreter and **PYTHONPATH** that you have specified using **Configure Python** in the **Edit** menu. Whenever any of these values changes, Wing will re-analyze some or all of your source code.

When Wing tries to find analysis information for a particular module or file, it takes the following steps:

- The path and same directory as the referencing module are searched for an importable module
- If the module is Python code, Wing statically analyses the code to extract information from it
- If the module is an extension module, Wing looks for a ***.pi** interface description file as described later in this section
- If the module cannot be found, Wing tries to import it in a separate process space in order to analyze its contents
- If a debug process is active, Wing tries to read relevant type information from the live runtime state associated with the source code

7.2. Static Analysis Limitations

The following are known limitations affecting features based on static source analysis:

- Argument number, name, and type is not determined for functions and methods in extension modules.
- Analysis sometimes fails to identify the type of a construct because Python code doesn't always provide clues to determine the data type.
- Types of elements in lists, tuples, and dictionaries are not identified.
- Analysis information may be out of date if you edit a file externally with another editor and don't reload it in Wing. See section Auto-reloading Changed Files for reload options.
- From time to time, as Python changes, some newer Python language constructs and possible type inferencing cases are not supported.

A good way to work around these limitations, when they arise, is to place a breakpoint in the code where you are working, run to it, and then auto-completion and other information presented by the IDE will be based on the actual runtime state rather than static analysis.

See Helping Wing Analyze Code for more information.

7.3. Helping Wing Analyze Code

Wing's source analyser can only read Python code and does not contain support for understanding C/C++ extension module code other than by attempting to import the extension module and introspecting its contents (which yields only a limited amount of information and cannot determine argument number, name, or types). Also, since Python is a dynamic language, it is possible to craft code that Wing's static analysis engine cannot understand.

There are a number of ways of assistant Wing's static source analyzer in determining the type of values in Python code.

Using Live Runtime State

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

In the editor, the cog icon in the auto-completer indicates that type information was found in the live runtime state.

Using PEP484 and PEP 526 to Assist Analysis

Wing can understand type hints in the style standardized by PEP 484 (Python 3.5+) and PEP 527 (Python 3.6+). 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 it:

```
x = Something() # type: int
```

In Python 3.6+ the type can instead be specified inline as follows:

```
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**.

Limitation: Wing currently cannot remember the type of the elements of lists, tuples, dicts, and sets.

Using isinstance() to Assist Analysis

One way to inform the static analysis engine of the type of a variable is to add an **isinstance** call in your code. For example **isinstance(obj, CMyClass)** or **assert isinstance(obj, CMyClass)** when runtime type checking is desired. The code analyzer will pick up on these and present more complete information for the asserted values.

In cases where doing this introduces a circular import, you can use a conditional to allow Wing's static analyser to process the code without causing problems when it is executed:

