Macintosh Library Modules

Release 2.5.3

Guido van Rossum
Fred L. Drake, Jr., editor

19th December, 2008

Python Software Foundation
Email: docs@python.org

Abstract

This library reference manual documents Python's extensions for the Macintosh. It should be used in conjunction with the *Python Library Reference*, which documents the standard library and built-in types.

This manual assumes basic knowledge about the Python language. For an informal introduction to Python, see the *Python Tutorial*; the *Python Reference Manual* remains the highest authority on syntactic and semantic questions. Finally, the manual entitled *Extending and Embedding the Python Interpreter* describes how to add new extensions to Python and how to embed it in other applications.

CONTENTS

CHAPTER

ONE

Using Python on a Macintosh

Python on a Macintosh running Mac OS X is in principle very similar to Python on any other UNIX platform, but there are a number of additional features such as the IDE and the Package Manager that are worth pointing out.

Python on Mac OS 9 or earlier can be quite different from Python on UNIX or Windows, but is beyond the scope of this manual, as that platform is no longer supported, starting with Python 2.4. See http://www.cwi.nl/~jack/macpython for installers for the latest 2.3 release for Mac OS 9 and related documentation.

1.1 Getting and Installing MacPython

Mac OS X 10.4 comes with Python 2.3 pre-installed by Apple. However, you are encouraged to install the most recent version of Python from the Python website (http://www.python.org). A "universal binary" build of Python 2.5, which runs natively on the Mac's new Intel and legacy PPC CPU's, is available there.

What you get after installing is a number of things:

- A 'MacPython 2.5' folder in your 'Applications' folder. In here you find IDLE, the development environment that is a standard part of official Python distributions; PythonLauncher, which handles double-clicking Python scripts from the Finder; and the "Build Applet" tool, which allows you to package Python scripts as standalone applications on your system.
- A framework '/Library/Frameworks/Python.framework', which includes the Python executable and libraries. The installer adds this location to your shell path. To uninstall MacPython, you can simply remove these three things. A symlink to the Python executable is placed in /usr/local/bin/.

The Apple-provided build of Python is installed in '/System/Library/Frameworks/Python.framework' and '/usr/bin/python', respectively. You should never modify or delete these, as they are Apple-controlled and are used by Apple- or third-party software.

IDLE includes a help menu that allows you to access Python documentation. If you are completely new to Python you should start reading the tutorial introduction in that document.

If you are familiar with Python on other UNIX platforms you should read the section on running Python scripts from the UNIX shell.

1.1.1 How to run a Python script

Your best way to get started with Python on Mac OS X is through the IDLE integrated development environment, see section 1.2 and use the Help menu when the IDE is running.

If you want to run Python scripts from the Terminal window command line or from the Finder you first need an editor to create your script. Mac OS X comes with a number of standard UNIX command line editors, **vim** and **emacs** among them. If you want a more Mac-like editor, **BBEdit** or **TextWrangler** from Bare Bones Software (see http://www.barebones.com/products/bbedit/index.shtml) are good choices, as is **TextMate** (see http://macromates.com/). Other editors include **Gvim** (http://macvim.org) and **Aquamacs** (http://aquamacs.org).

To run your script from the Terminal window you must make sure that '/usr/local/bin' is in your shell search path.

To run your script from the Finder you have two options:

- Drag it to **PythonLauncher**
- Select **PythonLauncher** as the default application to open your script (or any .py script) through the finder Info window and double-click it. **PythonLauncher** has various preferences to control how your script is launched. Option-dragging allows you to change these for one invocation, or use its Preferences menu to change things globally.

1.1.2 Running scripts with a GUI

With older versions of Python, there is one Mac OS X quirk that you need to be aware of: programs that talk to the Aqua window manager (in other words, anything that has a GUI) need to be run in a special way. Use **pythonw** instead of **python** to start such scripts.

With Python 2.5, you can use either **python** or **pythonw**.

1.1.3 Configuration

Python on OS X honors all standard UNIX environment variables such as PYTHONPATH, but setting these variables for programs started from the Finder is non-standard as the Finder does not read your '.profile' or '.cshrc' at startup. You need to create a file '˜/.MacOSX/environment.plist'. See Apple's Technical Document QA1067 for details.

For more information on installation Python packages in MacPython, see section 1.3, "Installing Additional Python Packages."

1.2 The IDE

MacPython ships with the standard IDLE development environment. A good introduction to using IDLE can be found at http://hkn.eecs.berkeley.edu/ dyoo/python/idle_intro/index.html.

1.3 Installing Additional Python Packages

There are several methods to install additional Python packages:

- http://pythonmac.org/packages/ contains selected compiled packages for Python 2.5, 2.4, and 2.3.
- Packages can be installed via the standard Python distutils mode ('`python setup.py install`').
- Many packages can also be installed via the **setuptools** extension.

1.4 GUI Programming on the Mac

There are several options for building GUI applications on the Mac with Python.

PyObjC is a Python binding to Apple's Objective-C/Cocoa framework, which is the foundation of most modern Mac development. Information on PyObjC is available from http://pybojc.sourceforge.net.

The standard Python GUI toolkit is `Tkinter`, based on the cross-platform Tk toolkit (http://www.tcl.tk). An Aqua-native version of Tk is bundled with OS X by Apple, and the latest version can be downloaded and installed from http://www.activestate.com; it can also be built from source.

wxPython is another popular cross-platform GUI toolkit that runs natively on Mac OS X. Packages and documentation are available from http://www.wxpython.org.

PyQt is another popular cross-platform GUI toolkit that runs natively on Mac OS X. More information can be found at http://www.riverbankcomputing.co.uk/pyqt/.

1.5 Distributing Python Applications on the Mac

The "Build Applet" tool that is placed in the MacPython 2.5 folder is fine for packaging small Python scripts on your own machine to run as a standard Mac application. This tool, however, is not robust enough to distribute Python applications to other users.

The standard tool for deploying standalone Python applications on the Mac is **py2app**. More information on installing and using py2app can be found at http://undefined.org/python/#py2app.

1.6 Application Scripting

Python can also be used to script other Mac applications via Apple's Open Scripting Architecture (OSA); see http://appscript.sourceforge.net. Appscript is a high-level, user-friendly Apple event bridge that allows you to control scriptable Mac OS X applications using ordinary Python scripts. Appscript makes Python a serious alternative to Apple's own *AppleScript* language for automating your Mac. A related package, *PyOSA*, is an OSA language component for the Python scripting language, allowing Python code to be executed by any OSA-enabled application (Script Editor, Mail, iTunes, etc.). PyOSA makes Python a full peer to AppleScript.

1.7 Other Resources

The MacPython mailing list is an excellent support resource for Python users and developers on the Mac:

http://www.python.org/community/sigs/current/pythonmac-sig/

Another useful resource is the MacPython wiki:

http://wiki.python.org/moin/MacPython

CHAPTER

TWO

MacPython Modules

The following modules are only available on the Macintosh, and are documented here:

macpath	MacOS path manipulation functions.
macfs	Support for FSSpec, the Alias Manager, **finder** aliases, and the Standard File package.
ic	Access to Internet Config.
MacOS	Access to Mac OS-specific interpreter features.
macostools	Convenience routines for file manipulation.
findertools	Wrappers around the **finder**'s Apple Events interface.
EasyDialogs	Basic Macintosh dialogs.
FrameWork	Interactive application framework.
autoGIL	Global Interpreter Lock handling in event loops.

2.1 `macpath` — MacOS path manipulation functions

This module is the Mac OS 9 (and earlier) implementation of the `os.path` module. It can be used to manipulate old-style Macintosh pathnames on Mac OS X (or any other platform). Refer to the *Python Library Reference* for documentation of `os.path`.

The following functions are available in this module: `normcase()`, `normpath()`, `isabs()`, `join()`, `split()`, `isdir()`, `isfile()`, `walk()`, `exists()`. For other functions available in `os.path` dummy counterparts are available.

2.2 `macfs` — Various file system services

Deprecated since release 2.3. The macfs module should be considered obsolete. For `FSSpec`, `FSRef` and `Alias` handling use the `Carbon.File` or `Carbon.Folder` module. For file dialogs use the `EasyDialogs` module. Also, this module is known to not work correctly with UFS partitions.

This module provides access to Macintosh `FSSpec` handling, the Alias Manager, **finder** aliases and the Standard File package.

Whenever a function or method expects a *file* argument, this argument can be one of three things: (1) a full or partial Macintosh pathname, (2) an `FSSpec` object or (3) a 3-tuple `(`*wdRefNum*`, `*parID*`, `*name*`)` as described in *Inside Macintosh: Files*. An `FSSpec` can point to a non-existing file, as long as the folder containing the file exists. Under MacPython the same is true for a pathname, but not under UNIX-Python because of the way pathnames and FSRefs works. See Apple's documentation for details.

A description of aliases and the Standard File package can also be found there.

FSSpec(*file*)

Create an `FSSpec` object for the specified file.

`RawFSSpec`(*data*)
Create an `FSSpec` object given the raw data for the C structure for the `FSSpec` as a string. This is mainly useful if you have obtained an `FSSpec` structure over a network.

`RawAlias`(*data*)
Create an `Alias` object given the raw data for the C structure for the alias as a string. This is mainly useful if you have obtained an `FSSpec` structure over a network.

`FInfo`()
Create a zero-filled `FInfo` object.

