37.5. `EasyDialogs` — Basic Macintosh dialogs¶

Platforms: Mac

The `EasyDialogs` module contains some simple dialogs for the Macintosh. The dialogs get launched in a separate application which appears in the dock and must be clicked on for the dialogs be displayed. 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.

Note

This module has been removed in Python 3.x.

The `EasyDialogs` module defines the following functions:

`EasyDialogs.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.

`EasyDialogs.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.

`EasyDialogs.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()`.

`EasyDialogs.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.

`EasyDialogs.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.

`EasyDialogs.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.

`EasyDialogs.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.

`EasyDialogs.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.

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

37.5.1. 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:

`ProgressBar.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.

`ProgressBar.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.

`ProgressBar.title`([*newstr*])¶
Sets the text in the title bar of the progress dialog to *newstr*.

`ProgressBar.label`([*newstr*])¶
Sets the text in the progress box of the progress dialog to *newstr*.

`ProgressBar.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.

`ProgressBar.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.

Table Of Contents

Previous topic

37.3. `macostools` — Convenience routines for file manipulation

Next topic

37.6. `FrameWork` — Interactive application framework

This Page

- Show Source