```
if 0:
  import othermodule
  assert isinstance(myvariable, othermodule.COtherClass)
```

In most code, a few **isinstance** calls go a long way to making code faster and easier to edit and navigate.

*Using *.pi or *.pyi Files to Assist Analysis*

It is also possible to create a ***.pi** or ***.pyi** (Python Interface) file that describes the contents of a module. 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 IDE will read this file and merge its contents with any information it can obtain through static analysis or by loading an extension module. If the file has a **.pyi** extension, it can use PEP 484 and PEP 526 type annotations regardless of whether Python 2 or Python 3 is used. Newly written interface files should follow PEP 484 and use the **.pyi** extension; the **.pi** extension was used in previous versions of Wing and is still recognized.

In somes cases, as for Python bindings for GUI and other toolkits, these ***.pi** or ***.pyi** files can be auto-generated from interface description files. The code that Wing uses to automatically generate ***.pi** files from extension modules is in **src/wingutils/generate_pi.py** in your Wing IDE 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** as **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** for the module is 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 > Interfaces Path** preference.
2. In the **resources/builtin-pi-files** in the Wing IDE 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 an 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.

For Python source modules, absence of a ***.pyi** causes Wing to fall back on static analysis and (if available) runtime analysis through the debugger.

*Merging *.pyi Name Spaces*

When Wing finds a ***.pyi** file in the same directory as a Python module or extension module, or if it finds it using the **Source Analysis > Advanced > Interfaces Path** preference, then Wing merges the contents of the ***.pyi** file with any information found by analyzing or introspecting the module. The contents of the ***.pyi** file take precedence when symbols are defined in both places.

Creating 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.

As noted above, 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.

7.4. Analysis Disk Cache

The source code analyzer writes information about files it has recently examined into the Cache Directory that is listed in Wing's About box, which is accessed from the **Help** menu.

Cache size may be controlled with the **Max Cache Size** preference However, Wing does not perform well if the space available for the cache is smaller than the space needed for a single project's source analysis information. If you see excessive sluggishness, either increase the size of the cache or disable it entirely by setting its size to 0.

If the same cache will be used by more than one computer, make sure the clocks of the two computers are synchronized. The caching mechanism uses time stamps, and may become confused if this is not done.

The analysis cache may be removed in its entirety. Wing IDE will reanalyze your code and recreate the cache as necessary.

Trouble-shooting Guide

This chapter describes what to do if you are having trouble installing or using Wing IDE.

> ***Note***
>
> We welcome feedback and bug reports, both of which can be submitted directly from Wing IDE using the **Submit Feedback** and **Submit Bug Report** items in the Help menu, or by emailing us at support at wingware.com.

8.1. Trouble-shooting Failure to Start

If you are having trouble getting Wing to start at all, read through this section for information on diagnosing the problem.

To rule out problems with a project file or preferences, try renaming your User Settings Directory and restart Wing. If this works, you can copy over files from the renamed directory one at a time to isolate the problem -- or email support at wingware dot com for help.

On Windows, the user's temporary directory sometimes becomes full, which prevents Wing from starting. Check whether the directory contains more than 65,000 files.

On Linux, OS X, or other Posix systems, in some cases when the **~/.cache** directory or the cache directory set by the **$XDG_CACHE_DIR** is located on an NFS or other remote file server, Wing can't obtain a lock on a database file. To use slower, dotfile locking set the **Use sqlite dotfile locking** preference to enabled or run Wing with the **--use-sqlite-dotfile-locking** command line option. Note that all Wing processes, regardless of the system they're running on, that use the same cache directory need to either use or not use dotfile locking.

Under a Windows terminal server, Wing may not be able to set up the environment variables it uses internally and will not start up. In this case, you can get Wing to start with the following commands:

```
set PYTHONOPTIMIZE=1
set PYTHONHOME=D:\Program Files\WingIDE\bin\PyCore
wing.exe
```

Alter **PYTHONHOME** according to the location at which you've installed Wing IDE.

Constant Guard from Comcast can prevent Wing IDE from starting without showing any dialog or message that it is doing so.

8.2. Speeding up Wing

Wing should present a responsive, snappy user interface even on relatively slow hardware. In some cases, Wing may appear sluggish:

With New Projects, the first time you set up a project file, Wing analyzes all source files for the source code browser and auto-completion facilities. During this time, the browser's class-oriented views will display only the source constructs from files of which analysis information has already been obtained. The user interface may also appear to be sluggish and Wing will consume substantial amounts of CPU time.

To avoid this in subsequent sessions, Wing stores its source analysis information to disk in a cache within your User Settings Directory.

On a multi-core virtual machine where Wing runs slowly, you may be able to improve performance by setting the processor affinity for Wing. This is done with **schedtool -a 0x1 -e wing-1016.0** on Linux (the schedtool package needs to be installed if not already present) and with **START /AFFINITY 01 "Wing IDE" "C:\Program Files\Wing IDE 6.0\bin\wing.exe"** on Windows. Although Wing runs on only one core, this technique has been reported to improve performance.

On OS X Mavericks, certain graphics drivers have a bug that substantially slows down Wing IDE because the OS is incorrectly detecting Wing IDE as inactive. Turning off App Nap has no effect on this, although the bug may be associated with that feature. The work-around is to put the computer to sleep briefly while Wing IDE is already running. Wing should then remain responsive until it is quit.

8.3. Trouble-shooting Other Known Problems

Here are some other known problems that can affect some of Wing IDE's functionality:

Windows File Names with Spaces

When using Windows File Types or Open With to cause Python files to be opened with Wing, some versions of Windows set up the wrong command line for opening the file. You can fix this using *regedt32.exe*, *regedit.exe*, or similar tool to edit the following registry location:

```
HKEY_LOCAL_MACHINE\SOFTWARE\Classes\Applications\wing.exe\shell\open\command
```

The problem is that the association stored there is missing quotes around the *%1* argument. It should instead read as follows:

```
"C:\Program Files\Wing IDE\bin\wing.exe" "%1" %*
```

Copy/Paste Fails on Windows

Webroot Secure Anywhere v8.0.4.66 blocks Wing IDE and Python's access to the clipboard by default so Copy/Paste will not work. The solution is to remove Wing IDE and Python from the list of applications that Webroot is denying access to the clipboard.

Failure to Find Python

Wing scans for Python at startup and may sometimes report that it could not be found even if it is on your machine.

If this happens all the time, point **Python Executable** in **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 causes this during periods of intensive scanning.

Failure to Detect HTTP Proxy and Connect to wingware.com

Wing will try to open an http connection to **wingware.com** when you activate a license, check for product updates, or submit feedback or a bug report. If you are running in an environment with an http proxy, Wing will try to auto-detect your proxy settings. If this fails you will need to configure your proxy manually using Wing's **HTTP Proxy Server** preference. To determine the correct settings to use, ask your network administrator or see how to determine proxy settings.

License Information

Wing IDE is a commercial product that is based on a number of open source technologies. Although the product source code is available for Wing IDE Professional users (with signed non-disclosure agreement) the product is not itself open source.

The following sections describe the licensing of the product as a whole (the End User License Agreement) and provide required legal statements for the incorporated open source components.

9.1. Wing IDE Software License

This End User License Agreement (EULA) is a CONTRACT between you (either an individual or a single entity) and Wingware, which covers your use of "Wing IDE 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

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 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 redigés en langue anglaise. (translation: "The parties confirm that this EULA and all related documentation is and will be in the English language.") You are responsible for complying with any local laws in your jurisdiction which might impact your right to import, export or use the Software Product, and you represent that you have complied with any regulations or registration procedures required by applicable law to make this license enforceable.

16 TRADEMARKS

.

The following are trademarks or registered trademarks of Wingware: Wingware, the feather logo, Wing IDE, Wing IDE 101, Wing IDE Personal, Wing IDE Professional, Wing 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.

9.2. Open Source License Information

Wing IDE incorporates the following open source technologies, most of which are under OSI Certified Open Source licenses except as indicated in the footnotes:

- Crystal Clear -- An icon set by Everaldo -- LGPL v. 2.1 [1]
- docutils -- reStructuredText markup processing by David Goodger and contributors-- Public Domain [2]
- parsetools -- Python parse tree conversion tools by John Ehresman -- MIT License
- pexpect -- Sub-process control library by Noah Spurrier, Richard Holden, Marco Molteni, Kimberley Burchett, Robert Stone, Hartmut Goebel, Chad Schroeder, Erick Tryzelaar, Dave Kirby, Ids vander Molen, George Todd, Noel Taylor, Nicolas D. Cesar, Alexander Gattin, Geoffrey Marshall, Francisco Lourenco, Glen Mabey, Karthik Gurusamy, and Fernando Perez -- MIT License

- PySide -- Python bindings for Qt by Nokia and contributors -- LGPL v. 2.1 [1]
- pysqlite -- Python bindings for sqlite by Gerhard Haering -- BSD-like custom license [4]
- Python -- The Python programming language by Guido van Rossum, PythonLabs, and many contributors -- Python Software Foundation License version 2 [3]
- Python Imaging Library -- Library for image manipulation with Python, written by Secret Labs AB and Fredrik Lundh -- MIT License
- Qt -- Graphical user interface toolkit by many contributors and Digia -- LGPL v. 2.1 [1] [6]
- scintilla -- Source code editor component by Neil Hodgson and contributors -- MIT License
- sqlite -- A self-contained, serverless, zero-configuration, transactional SQL database engine -- Public domain [5]
- Tulliana-1.0 -- An icon set by M. Umut Pulat, based on Nuvola created by David Vignoni -- LGPL v. 2.1 [1]
- A few stock icons from the GTK GUI development framework -- LGPL v. 2.1 [1]

Notes

[1] The LGPL requires us to redistribute the source code for all libraries linked into Wing IDE. All of these modules are readily available on the internet. In some cases we may have modifications that have not yet been incorporated into the official versions; if you wish to obtain a copy of our version of the sources of any of these modules, please email us at info at wingware.com.

[2] Docutils contains a few parts under other licenses (BSD, Python 2.1, Python 2.2, Python 2.3, and GPL). See the COPYING.txt file in the source distribution for details.

[3] The Python Software Foundation License version 2 is an OSI Approved Open Source license. It consists of a stack of licenses that also include other licenses that apply to older parts of the Python code base. All of these are included in the OSI Approved license: PSF License, BeOpen Python License, CNRI Python License, and CWI Python License. The intellectual property rights for Python are managed by the Python Software Foundation.

[4] Not OSI Approved, but similar to other OSI approved licenses. The license grants anyone to use the software for any purpose, including commercial applications.

[5] The source code states the author has disclaimed copyright of the source code. The sqllite.org website states: "All of the deliverable code in SQLite has been dedicated to the public domain by the authors. All code authors, and representatives of the companies they work for, have signed affidavits dedicating

their contributions to the public domain and originals of those signed affidavits are stored in a firesafe at the main offices of Hwaci. Anyone is free to copy, modify, publish, use, compile, sell, or distribute the original SQLite code, either in source code form or as a compiled binary, for any purpose, commercial or non-commercial, and by any means."

[6] Qt is available under several licenses. The LGPL v. 2.1 version of the software was used for Wing IDE.

Scintilla Copyright

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