`ResolveAliasFile`(*file*)
Resolve an alias file. Returns a 3-tuple (*fsspec*, *isfolder*, *aliased*) where *fsspec* is the resulting `FSSpec` object, *isfolder* is true if *fsspec* points to a folder and *aliased* is true if the file was an alias in the first place (otherwise the `FSSpec` object for the file itself is returned).

`StandardGetFile`([*type, ...*])
Present the user with a standard "open input file" dialog. Optionally, you can pass up to four 4-character file types to limit the files the user can choose from. The function returns an `FSSpec` object and a flag indicating that the user completed the dialog without cancelling.

`PromptGetFile`(*prompt*[*, type, ...*])
Similar to `StandardGetFile()` but allows you to specify a prompt which will be displayed at the top of the dialog.

`StandardPutFile`(*prompt*[*, default*])
Present the user with a standard "open output file" dialog. *prompt* is the prompt string, and the optional *default* argument initializes the output file name. The function returns an `FSSpec` object and a flag indicating that the user completed the dialog without cancelling.

`GetDirectory`([*prompt*])
Present the user with a non-standard "select a directory" dialog. You have to first open the directory before clicking on the "select current directory" button. *prompt* is the prompt string which will be displayed at the top of the dialog. Return an `FSSpec` object and a success-indicator.

`SetFolder`([*fsspec*])
Set the folder that is initially presented to the user when one of the file selection dialogs is presented. *fsspec* should point to a file in the folder, not the folder itself (the file need not exist, though). If no argument is passed the folder will be set to the current directory, i.e. what `os.getcwd()` returns.

Note that starting with System 7.5 the user can change Standard File behaviour with the "general controls" control panel, thereby making this call inoperative.

`FindFolder`(*where, which, create*)
Locates one of the "special" folders that Mac OS knows about, such as the trash or the Preferences folder. *where* is the disk to search, *which* is the 4-character string specifying which folder to locate. Setting *create* causes the folder to be created if it does not exist. Returns a (*vrefnum*, *dirid*) tuple.

The constants for *where* and *which* can be obtained from the standard module *Carbon.Folders*.

`NewAliasMinimalFromFullPath`(*pathname*)
Return a minimal `alias` object that points to the given file, which must be specified as a full pathname. This is the only way to create an `Alias` pointing to a non-existing file.

`FindApplication`(*creator*)
Locate the application with 4-character creator code *creator*. The function returns an `FSSpec` object pointing to the application.

2.2.1 FSSpec Objects

data
: The raw data from the FSSpec object, suitable for passing to other applications, for instance.

as_pathname()
: Return the full pathname of the file described by the `FSSpec` object.

as_tuple()
: Return the `(`*wdRefNum*`,` *parID*`,` *name*`)` tuple of the file described by the `FSSpec` object.

NewAlias([*file*])
: Create an Alias object pointing to the file described by this FSSpec. If the optional *file* parameter is present the alias will be relative to that file, otherwise it will be absolute.

NewAliasMinimal()
: Create a minimal alias pointing to this file.

GetCreatorType()
: Return the 4-character creator and type of the file.

SetCreatorType(*creator, type*)
: Set the 4-character creator and type of the file.

GetFInfo()
: Return a `FInfo` object describing the finder info for the file.

SetFInfo(*finfo*)
: Set the finder info for the file to the values given as *finfo* (an `FInfo` object).

GetDates()
: Return a tuple with three floating point values representing the creation date, modification date and backup date of the file.

SetDates(*crdate, moddate, backupdate*)
: Set the creation, modification and backup date of the file. The values are in the standard floating point format used for times throughout Python.

2.2.2 Alias Objects

data
: The raw data for the Alias record, suitable for storing in a resource or transmitting to other programs.

Resolve([*file*])
: Resolve the alias. If the alias was created as a relative alias you should pass the file relative to which it is. Return the FSSpec for the file pointed to and a flag indicating whether the `Alias` object itself was modified during the search process. If the file does not exist but the path leading up to it does exist a valid fsspec is returned.

GetInfo(*num*)
: An interface to the C routine `GetAliasInfo()`.

Update(*file*[, *file2*])
: Update the alias to point to the *file* given. If *file2* is present a relative alias will be created.

Note that it is currently not possible to directly manipulate a resource as an `Alias` object. Hence, after calling `Update()` or after `Resolve()` indicates that the alias has changed the Python program is responsible for getting the `data` value from the `Alias` object and modifying the resource.

2.2.3 FInfo Objects

See *Inside Macintosh: Files* for a complete description of what the various fields mean.

`Creator`
The 4-character creator code of the file.

`Type`
The 4-character type code of the file.

`Flags`
The finder flags for the file as 16-bit integer. The bit values in *Flags* are defined in standard module `MACFS`.

`Location`
A Point giving the position of the file's icon in its folder.

`Fldr`
The folder the file is in (as an integer).

2.3 `ic` — Access to Internet Config

This module provides access to various internet-related preferences set through **System Preferences** or the **Finder**.

There is a low-level companion module `icglue` which provides the basic Internet Config access functionality. This low-level module is not documented, but the docstrings of the routines document the parameters and the routine names are the same as for the Pascal or C API to Internet Config, so the standard IC programmers' documentation can be used if this module is needed.

The `ic` module defines the `error` exception and symbolic names for all error codes Internet Config can produce; see the source for details.

exception `error`
Exception raised on errors in the `ic` module.

The `ic` module defines the following class and function:

class `IC`(*[signature[, ic]]*)
Create an Internet Config object. The signature is a 4-character creator code of the current application (default `'Pyth'`) which may influence some of ICs settings. The optional *ic* argument is a low-level `icglue.icinstance` created beforehand, this may be useful if you want to get preferences from a different config file, etc.

`launchurl`(*url[, hint]*)
`parseurl`(*data[, start[, end[, hint]]]*)
`mapfile`(*file*)
`maptypecreator`(*type, creator[, filename]*)
`settypecreator`(*file*)
These functions are "shortcuts" to the methods of the same name, described below.

2.3.1 IC Objects

`IC` objects have a mapping interface, hence to obtain the mail address you simply get *ic*`['MailAddress']`. Assignment also works, and changes the option in the configuration file.

The module knows about various datatypes, and converts the internal IC representation to a "logical" Python data structure. Running the `ic` module standalone will run a test program that lists all keys and values in your IC database, this will have to serve as documentation.

If the module does not know how to represent the data it returns an instance of the `ICOpaqueData` type, with the raw data in its `data` attribute. Objects of this type are also acceptable values for assignment.

Besides the dictionary interface, `IC` objects have the following methods:

launchurl(*url*[, *hint*])
: Parse the given URL, launch the correct application and pass it the URL. The optional *hint* can be a scheme name such as `'mailto:'`, in which case incomplete URLs are completed with this scheme. If *hint* is not provided, incomplete URLs are invalid.

parseurl(*data*[, *start*[, *end*[, *hint*]]])
: Find an URL somewhere in *data* and return start position, end position and the URL. The optional *start* and *end* can be used to limit the search, so for instance if a user clicks in a long text field you can pass the whole text field and the click-position in *start* and this routine will return the whole URL in which the user clicked. As above, *hint* is an optional scheme used to complete incomplete URLs.

mapfile(*file*)
: Return the mapping entry for the given *file*, which can be passed as either a filename or an `FSSpec()` result, and which need not exist.

  The mapping entry is returned as a tuple (*version*, *type*, *creator*, *postcreator*, *flags*, *extension*, *appname*, *postappname*, *mimetype*, *entryname*), where *version* is the entry version number, *type* is the 4-character filetype, *creator* is the 4-character creator type, *postcreator* is the 4-character creator code of an optional application to post-process the file after downloading, *flags* are various bits specifying whether to transfer in binary or ascii and such, *extension* is the filename extension for this file type, *appname* is the printable name of the application to which this file belongs, *postappname* is the name of the postprocessing application, *mimetype* is the MIME type of this file and *entryname* is the name of this entry.

maptypecreator(*type, creator*[, *filename*])
: Return the mapping entry for files with given 4-character *type* and *creator* codes. The optional *filename* may be specified to further help finding the correct entry (if the creator code is `'????'`, for instance).

  The mapping entry is returned in the same format as for *mapfile*.

settypecreator(*file*)
: Given an existing *file*, specified either as a filename or as an `FSSpec()` result, set its creator and type correctly based on its extension. The finder is told about the change, so the finder icon will be updated quickly.

2.4 `MacOS` — Access to Mac OS interpreter features

This module provides access to MacOS specific functionality in the Python interpreter, such as how the interpreter eventloop functions and the like. Use with care.

Note the capitalization of the module name; this is a historical artifact.

runtimemodel
: Always `'macho'`, from Python 2.4 on. In earlier versions of Python the value could also be `'ppc'` for the classic Mac OS 8 runtime model or `'carbon'` for the Mac OS 9 runtime model.

linkmodel
: The way the interpreter has been linked. As extension modules may be incompatible between linking models, packages could use this information to give more decent error messages. The value is one of `'static'` for a statically linked Python, `'framework'` for Python in a Mac OS X framework, `'shared'` for Python in a standard UNIX shared library. Older Pythons could also have the value `'cfm'` for Mac OS 9-compatible Python.

exception Error
: This exception is raised on MacOS generated errors, either from functions in this module or from other mac-specific modules like the toolbox interfaces. The arguments are the integer error code (the `OSErr` value) and

a textual description of the error code. Symbolic names for all known error codes are defined in the standard module `macerrors`.

`GetErrorString`(*errno*)
Return the textual description of MacOS error code *errno*.

`DebugStr`(*message* [, *object*])
On Mac OS X the string is simply printed to stderr (on older Mac OS systems more elaborate functionality was available), but it provides a convenient location to attach a breakpoint in a low-level debugger like **gdb**.

`SysBeep`()
Ring the bell.

`GetTicks`()
Get the number of clock ticks (1/60th of a second) since system boot.

`GetCreatorAndType`(*file*)
Return the file creator and file type as two four-character strings. The *file* parameter can be a pathname or an `FSSpec` or `FSRef` object.

`SetCreatorAndType`(*file, creator, type*)
Set the file creator and file type. The *file* parameter can be a pathname or an `FSSpec` or `FSRef` object. *creator* and *type* must be four character strings.

`openrf`(*name* [, *mode*])
Open the resource fork of a file. Arguments are the same as for the built-in function `open()`. The object returned has file-like semantics, but it is not a Python file object, so there may be subtle differences.

`WMAvailable`()
Checks whether the current process has access to the window manager. The method will return `False` if the window manager is not available, for instance when running on Mac OS X Server or when logged in via ssh, or when the current interpreter is not running from a fullblown application bundle. A script runs from an application bundle either when it has been started with **pythonw** instead of **python** or when running as an applet.

2.5 `macostools` — Convenience routines for file manipulation

This module contains some convenience routines for file-manipulation on the Macintosh. All file parameters can be specified as pathnames, `FSRef` or `FSSpec` objects. This module expects a filesystem which supports forked files, so it should not be used on UFS partitions.

The `macostools` module defines the following functions:

`copy`(*src, dst*[, *createpath*[, *copytimes*]])
Copy file *src* to *dst*. If *createpath* is non-zero the folders leading to *dst* are created if necessary. The method copies data and resource fork and some finder information (creator, type, flags) and optionally the creation, modification and backup times (default is to copy them). Custom icons, comments and icon position are not copied.

`copytree`(*src, dst*)
Recursively copy a file tree from *src* to *dst*, creating folders as needed. *src* and *dst* should be specified as pathnames.

`mkalias`(*src, dst*)
Create a finder alias *dst* pointing to *src*.

`touched`(*dst*)
Tell the finder that some bits of finder-information such as creator or type for file *dst* has changed. The file can be specified by pathname or fsspec. This call should tell the finder to redraw the files icon.

BUFSIZ
: The buffer size for `copy`, default 1 megabyte.

Note that the process of creating finder aliases is not specified in the Apple documentation. Hence, aliases created with `mkalias()` could conceivably have incompatible behaviour in some cases.

2.6 `findertools` — The **finder**'s Apple Events interface

This module contains routines that give Python programs access to some functionality provided by the finder. They are implemented as wrappers around the AppleEvent interface to the finder.

All file and folder parameters can be specified either as full pathnames, or as `FSRef` or `FSSpec` objects.

The `findertools` module defines the following functions:

launch(*file*)
: Tell the finder to launch *file*. What launching means depends on the file: applications are started, folders are opened and documents are opened in the correct application.

Print(*file*)
: Tell the finder to print a file. The behaviour is identical to selecting the file and using the print command in the finder's file menu.

copy(*file, destdir*)
: Tell the finder to copy a file or folder *file* to folder *destdir*. The function returns an `Alias` object pointing to the new file.

move(*file, destdir*)
: Tell the finder to move a file or folder *file* to folder *destdir*. The function returns an `Alias` object pointing to the new file.

sleep()
: Tell the finder to put the Macintosh to sleep, if your machine supports it.

restart()
: Tell the finder to perform an orderly restart of the machine.

shutdown()
: Tell the finder to perform an orderly shutdown of the machine.

2.7 `EasyDialogs` — Basic Macintosh dialogs

The `EasyDialogs` module contains some simple dialogs for the Macintosh. All routines take an optional resource ID parameter *id* with which one can override the `DLOG` resource used for the dialog, provided that the dialog items correspond (both type and item number) to those in the default `DLOG` resource. See source code for details.

The `EasyDialogs` module defines the following functions:

Message(*str*[, *id*[, *ok*]])
: Displays a modal dialog with the message text *str*, which should be at most 255 characters long. The button text defaults to "OK", but is set to the string argument *ok* if the latter is supplied. Control is returned when the user clicks the "OK" button.

AskString(*prompt*[, *default*[, *id*[, *ok*[, *cancel*]]]])
: Asks the user to input a string value via a modal dialog. *prompt* is the prompt message, and the optional *default* supplies the initial value for the string (otherwise `""` is used). The text of the "OK" and "Cancel" buttons can be changed with the *ok* and *cancel* arguments. All strings can be at most 255 bytes long. `AskString()` returns the string entered or `None` in case the user cancelled.

AskPassword(*prompt*[, *default*[, *id*[, *ok*[, *cancel*]]]])
: Asks the user to input a string value via a modal dialog. Like `AskString()`, but with the text shown as bullets. The arguments have the same meaning as for `AskString()`.

AskYesNoCancel(*question*[, *default*[, *yes*[, *no*[, *cancel*[, *id*]]]]])
: Presents a dialog with prompt *question* and three buttons labelled "Yes", "No", and "Cancel". Returns `1` for "Yes", `0` for "No" and `-1` for "Cancel". The value of *default* (or `0` if *default* is not supplied) is returned when the `RETURN` key is pressed. The text of the buttons can be changed with the *yes*, *no*, and *cancel* arguments; to prevent a button from appearing, supply `""` for the corresponding argument.

ProgressBar([*title*[, *maxval*[, *label*[, *id*]]]])
: Displays a modeless progress-bar dialog. This is the constructor for the `ProgressBar` class described below. *title* is the text string displayed (default "Working..."), *maxval* is the value at which progress is complete (default `0`, indicating that an indeterminate amount of work remains to be done), and *label* is the text that is displayed above the progress bar itself.

GetArgv([*optionlist*[*commandlist*[, *addoldfile*[, *addnewfile*[, *addfolder*[, *id*]]]]]])
: Displays a dialog which aids the user in constructing a command-line argument list. Returns the list in `sys.argv` format, suitable for passing as an argument to `getopt.getopt()`. *addoldfile*, *addnewfile*, and *addfolder* are boolean arguments. When nonzero, they enable the user to insert into the command line paths to an existing file, a (possibly) not-yet-existent file, and a folder, respectively. (Note: Option arguments must appear in the command line before file and folder arguments in order to be recognized by `getopt.getopt()`.) Arguments containing spaces can be specified by enclosing them within single or double quotes. A `SystemExit` exception is raised if the user presses the "Cancel" button.

optionlist is a list that determines a popup menu from which the allowed options are selected. Its items can take one of two forms: *optstr* or `(`*optstr*`, `*descr*`)`. When present, *descr* is a short descriptive string that is displayed in the dialog while this option is selected in the popup menu. The correspondence between *optstr*s and command-line arguments is:

***optstr* format**	**Command-line format**
`x`	**-x** (short option)
`x:` or `x=`	**-x** (short option with value)
`xyz`	**--xyz** (long option)
`xyz:` or `xyz=`	**--xyz** (long option with value)

commandlist is a list of items of the form *cmdstr* or `(`*cmdstr*`, `*descr*`)`, where *descr* is as above. The *cmdstr*s will appear in a popup menu. When chosen, the text of *cmdstr* will be appended to the command line as is, except that a trailing '`:`' or '`=`' (if present) will be trimmed off.

New in version 2.0.

AskFileForOpen([*message*] [, *typeList*] [, *defaultLocation*] [, *defaultOptionFlags*] [, *location*] [, *clientName*] [, *windowTitle*] [, *actionButtonLabel*] [, *cancelButtonLabel*] [, *preferenceKey*] [, *popupExtension*] [, *eventProc*] [, *previewProc*] [, *filterProc*] [, *wanted*])
: Post a dialog asking the user for a file to open, and return the file selected or `None` if the user cancelled. *message* is a text message to display, *typeList* is a list of 4-char filetypes allowable, *defaultLocation* is the pathname, `FSSpec` or `FSRef` of the folder to show initially, *location* is the `(x, y)` position on the screen where the dialog is shown, *actionButtonLabel* is a string to show instead of "Open" in the OK button, *cancelButtonLabel* is a string to show instead of "Cancel" in the cancel button, *wanted* is the type of value wanted as a return: `str`, `unicode`, `FSSpec`, `FSRef` and subtypes thereof are acceptable.

For a description of the other arguments please see the Apple Navigation Services documentation and the `EasyDialogs` source code.

AskFileForSave([*message*] [, *savedFileName*] [, *defaultLocation*] [, *defaultOptionFlags*] [, *location*] [, *clientName*] [, *windowTitle*] [, *actionButtonLabel*] [, *cancelButtonLabel*] [, *preferenceKey*] [, *popupExtension*] [, *fileType*] [, *fileCreator*] [, *eventProc*] [, *wanted*])
: Post a dialog asking the user for a file to save to, and return the file selected or `None` if the user cancelled. *savedFileName* is the default for the file name to save to (the return value). See `AskFileForOpen()` for a description of the other arguments.