```
Copyright 1998-2003 by Neil Hodgson <neilh@scintilla.org>

All Rights Reserved

Permission to use, copy, modify, and distribute this software and its
documentation for any purpose and without fee is hereby granted,
provided that the above copyright notice appear in all copies and that
both that copyright notice and this permission notice appear in
supporting documentation.

NEIL HODGSON DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS
SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY
AND FITNESS, IN NO EVENT SHALL NEIL HODGSON BE LIABLE FOR ANY
SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS,
WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER
TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE
OR PERFORMANCE OF THIS SOFTWARE.
```

Python Imaging Library Copyright

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

```
The Python Imaging Library (PIL) is

    Copyright ï¿½ 1997-2011 by Secret Labs AB
    Copyright ï¿½ 1995-2011 by Fredrik Lundh

By obtaining, using, and/or copying this software and/or its associated documentation,
that you have read, understood, and will comply with the following terms and condition

Permission to use, copy, modify, and distribute this software and its associated docu
any purpose and without fee is hereby granted, provided that the above copyright noti
all copies, and that both that copyright notice and this permission notice appear in
documentation, and that the name of Secret Labs AB or the author not be used in advert
publicity pertaining to distribution of the software without specific, written prior
```

License Information

```
SECRET LABS AB AND THE AUTHOR DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, I
IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL SECRET LABS AB OR
BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER
FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTH
ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
```