AskFolder([*message*] [, *defaultLocation*] [, *defaultOptionFlags*] [, *location*] [, *clientName*] [, *windowTitle*] [, *actionButtonLabel*] [, *cancelButtonLabel*] [, *preferenceKey*] [, *popupExtension*] [, *eventProc*] [, *filterProc*] [, *wanted*])

Post a dialog asking the user to select a folder, and return the folder selected or `None` if the user cancelled. See `AskFileForOpen()` for a description of the arguments.

See Also:

Navigation Services Reference
(http://developer.apple.com/documentation/Carbon/Reference/Navigation_Services_Ref/)

Programmer's reference documentation for the Navigation Services, a part of the Carbon framework.

2.7.1 ProgressBar Objects

`ProgressBar` objects provide support for modeless progress-bar dialogs. Both determinate (thermometer style) and indeterminate (barber-pole style) progress bars are supported. The bar will be determinate if its maximum value is greater than zero; otherwise it will be indeterminate. Changed in version 2.2: Support for indeterminate-style progress bars was added.

The dialog is displayed immediately after creation. If the dialog's "Cancel" button is pressed, or if `Cmd-.` or `ESC` is typed, the dialog window is hidden and `KeyboardInterrupt` is raised (but note that this response does not occur until the progress bar is next updated, typically via a call to `inc()` or `set()`). Otherwise, the bar remains visible until the `ProgressBar` object is discarded.

`ProgressBar` objects possess the following attributes and methods:

curval

The current value (of type integer or long integer) of the progress bar. The normal access methods coerce `curval` between `0` and `maxval`. This attribute should not be altered directly.

maxval

The maximum value (of type integer or long integer) of the progress bar; the progress bar (thermometer style) is full when `curval` equals `maxval`. If `maxval` is `0`, the bar will be indeterminate (barber-pole). This attribute should not be altered directly.

title([*newstr*])

Sets the text in the title bar of the progress dialog to *newstr*.

label([*newstr*])

Sets the text in the progress box of the progress dialog to *newstr*.

set(*value*[, *max*])

Sets the progress bar's `curval` to *value*, and also `maxval` to *max* if the latter is provided. *value* is first coerced between 0 and `maxval`. The thermometer bar is updated to reflect the changes, including a change from indeterminate to determinate or vice versa.

inc([*n*])

Increments the progress bar's `curval` by *n*, or by `1` if *n* is not provided. (Note that *n* may be negative, in which case the effect is a decrement.) The progress bar is updated to reflect the change. If the bar is indeterminate, this causes one "spin" of the barber pole. The resulting `curval` is coerced between 0 and `maxval` if incrementing causes it to fall outside this range.

2.8 `FrameWork` — Interactive application framework

The `FrameWork` module contains classes that together provide a framework for an interactive Macintosh application. The programmer builds an application by creating subclasses that override various methods of the bases classes, thereby implementing the functionality wanted. Overriding functionality can often be done on various different levels, i.e. to handle clicks in a single dialog window in a non-standard way it is not necessary to override the complete event handling.

Work on the `FrameWork` has pretty much stopped, now that `PyObjC` is available for full Cocoa access from Python, and the documentation describes only the most important functionality, and not in the most logical manner at that. Examine the source or the examples for more details. The following are some comments posted on the MacPython newsgroup about the strengths and limitations of `FrameWork`:

> The strong point of `FrameWork` is that it allows you to break into the control-flow at many different places. `W`, for instance, uses a different way to enable/disable menus and that plugs right in leaving the rest intact. The weak points of `FrameWork` are that it has no abstract command interface (but that shouldn't be difficult), that its dialog support is minimal and that its control/toolbar support is non-existent.

The `FrameWork` module defines the following functions:

`Application``()`
: An object representing the complete application. See below for a description of the methods. The default `__init__()` routine creates an empty window dictionary and a menu bar with an apple menu.

`MenuBar``()`
: An object representing the menubar. This object is usually not created by the user.

`Menu``(`*bar, title*[, *after*]`)`
: An object representing a menu. Upon creation you pass the `MenuBar` the menu appears in, the *title* string and a position (1-based) *after* where the menu should appear (default: at the end).

`MenuItem``(`*menu, title*[, *shortcut, callback*]`)`
: Create a menu item object. The arguments are the menu to create, the item title string and optionally the keyboard shortcut and a callback routine. The callback is called with the arguments menu-id, item number within menu (1-based), current front window and the event record.

  Instead of a callable object the callback can also be a string. In this case menu selection causes the lookup of a method in the topmost window and the application. The method name is the callback string with `'domenu_'` prepended.

  Calling the `MenuBar` `fixmenudimstate()` method sets the correct dimming for all menu items based on the current front window.

`Separator``(`*menu*`)`
: Add a separator to the end of a menu.

`SubMenu``(`*menu, label*`)`
: Create a submenu named *label* under menu *menu*. The menu object is returned.

`Window``(`*parent*`)`
: Creates a (modeless) window. *Parent* is the application object to which the window belongs. The window is not displayed until later.

`DialogWindow``(`*parent*`)`
: Creates a modeless dialog window.

`windowbounds``(`*width, height*`)`
: Return a `(`*left*`, `*top*`, `*right*`, `*bottom*`)` tuple suitable for creation of a window of given width and height. The window will be staggered with respect to previous windows, and an attempt is made to keep the whole window on-screen. However, the window will however always be the exact size given, so parts may be offscreen.

`setwatchcursor``()`
: Set the mouse cursor to a watch.

`setarrowcursor``()`
: Set the mouse cursor to an arrow.

2.8.1 Application Objects

Application objects have the following methods, among others:

makeusermenus()
Override this method if you need menus in your application. Append the menus to the attribute `menubar`.

getabouttext()
Override this method to return a text string describing your application. Alternatively, override the `do_-about()` method for more elaborate "about" messages.

mainloop([*mask*[, *wait*]])
This routine is the main event loop, call it to set your application rolling. *Mask* is the mask of events you want to handle, *wait* is the number of ticks you want to leave to other concurrent application (default 0, which is probably not a good idea). While raising *self* to exit the mainloop is still supported it is not recommended: call `self._quit()` instead.

The event loop is split into many small parts, each of which can be overridden. The default methods take care of dispatching events to windows and dialogs, handling drags and resizes, Apple Events, events for non-FrameWork windows, etc.

In general, all event handlers should return `1` if the event is fully handled and `0` otherwise (because the front window was not a FrameWork window, for instance). This is needed so that update events and such can be passed on to other windows like the Sioux console window. Calling `MacOS.HandleEvent()` is not allowed within *our_dispatch* or its callees, since this may result in an infinite loop if the code is called through the Python inner-loop event handler.

asyncevents(*onoff*)
Call this method with a nonzero parameter to enable asynchronous event handling. This will tell the inner interpreter loop to call the application event handler *async_dispatch* whenever events are available. This will cause FrameWork window updates and the user interface to remain working during long computations, but will slow the interpreter down and may cause surprising results in non-reentrant code (such as FrameWork itself). By default *async_dispatch* will immediately call *our_dispatch* but you may override this to handle only certain events asynchronously. Events you do not handle will be passed to Sioux and such.

The old on/off value is returned.

_quit()
Terminate the running `mainloop()` call at the next convenient moment.

do_char(*c, event*)
The user typed character *c*. The complete details of the event can be found in the *event* structure. This method can also be provided in a `Window` object, which overrides the application-wide handler if the window is frontmost.

do_dialogevent(*event*)
Called early in the event loop to handle modeless dialog events. The default method simply dispatches the event to the relevant dialog (not through the `DialogWindow` object involved). Override if you need special handling of dialog events (keyboard shortcuts, etc).

idle(*event*)
Called by the main event loop when no events are available. The null-event is passed (so you can look at mouse position, etc).

2.8.2 Window Objects

Window objects have the following methods, among others:

open()
Override this method to open a window. Store the MacOS window-id in `self.wid` and call the `do_-`

`postopen()` method to register the window with the parent application.

close()
Override this method to do any special processing on window close. Call the `do_postclose()` method to cleanup the parent state.

do_postresize(*width, height, macoswindowid*)
Called after the window is resized. Override if more needs to be done than calling `InvalRect`.

do_contentclick(*local, modifiers, event*)
The user clicked in the content part of a window. The arguments are the coordinates (window-relative), the key modifiers and the raw event.

do_update(*macoswindowid, event*)
An update event for the window was received. Redraw the window.

do_activate(*activate, event*)
The window was activated (*activate* `== 1`) or deactivated (*activate* `== 0`). Handle things like focus highlighting, etc.

2.8.3 ControlsWindow Object

ControlsWindow objects have the following methods besides those of `Window` objects:

do_controlhit(*window, control, pcode, event*)
Part *pcode* of control *control* was hit by the user. Tracking and such has already been taken care of.

2.8.4 ScrolledWindow Object

ScrolledWindow objects are ControlsWindow objects with the following extra methods:

scrollbars([*wantx*[, *wanty*]])
Create (or destroy) horizontal and vertical scrollbars. The arguments specify which you want (default: both). The scrollbars always have minimum `0` and maximum `32767`.

getscrollbarvalues()
You must supply this method. It should return a tuple `(`*x*`, `*y*`)` giving the current position of the scrollbars (between `0` and `32767`). You can return `None` for either to indicate the whole document is visible in that direction.

updatescrollbars()
Call this method when the document has changed. It will call `getscrollbarvalues()` and update the scrollbars.

scrollbar_callback(*which, what, value*)
Supplied by you and called after user interaction. *which* will be `'x'` or `'y'`, *what* will be `'-'`, `'--'`, `'set'`, `'++'` or `'+'`. For `'set'`, *value* will contain the new scrollbar position.

scalebarvalues(*absmin, absmax, curmin, curmax*)
Auxiliary method to help you calculate values to return from `getscrollbarvalues()`. You pass document minimum and maximum value and topmost (leftmost) and bottommost (rightmost) visible values and it returns the correct number or `None`.

do_activate(*onoff, event*)
Takes care of dimming/highlighting scrollbars when a window becomes frontmost. If you override this method, call this one at the end of your method.

do_postresize(*width, height, window*)
Moves scrollbars to the correct position. Call this method initially if you override it.

do_controlhit(*window, control, pcode, event*)
Handles scrollbar interaction. If you override it call this method first, a nonzero return value indicates the hit was in the scrollbars and has been handled.

2.8.5 DialogWindow Objects

DialogWindow objects have the following methods besides those of `Window` objects:

open(*resid*)
Create the dialog window, from the DLOG resource with id *resid*. The dialog object is stored in `self.wid`.

do_itemhit(*item, event*)
Item number *item* was hit. You are responsible for redrawing toggle buttons, etc.

2.9 `autoGIL` — Global Interpreter Lock handling in event loops

The `autoGIL` module provides a function `installAutoGIL` that automatically locks and unlocks Python's Global Interpreter Lock when running an event loop.

exception AutoGILError
Raised if the observer callback cannot be installed, for example because the current thread does not have a run loop.

installAutoGIL()
Install an observer callback in the event loop (CFRunLoop) for the current thread, that will lock and unlock the Global Interpreter Lock (GIL) at appropriate times, allowing other Python threads to run while the event loop is idle.

Availability: OSX 10.1 or later.

CHAPTER

THREE

MacPython OSA Modules

This chapter describes the current implementation of the Open Scripting Architecure (OSA, also commonly referred to as AppleScript) for Python, allowing you to control scriptable applications from your Python program, and with a fairly pythonic interface. Development on this set of modules has stopped, and a replacement is expected for Python 2.5.

For a description of the various components of AppleScript and OSA, and to get an understanding of the architecture and terminology, you should read Apple's documentation. The "Applescript Language Guide" explains the conceptual model and the terminology, and documents the standard suite. The "Open Scripting Architecture" document explains how to use OSA from an application programmers point of view. In the Apple Help Viewer these books are located in the Developer Documentation, Core Technologies section.

As an example of scripting an application, the following piece of AppleScript will get the name of the frontmost **Finder** window and print it:

```
tell application "Finder"
    get name of window 1
end tell
```

In Python, the following code fragment will do the same:

```
import Finder

f = Finder.Finder()
print f.get(f.window(1).name)
```

As distributed the Python library includes packages that implement the standard suites, plus packages that interface to a small number of common applications.

To send AppleEvents to an application you must first create the Python package interfacing to the terminology of the application (what **Script Editor** calls the "Dictionary"). This can be done from within the **PythonIDE** or by running the 'gensuitemodule.py' module as a standalone program from the command line.

The generated output is a package with a number of modules, one for every suite used in the program plus an `__init__` module to glue it all together. The Python inheritance graph follows the AppleScript inheritance graph, so if a program's dictionary specifies that it includes support for the Standard Suite, but extends one or two verbs with extra arguments then the output suite will contain a module `Standard_Suite` that imports and re-exports everything from `StdSuites.Standard_Suite` but overrides the methods that have extra functionality. The output of `gensuitemodule` is pretty readable, and contains the documentation that was in the original AppleScript dictionary in Python docstrings, so reading it is a good source of documentation.

The output package implements a main class with the same name as the package which contains all the AppleScript verbs as methods, with the direct object as the first argument and all optional parameters as keyword arguments. AppleScript classes are also implemented as Python classes, as are comparisons and all the other thingies.

The main Python class implementing the verbs also allows access to the properties and elements declared in the AppleScript class "application". In the current release that is as far as the object orientation goes, so in the example above we need to use `f.get(f.window(1).name)` instead of the more Pythonic `f.window(1).name.get()`.

If an AppleScript identifier is not a Python identifier the name is mangled according to a small number of rules:

- spaces are replaced with underscores
- other non-alphanumeric characters are replaced with `_xx_` where `xx` is the hexadecimal character value
- any Python reserved word gets an underscore appended

Python also has support for creating scriptable applications in Python, but The following modules are relevant to MacPython AppleScript support:

gensuitemodule	Create a stub package from an OSA dictionary
aetools	Basic support for sending Apple Events
aepack	Conversion between Python variables and AppleEvent data containers.
aetypes	Python representation of the Apple Event Object Model.
MiniAEFrame	Support to act as an Open Scripting Architecture (OSA) server ("Apple Events").

In addition, support modules have been pre-generated for `Finder`, `Terminal`, `Explorer`, `Netscape`, `CodeWarrior`, `SystemEvents` and `StdSuites`.

3.1 `gensuitemodule` — Generate OSA stub packages

The `gensuitemodule` module creates a Python package implementing stub code for the AppleScript suites that are implemented by a specific application, according to its AppleScript dictionary.

It is usually invoked by the user through the **PythonIDE**, but it can also be run as a script from the command line (pass **--help** for help on the options) or imported from Python code. For an example of its use see 'Mac/scripts/genallsuites.py' in a source distribution, which generates the stub packages that are included in the standard library.

It defines the following public functions:

`is_scriptable`(*application*)

Returns true if `application`, which should be passed as a pathname, appears to be scriptable. Take the return value with a grain of salt: **Internet Explorer** appears not to be scriptable but definitely is.

`processfile`(*application*[*, output, basepkgname, edit_modnames, creatorsignature, dump, verbose*])

Create a stub package for `application`, which should be passed as a full pathname. For a '.app' bundle this is the pathname to the bundle, not to the executable inside the bundle; for an unbundled CFM application you pass the filename of the application binary.

This function asks the application for its OSA terminology resources, decodes these resources and uses the resultant data to create the Python code for the package implementing the client stubs.

`output` is the pathname where the resulting package is stored, if not specified a standard "save file as" dialog is presented to the user. `basepkgname` is the base package on which this package will build, and defaults to `StdSuites`. Only when generating `StdSuites` itself do you need to specify this. `edit_modnames` is a dictionary that can be used to change modulenames that are too ugly after name mangling. `creator_signature` can be used to override the 4-char creator code, which is normally obtained from the 'PkgInfo' file in the package or from the CFM file creator signature. When `dump` is given it should refer to a file object, and `processfile` will stop after decoding the resources and dump the Python representation of the terminology resources to this file. `verbose` should also be a file object, and specifying it will cause `processfile` to tell you what it is doing.

`processfile_fromresource`(*application*[*, output, basepkgname, edit_modnames, creatorsignature, dump, verbose*])
This function does the same as `processfile`, except that it uses a different method to get the terminology resources. It opens `application` as a resource file and reads all `"aete"` and `"aeut"` resources from this file.

3.2 `aetools` — OSA client support

The `aetools` module contains the basic functionality on which Python AppleScript client support is built. It also imports and re-exports the core functionality of the `aetypes` and `aepack` modules. The stub packages generated by `gensuitemodule` import the relevant portions of `aetools`, so usually you do not need to import it yourself. The exception to this is when you cannot use a generated suite package and need lower-level access to scripting.

The `aetools` module itself uses the AppleEvent support provided by the `Carbon.AE` module. This has one drawback: you need access to the window manager, see section 1.1.2 for details. This restriction may be lifted in future releases.

The `aetools` module defines the following functions:

`packevent`(*ae, parameters, attributes*)
Stores parameters and attributes in a pre-created `Carbon.AE.AEDesc` object. `parameters` and `attributes` are dictionaries mapping 4-character OSA parameter keys to Python objects. The objects are packed using `aepack.pack()`.

`unpackevent`(*ae*[*, formodulename*])
Recursively unpacks a `Carbon.AE.AEDesc` event to Python objects. The function returns the parameter dictionary and the attribute dictionary. The `formodulename` argument is used by generated stub packages to control where AppleScript classes are looked up.

`keysubst`(*arguments, keydict*)
Converts a Python keyword argument dictionary `arguments` to the format required by `packevent` by replacing the keys, which are Python identifiers, by the four-character OSA keys according to the mapping specified in `keydict`. Used by the generated suite packages.

`enumsubst`(*arguments, key, edict*)
If the `arguments` dictionary contains an entry for `key` convert the value for that entry according to dictionary `edict`. This converts human-readable Python enumeration names to the OSA 4-character codes. Used by the generated suite packages.

The `aetools` module defines the following class:

class `TalkTo`([*signature=None, start=0, timeout=0*])
Base class for the proxy used to talk to an application. `signature` overrides the class attribute `_signature` (which is usually set by subclasses) and is the 4-char creator code defining the application to talk to. `start` can be set to true to enable running the application on class instantiation. `timeout` can be specified to change the default timeout used while waiting for an AppleEvent reply.

`_start`()
Test whether the application is running, and attempt to start it if not.

`send`(*code, subcode*[*, parameters, attributes*])
Create the AppleEvent `Carbon.AE.AEDesc` for the verb with the OSA designation `code, subcode` (which are the usual 4-character strings), pack the `parameters` and `attributes` into it, send it to the target application, wait for the reply, unpack the reply with `unpackevent` and return the reply appleevent, the unpacked return values as a dictionary and the return attributes.

3.3 `aepack` — Conversion between Python variables and AppleEvent data containers

The `aepack` module defines functions for converting (packing) Python variables to AppleEvent descriptors and back (unpacking). Within Python the AppleEvent descriptor is handled by Python objects of built-in type `AEDesc`, defined in module `Carbon.AE`.

The `aepack` module defines the following functions:

`pack`(*x*[, *forcetype*])

Returns an `AEDesc` object containing a conversion of Python value `x`. If *forcetype* is provided it specifies the descriptor type of the result. Otherwise, a default mapping of Python types to Apple Event descriptor types is used, as follows:

Python type	descriptor type
`FSSpec`	typeFSS
`FSRef`	typeFSRef
`Alias`	typeAlias
integer	typeLong (32 bit integer)
float	typeFloat (64 bit floating point)
string	typeText
unicode	typeUnicodeText
list	typeAEList
dictionary	typeAERecord
instance	*see below*

If *x* is a Python instance then this function attempts to call an `__aepack__()` method. This method should return an `AEDesc` object.

If the conversion *x* is not defined above, this function returns the Python string representation of a value (the repr() function) encoded as a text descriptor.

`unpack`(*x*[, *formodulename*])

x must be an object of type `AEDesc`. This function returns a Python object representation of the data in the Apple Event descriptor *x*. Simple AppleEvent data types (integer, text, float) are returned as their obvious Python counterparts. Apple Event lists are returned as Python lists, and the list elements are recursively unpacked. Object references (ex. `line 3 of document 1`) are returned as instances of `aetypes.ObjectSpecifier`, unless `formodulename` is specified. AppleEvent descriptors with descriptor type typeFSS are returned as `FSSpec` objects. AppleEvent record descriptors are returned as Python dictionaries, with 4-character string keys and elements recursively unpacked.

The optional `formodulename` argument is used by the stub packages generated by `gensuitemodule`, and ensures that the OSA classes for object specifiers are looked up in the correct module. This ensures that if, say, the Finder returns an object specifier for a window you get an instance of `Finder.Window` and not a generic `aetypes.Window`. The former knows about all the properties and elements a window has in the Finder, while the latter knows no such things.

See Also:

Module `Carbon.AE` (section 4.1):
Built-in access to Apple Event Manager routines.

Module `aetypes` (section 3.4):
Python definitions of codes for Apple Event descriptor types.

Inside Macintosh: Interapplication Communication
(http://developer.apple.com/techpubs/mac/IAC/IAC-2.html)
Information about inter-process communications on the Macintosh.

3.4 `aetypes` — AppleEvent objects

The `aetypes` defines classes used to represent Apple Event data descriptors and Apple Event object specifiers.

Apple Event data is contained in descriptors, and these descriptors are typed. For many descriptors the Python representation is simply the corresponding Python type: `typeText` in OSA is a Python string, `typeFloat` is a float, etc. For OSA types that have no direct Python counterpart this module declares classes. Packing and unpacking instances of these classes is handled automatically by `aepack`.

An object specifier is essentially an address of an object implemented in a Apple Event server. An Apple Event specifier is used as the direct object for an Apple Event or as the argument of an optional parameter. The `aetypes` module contains the base classes for OSA classes and properties, which are used by the packages generated by `gensuitemodule` to populate the classes and properties in a given suite.

For reasons of backward compatibility, and for cases where you need to script an application for which you have not generated the stub package this module also contains object specifiers for a number of common OSA classes such as `Document`, `Window`, `Character`, etc.

The `AEObjects` module defines the following classes to represent Apple Event descriptor data:

class `Unknown`(*type, data*)
The representation of OSA descriptor data for which the `aepack` and `aetypes` modules have no support, i.e. anything that is not represented by the other classes here and that is not equivalent to a simple Python value.

class `Enum`(*enum*)
An enumeration value with the given 4-character string value.

class `InsertionLoc`(*of, pos*)
Position `pos` in object `of`.

class `Boolean`(*bool*)
A boolean.

class `StyledText`(*style, text*)
Text with style information (font, face, etc) included.

class `AEText`(*script, style, text*)
Text with script system and style information included.

class `IntlText`(*script, language, text*)
Text with script system and language information included.

class `IntlWritingCode`(*script, language*)
Script system and language information.

class `QDPoint`(*v, h*)
A quickdraw point.

class `QDRectangle`(*v0, h0, v1, h1*)
A quickdraw rectangle.

class `RGBColor`(*r, g, b*)
A color.

class `Type`(*type*)
An OSA type value with the given 4-character name.

class `Keyword`(*name*)
An OSA keyword with the given 4-character name.

class `Range`(*start, stop*)
A range.

class Ordinal(*abso*)

Non-numeric absolute positions, such as `"firs"`, first, or `"midd"`, middle.

class Logical(*logc, term*)

The logical expression of applying operator `logc` to `term`.

class Comparison(*obj1, relo, obj2*)

The comparison `relo` of `obj1` to `obj2`.

The following classes are used as base classes by the generated stub packages to represent AppleScript classes and properties in Python:

class ComponentItem(*which*[*, fr*])

Abstract baseclass for an OSA class. The subclass should set the class attribute `want` to the 4-character OSA class code. Instances of subclasses of this class are equivalent to AppleScript Object Specifiers. Upon instantiation you should pass a selector in `which`, and optionally a parent object in `fr`.

class NProperty(*fr*)

Abstract baseclass for an OSA property. The subclass should set the class attributes `want` and `which` to designate which property we are talking about. Instances of subclasses of this class are Object Specifiers.

class ObjectSpecifier(*want, form, seld*[*, fr*])

Base class of `ComponentItem` and `NProperty`, a general OSA Object Specifier. See the Apple Open Scripting Architecture documentation for the parameters. Note that this class is not abstract.

3.5 `MiniAEFrame` — Open Scripting Architecture server support

The module `MiniAEFrame` provides a framework for an application that can function as an Open Scripting Architecture (OSA) server, i.e. receive and process AppleEvents. It can be used in conjunction with `FrameWork` or standalone. As an example, it is used in **PythonCGISlave**.

The `MiniAEFrame` module defines the following classes:

class AEServer()

A class that handles AppleEvent dispatch. Your application should subclass this class together with either `MiniApplication` or `FrameWork.Application`. Your `__init__()` method should call the `__init__()` method for both classes.

class MiniApplication()

A class that is more or less compatible with `FrameWork.Application` but with less functionality. Its event loop supports the apple menu, command-dot and AppleEvents; other events are passed on to the Python interpreter and/or Sioux. Useful if your application wants to use `AEServer` but does not provide its own windows, etc.

3.5.1 AEServer Objects

installaehandler(*classe, type, callback*)

Installs an AppleEvent handler. *classe* and *type* are the four-character OSA Class and Type designators, `'****'` wildcards are allowed. When a matching AppleEvent is received the parameters are decoded and your callback is invoked.

callback(*_object, **kwargs*)

Your callback is called with the OSA Direct Object as first positional parameter. The other parameters are passed as keyword arguments, with the 4-character designator as name. Three extra keyword parameters are passed: `_class` and `_type` are the Class and Type designators and `_attributes` is a dictionary with the AppleEvent attributes.

The return value of your method is packed with `aetools.packevent()` and sent as reply.

Note that there are some serious problems with the current design. AppleEvents which have non-identifier 4-character designators for arguments are not implementable, and it is not possible to return an error to the originator. This will be addressed in a future release.

CHAPTER

FOUR

MacOS Toolbox Modules

There are a set of modules that provide interfaces to various MacOS toolboxes. If applicable the module will define a number of Python objects for the various structures declared by the toolbox, and operations will be implemented as methods of the object. Other operations will be implemented as functions in the module. Not all operations possible in C will also be possible in Python (callbacks are often a problem), and parameters will occasionally be different in Python (input and output buffers, especially). All methods and functions have a `__doc__` string describing their arguments and return values, and for additional description you are referred to *Inside Macintosh* or similar works.

These modules all live in a package called `Carbon`. Despite that name they are not all part of the Carbon framework: CF is really in the CoreFoundation framework and Qt is in the QuickTime framework. The normal use pattern is

```
from Carbon import AE
```

Warning! These modules are not yet documented. If you wish to contribute documentation of any of these modules, please get in touch with docs@python.org.

`Carbon.AE`	Interface to the Apple Events toolbox.
`Carbon.AH`	Interface to the Apple Help manager.
`Carbon.App`	Interface to the Appearance Manager.
`Carbon.CF`	Interface to the Core Foundation.
`Carbon.CG`	Interface to the Component Manager.
`Carbon.CarbonEvt`	Interface to the Carbon Event Manager.
`Carbon.Cm`	Interface to the Component Manager.
`Carbon.Ctl`	Interface to the Control Manager.
`Carbon.Dlg`	Interface to the Dialog Manager.
`Carbon.Evt`	Interface to the classic Event Manager.
`Carbon.Fm`	Interface to the Font Manager.
`Carbon.Folder`	Interface to the Folder Manager.
`Carbon.Help`	Interface to the Carbon Help Manager.
`Carbon.List`	Interface to the List Manager.
`Carbon.Menu`	Interface to the Menu Manager.
`Carbon.Mlte`	Interface to the MultiLingual Text Editor.
`Carbon.Qd`	Interface to the QuickDraw toolbox.
`Carbon.Qdoffs`	Interface to the QuickDraw Offscreen APIs.
`Carbon.Qt`	Interface to the QuickTime toolbox.
`Carbon.Res`	Interface to the Resource Manager and Handles.
`Carbon.Scrap`	Interface to the Carbon Scrap Manager.
`Carbon.Snd`	Interface to the Sound Manager.
`Carbon.TE`	Interface to TextEdit.
`Carbon.Win`	Interface to the Window Manager.
`ColorPicker`	Interface to the standard color selection dialog.

4.1 `Carbon.AE` — Apple Events

4.2 `Carbon.AH` — Apple Help

4.3 `Carbon.App` — Appearance Manager

4.4 `Carbon.CF` — Core Foundation

The `CFBase`, `CFArray`, `CFData`, `CFDictionary`, `CFString` and `CFURL` objects are supported, some only partially.

4.5 `Carbon.CG` — Core Graphics

4.6 `Carbon.CarbonEvt` — Carbon Event Manager

4.7 `Carbon.Cm` — Component Manager

4.8 `Carbon.Ctl` — Control Manager

4.9 `Carbon.Dlg` — Dialog Manager

4.10 `Carbon.Evt` — Event Manager

4.11 `Carbon.Fm` — Font Manager

4.12 `Carbon.Folder` — Folder Manager

4.13 `Carbon.Help` — Help Manager

4.14 `Carbon.List` — List Manager

4.15 `Carbon.Menu` — Menu Manager

4.16 `Carbon.Mlte` — MultiLingual Text Editor

4.17 `Carbon.Qd` — QuickDraw

4.18 `Carbon.Qdoffs` — QuickDraw Offscreen

4.19 `Carbon.Qt` — QuickTime

4.20 `Carbon.Res` — Resource Manager and Handles

4.21 `Carbon.Scrap` — Scrap Manager

4.22 `Carbon.Snd` — Sound Manager

4.23 `Carbon.TE` — TextEdit

4.24 `Carbon.Win` — Window Manager

4.25 `ColorPicker` — Color selection dialog

The `ColorPicker` module provides access to the standard color picker dialog.

GetColor(*prompt, rgb*)

Show a standard color selection dialog and allow the user to select a color. The user is given instruction by the *prompt* string, and the default color is set to *rgb*. *rgb* must be a tuple giving the red, green, and blue components of the color. `GetColor()` returns a tuple giving the user's selected color and a flag indicating whether they accepted the selection of cancelled.

CHAPTER

FIVE

Undocumented Modules

The modules in this chapter are poorly documented (if at all). If you wish to contribute documentation of any of these modules, please get in touch with docs@python.org.

applesingle	Rudimentary decoder for AppleSingle format files.
buildtools	Helper module for BuildApplet, BuildApplication and macfreeze.
cfmfile	Code Fragment Resource module.
icopen	Internet Config replacement for `open()`.
macerrors	Constant definitions for many Mac OS error codes.
macresource	Locate script resources.
Nav	Interface to Navigation Services.
PixMapWrapper	Wrapper for PixMap objects.
videoreader	Read QuickTime movies frame by frame for further processing.
W	Widgets for the Mac, built on top of `FrameWork`.

5.1 `applesingle` — AppleSingle decoder

5.2 `buildtools` — Helper module for BuildApplet and Friends

Deprecated since release 2.4.

5.3 `cfmfile` — Code Fragment Resource module

`cfmfile` is a module that understands Code Fragments and the accompanying "cfrg" resources. It can parse them and merge them, and is used by BuildApplication to combine all plugin modules to a single executable.

Deprecated since release 2.4.

5.4 `icopen` — Internet Config replacement for `open()`

Importing `icopen` will replace the builtin `open()` with a version that uses Internet Config to set file type and creator for new files.

5.5 `macerrors` — Mac OS Errors

`macerrors` contains constant definitions for many Mac OS error codes.

5.6 `macresource` — Locate script resources

`macresource` helps scripts finding their resources, such as dialogs and menus, without requiring special case code for when the script is run under MacPython, as a MacPython applet or under OSX Python.

5.7 `Nav` — NavServices calls

A low-level interface to Navigation Services.

5.8 `PixMapWrapper` — Wrapper for PixMap objects

`PixMapWrapper` wraps a PixMap object with a Python object that allows access to the fields by name. It also has methods to convert to and from `PIL` images.

5.9 `videoreader` — Read QuickTime movies

`videoreader` reads and decodes QuickTime movies and passes a stream of images to your program. It also provides some support for audio tracks.

5.10 `W` — Widgets built on `FrameWork`

The `W` widgets are used extensively in the **IDE**.

APPENDIX

A

History and License

A.1 History of the software

Python was created in the early 1990s by Guido van Rossum at Stichting Mathematisch Centrum (CWI, see http://www.cwi.nl/) in the Netherlands as a successor of a language called ABC. Guido remains Python's principal author, although it includes many contributions from others.

In 1995, Guido continued his work on Python at the Corporation for National Research Initiatives (CNRI, see http://www.cnri.reston.va.us/) in Reston, Virginia where he released several versions of the software.

In May 2000, Guido and the Python core development team moved to BeOpen.com to form the BeOpen PythonLabs team. In October of the same year, the PythonLabs team moved to Digital Creations (now Zope Corporation; see http://www.zope.com/). In 2001, the Python Software Foundation (PSF, see http://www.python.org/psf/) was formed, a non-profit organization created specifically to own Python-related Intellectual Property. Zope Corporation is a sponsoring member of the PSF.

All Python releases are Open Source (see http://www.opensource.org/ for the Open Source Definition). Historically, most, but not all, Python releases have also been GPL-compatible; the table below summarizes the various releases.

Release	Derived from	Year	Owner	GPL compatible?
0.9.0 thru 1.2	n/a	1991-1995	CWI	yes
1.3 thru 1.5.2	1.2	1995-1999	CNRI	yes
1.6	1.5.2	2000	CNRI	no
2.0	1.6	2000	BeOpen.com	no
1.6.1	1.6	2001	CNRI	no
2.1	2.0+1.6.1	2001	PSF	no
2.0.1	2.0+1.6.1	2001	PSF	yes
2.1.1	2.1+2.0.1	2001	PSF	yes
2.2	2.1.1	2001	PSF	yes
2.1.2	2.1.1	2002	PSF	yes
2.1.3	2.1.2	2002	PSF	yes
2.2.1	2.2	2002	PSF	yes
2.2.2	2.2.1	2002	PSF	yes
2.2.3	2.2.2	2002-2003	PSF	yes
2.3	2.2.2	2002-2003	PSF	yes
2.3.1	2.3	2002-2003	PSF	yes
2.3.2	2.3.1	2003	PSF	yes
2.3.3	2.3.2	2003	PSF	yes
2.3.4	2.3.3	2004	PSF	yes
2.3.5	2.3.4	2005	PSF	yes
2.4	2.3	2004	PSF	yes
2.4.1	2.4	2005	PSF	yes
2.4.2	2.4.1	2005	PSF	yes
2.4.3	2.4.2	2006	PSF	yes
2.4.4	2.4.3	2006	PSF	yes
2.5	2.4	2006	PSF	yes
2.5.1	2.5	2007	PSF	yes
2.5.2	2.5.1	2008	PSF	yes

Note: GPL-compatible doesn't mean that we're distributing Python under the GPL. All Python licenses, unlike the GPL, let you distribute a modified version without making your changes open source. The GPL-compatible licenses make it possible to combine Python with other software that is released under the GPL; the others don't.

Thanks to the many outside volunteers who have worked under Guido's direction to make these releases possible.

A.2 Terms and conditions for accessing or otherwise using Python

PSF LICENSE AGREEMENT FOR PYTHON 2.5.3

1. This LICENSE AGREEMENT is between the Python Software Foundation ("PSF"), and the Individual or Organization ("Licensee") accessing and otherwise using Python 2.5.3 software in source or binary form and its associated documentation.

2. Subject to the terms and conditions of this License Agreement, PSF hereby grants Licensee a nonexclusive, royalty-free, world-wide license to reproduce, analyze, test, perform and/or display publicly, prepare derivative works, distribute, and otherwise use Python 2.5.3 alone or in any derivative version, provided, however, that PSF's License Agreement and PSF's notice of copyright, i.e., "Copyright © 2001-2008 Python Software Foundation; All Rights Reserved" are retained in Python 2.5.3 alone or in any derivative version prepared by Licensee.

3. In the event Licensee prepares a derivative work that is based on or incorporates Python 2.5.3 or any part thereof, and wants to make the derivative work available to others as provided herein, then Licensee hereby agrees to include in any such work a brief summary of the changes made to Python 2.5.3.

4. PSF is making Python 2.5.3 available to Licensee on an "AS IS" basis. PSF MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED. BY WAY OF EXAMPLE, BUT NOT LIMITATION, PSF MAKES NO AND DISCLAIMS ANY REPRESENTATION OR WARRANTY OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF PYTHON 2.5.3 WILL NOT INFRINGE ANY THIRD PARTY RIGHTS.

5. PSF SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF PYTHON 2.5.3 FOR ANY INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR LOSS AS A RESULT OF MODIFYING, DISTRIBUTING, OR OTHERWISE USING PYTHON 2.5.3, OR ANY DERIVATIVE THEREOF, EVEN IF ADVISED OF THE POSSIBILITY THEREOF.

6. This License Agreement will automatically terminate upon a material breach of its terms and conditions.

7. Nothing in this License Agreement shall be deemed to create any relationship of agency, partnership, or joint venture between PSF and Licensee. This License Agreement does not grant permission to use PSF trademarks or trade name in a trademark sense to endorse or promote products or services of Licensee, or any third party.

8. By copying, installing or otherwise using Python 2.5.3, Licensee agrees to be bound by the terms and conditions of this License Agreement.

BEOPEN.COM LICENSE AGREEMENT FOR PYTHON 2.0
BEOPEN PYTHON OPEN SOURCE LICENSE AGREEMENT VERSION 1

1. This LICENSE AGREEMENT is between BeOpen.com ("BeOpen"), having an office at 160 Saratoga Avenue, Santa Clara, CA 95051, and the Individual or Organization ("Licensee") accessing and otherwise using this software in source or binary form and its associated documentation ("the Software").

2. Subject to the terms and conditions of this BeOpen Python License Agreement, BeOpen hereby grants Licensee a non-exclusive, royalty-free, world-wide license to reproduce, analyze, test, perform and/or display publicly, prepare derivative works, distribute, and otherwise use the Software alone or in any derivative version, provided, however, that the BeOpen Python License is retained in the Software, alone or in any derivative version prepared by Licensee.

3. BeOpen is making the Software available to Licensee on an "AS IS" basis. BEOPEN MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED. BY WAY OF EXAMPLE, BUT NOT LIMITATION, BEOPEN MAKES NO AND DISCLAIMS ANY REPRESENTATION OR WARRANTY OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF THE SOFTWARE WILL NOT INFRINGE ANY THIRD PARTY RIGHTS.

4. BEOPEN SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF THE SOFTWARE FOR ANY INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR LOSS AS A RESULT OF USING, MODIFYING OR DISTRIBUTING THE SOFTWARE, OR ANY DERIVATIVE THEREOF, EVEN IF ADVISED OF THE POSSIBILITY THEREOF.

5. This License Agreement will automatically terminate upon a material breach of its terms and conditions.

6. This License Agreement shall be governed by and interpreted in all respects by the law of the State of California, excluding conflict of law provisions. Nothing in this License Agreement shall be deemed to create any relationship of agency, partnership, or joint venture between BeOpen and Licensee. This License Agreement does not grant permission to use BeOpen trademarks or trade names in a trademark sense to endorse or promote products or services of Licensee, or any third party. As an exception, the "BeOpen Python" logos available at http://www.pythonlabs.com/logos.html may be used according to the permissions granted on that web page.

7. By copying, installing or otherwise using the software, Licensee agrees to be bound by the terms and conditions of this License Agreement.

CNRI LICENSE AGREEMENT FOR PYTHON 1.6.1

CWI LICENSE AGREEMENT FOR PYTHON 0.9.0 THROUGH 1.2

SHALL STICHTING MATHEMATISCH CENTRUM 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.

A.3 Licenses and Acknowledgements for Incorporated Software

This section is an incomplete, but growing list of licenses and acknowledgements for third-party software incorporated in the Python distribution.

A.3.1 Mersenne Twister

The `_random` module includes code based on a download from http://www.math.keio.ac.jp/ matumoto/MT2002/emt19937ar.html. The following are the verbatim comments from the original code:

```
A C-program for MT19937, with initialization improved 2002/1/26.
Coded by Takuji Nishimura and Makoto Matsumoto.

Before using, initialize the state by using init_genrand(seed)
or init_by_array(init_key, key_length).

Copyright (C) 1997 - 2002, Makoto Matsumoto and Takuji Nishimura,
All rights reserved.

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

 1. Redistributions of source code must retain the above copyright
    notice, this list of conditions and the following disclaimer.

 2. Redistributions in binary form must reproduce the above copyright
    notice, this list of conditions and the following disclaimer in the
    documentation and/or other materials provided with the distribution.

 3. The names of its contributors may not be used to endorse or promote
    products derived from this software without specific prior written
    permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
A PARTICULAR PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE COPYRIGHT OWNER OR
CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.


Any feedback is very welcome.
http://www.math.keio.ac.jp/matumoto/emt.html
email: matumoto@math.keio.ac.jp
```

A.3.2 Sockets

The `socket` module uses the functions, `getaddrinfo`, and `getnameinfo`, which are coded in separate source files from the WIDE Project, http://www.wide.ad.jp/about/index.html.

A.3.3 Floating point exception control

The source for the `fpectl` module includes the following notice:

A.3.4 MD5 message digest algorithm

The source code for the `md5` module contains the following notice:

```
  Copyright (C) 1999, 2002 Aladdin Enterprises.  All rights reserved.

  This software is provided 'as-is', without any express or implied
  warranty.  In no event will the authors be held liable for any damages
  arising from the use of this software.

  Permission is granted to anyone to use this software for any purpose,
  including commercial applications, and to alter it and redistribute it
  freely, subject to the following restrictions:

  1. The origin of this software must not be misrepresented; you must not
     claim that you wrote the original software. If you use this software
     in a product, an acknowledgment in the product documentation would be
     appreciated but is not required.
  2. Altered source versions must be plainly marked as such, and must not be
     misrepresented as being the original software.
  3. This notice may not be removed or altered from any source distribution.

  L. Peter Deutsch
  ghost@aladdin.com

  Independent implementation of MD5 (RFC 1321).

  This code implements the MD5 Algorithm defined in RFC 1321, whose
  text is available at
http://www.ietf.org/rfc/rfc1321.txt
  The code is derived from the text of the RFC, including the test suite
  (section A.5) but excluding the rest of Appendix A.  It does not include
  any code or documentation that is identified in the RFC as being
  copyrighted.

  The original and principal author of md5.h is L. Peter Deutsch
  <ghost@aladdin.com>.  Other authors are noted in the change history
  that follows (in reverse chronological order):

  2002-04-13 lpd Removed support for non-ANSI compilers; removed
references to Ghostscript; clarified derivation from RFC 1321;
now handles byte order either statically or dynamically.
  1999-11-04 lpd Edited comments slightly for automatic TOC extraction.
  1999-10-18 lpd Fixed typo in header comment (ansi2knr rather than md5);
added conditionalization for C++ compilation from Martin
Purschke <purschke@bnl.gov>.
  1999-05-03 lpd Original version.
```

A.3.5 Asynchronous socket services

The `asynchat` and `asyncore` modules contain the following notice:

```
Copyright 1996 by Sam Rushing

                        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, and that the name of Sam
Rushing not be used in advertising or publicity pertaining to
distribution of the software without specific, written prior
permission.

SAM RUSHING DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE,
INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS, IN
NO EVENT SHALL SAM RUSHING 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.
```

A.3.6 Cookie management

The `Cookie` module contains the following notice:

```
Copyright 2000 by Timothy O'Malley <timo@alum.mit.edu>

               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, and that the name of
Timothy O'Malley  not be used in advertising or publicity
pertaining to distribution of the software without specific, written
prior permission.

Timothy O'Malley DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS
SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY
AND FITNESS, IN NO EVENT SHALL Timothy O'Malley 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.
```

A.3.7 Profiling

The `profile` and `pstats` modules contain the following notice:

A.3.8 Execution tracing

The `trace` module contains the following notice:

```
portions copyright 2001, Autonomous Zones Industries, Inc., all rights...
err...  reserved and offered to the public under the terms of the
Python 2.2 license.
Author: Zooko O'Whielacronx
http://zooko.com/
mailto:zooko@zooko.com

Copyright 2000, Mojam Media, Inc., all rights reserved.
Author: Skip Montanaro

Copyright 1999, Bioreason, Inc., all rights reserved.
Author: Andrew Dalke

Copyright 1995-1997, Automatrix, Inc., all rights reserved.
Author: Skip Montanaro

Copyright 1991-1995, Stichting Mathematisch Centrum, all rights reserved.


Permission to use, copy, modify, and distribute this Python software and
its associated documentation for any purpose without fee is hereby
granted, provided that the above copyright notice appears in all copies,
and that both that copyright notice and this permission notice appear in
supporting documentation, and that the name of neither Automatrix,
Bioreason or Mojam Media be used in advertising or publicity pertaining to
distribution of the software without specific, written prior permission.
```

A.3.9 UUencode and UUdecode functions

The uu module contains the following notice:

```
Copyright 1994 by Lance Ellinghouse
Cathedral City, California Republic, United States of America.
                       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, and that the name of Lance Ellinghouse
not be used in advertising or publicity pertaining to distribution
of the software without specific, written prior permission.
LANCE ELLINGHOUSE DISCLAIMS ALL WARRANTIES WITH REGARD TO
THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS, IN NO EVENT SHALL LANCE ELLINGHOUSE CENTRUM 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.

Modified by Jack Jansen, CWI, July 1995:
- Use binascii module to do the actual line-by-line conversion
  between ascii and binary. This results in a 1000-fold speedup. The C
  version is still 5 times faster, though.
- Arguments more compliant with python standard
```

A.3.10 XML Remote Procedure Calls

The `xmlrpclib` module contains the following notice:

```
    The XML-RPC client interface is

Copyright (c) 1999-2002 by Secret Labs AB
Copyright (c) 1999-2002 by Fredrik Lundh

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

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

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

MODULE INDEX

INDEX

Symbols

A

B

C

D