
This is Python for S60 version 1.4.0 final created by Nokia Corporation. Files added by Nokia Corporation are licensed under Apache License Version 2.0. The original software, including modifications of Nokia Corporation therein, is licensed under the applicable license(s) for Python 2.2.2, unless specifically indicated otherwise in the relevant source code file.

See http://www.apache.org/licenses/LICENSE-2.0 and http://www.python.org/2.2.2/license.html
## CONTENTS

1 Introduction ................................................. 1
   1.1 Scope .................................................. 1
   1.2 Audience .............................................. 2
   1.3 Naming Conventions ................................. 2

2 API Summary ............................................... 3
   2.1 Python Standard Library ............................... 3
   2.2 Python for S60 Extensions ............................ 3
   2.3 Third-Party Extensions ............................... 4

3 Selected Issues on Python Programming for S60 ............ 5
   3.1 Concurrency Aspects ................................. 5
   3.2 Running Python for S60 Scripts ..................... 5
   3.3 Standard I/O Streams ................................ 6
   3.4 Usage of Unicode .................................... 6
   3.5 Date and Time ....................................... 6
   3.6 Limitations of Thread Support ...................... 6
   3.7 Scalable User Interface ............................. 7
   3.8 Error Handling ....................................... 7
   3.9 Limitations and Areas of Development ............... 7

4 Operating System Services and Information ................. 9
   4.1 e32 — A Symbian OS related services package ..... 9
   4.2 sysinfo — Access to system information .......... 11

5 User Interface and Graphics .................................. 13
   5.1 appuifw — Interface to the S60 GUI framework .... 13
   5.2 graphics — A graphics related services package ... 27
   5.3 camera — Interface for taking photographs ....... 33
   5.4 keycapture — Interface for global capturing of key events. 35
   5.5 topwindow — Interface for creating windows that are shown on top of other applications. 36
   5.6 gles — Bindings to OpenGL ES ...................... 38
   5.7 glcanvas — UI Control for Displaying OpenGL ES Graphics. 44

6 Audio and Communication Services ............................ 47
   6.1 audio — An audio related services package ....... 47
   6.2 telephone — Telephone services ..................... 49
   6.3 messaging — A messaging services package .. ... 49
   6.4 inbox — Interface to device inbox ................. 50
   6.5 location — GSM location information .............. 51
   6.6 position — Simplified interface to the Location Acquisition API .............................. 52

7 Data Management ........................................... 55
   7.1 contacts — A contacts related services package ... 55
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.2</td>
<td>calendar — Access to calendar related services</td>
<td>60</td>
</tr>
<tr>
<td>7.3</td>
<td>calendar for EKA2 — Access to calendar related services</td>
<td>65</td>
</tr>
<tr>
<td>7.4</td>
<td>e32db — Interface to the Symbian native DB</td>
<td>70</td>
</tr>
<tr>
<td>7.5</td>
<td>e32dbm — DBM implemented using the Symbian native DBMS</td>
<td>73</td>
</tr>
<tr>
<td>8</td>
<td>Standard Library Support and Extensions</td>
<td>75</td>
</tr>
<tr>
<td>8.1</td>
<td>Support for Python Standard Library</td>
<td>75</td>
</tr>
<tr>
<td>8.2</td>
<td>Extensions to Standard Library Modules</td>
<td>76</td>
</tr>
<tr>
<td>9</td>
<td>Extending and Embedding</td>
<td>79</td>
</tr>
<tr>
<td>9.1</td>
<td>Python/C API Extensions</td>
<td>79</td>
</tr>
<tr>
<td>9.2</td>
<td>Extending Python for S60</td>
<td>80</td>
</tr>
<tr>
<td>10</td>
<td>Terms and Abbreviations</td>
<td>83</td>
</tr>
<tr>
<td>A</td>
<td>Reporting Bugs</td>
<td>87</td>
</tr>
</tbody>
</table>
| Module Index | Index | 89   | 91
CHAPTER

ONE

Introduction

The Python for S60 Platform (Python for S60) simplifies application development and provides a scripting solution for the Symbian C++ APIs. This document is for Python for S60 release 1.4.0 final that is based on Python 2.2.2.

The documentation for Python for S60 includes three documents:

- Getting Started with Python for S60 Platform [5] contains information on how to install Python for S60 and how to write your first program.
- This document contains API and other reference material.
- Programming with Python for S60 Platform [6] contains code examples and programming patterns for S60 devices that can be used as a basis for programs.

The Python for S60 as installed on a S60 device consists of:

- Python runtime package that consists of:
  - Python interpreter DLL
  - Standard and proprietary Python library modules
  - S60 UI application framework adaptation component (a DLL) that connects the scripting domain components to the S60 UI

- Python script shell package that consists of:
  - an application written in Python and visible in the application menu of the device that provides an execution environment for Python scripts.
  - For S60 platform versions prior to 3rd Edition: Python Installer program for installing Python files on the device, which consists of:
    - A recognizer plug-in that recognizes .py, .pyc, .pyd and .pyo files as belonging to Python.
    - Symbian application written in Python that handles the installation of recognized Python files into the script shell environment.

A plugin for the S60 C++ SDK is also available. This plugin makes it possible to run Python scripts in the S60 emulator environment and to compile Python extension modules (PYDs) for the emulator and the device.

The Python for S60 developer discussion board [9] on the Forum Nokia Web site is a useful resource for finding out information on specific topics concerning Python for S60. You are welcome to give feedback or ask questions about Python for S60 through this discussion board.

1.1 Scope

This document includes the information required by developers to create applications that use Python for S60, and some advice on extending the platform.
1.2 Audience

This guide is intended for developers looking to create programs that use the native features and resources of the S60 phones. The reader should be familiar with the Python programming language ([http://www.python.org/](http://www.python.org/)) and the basics of using Python for S60 (see Getting Started with Python for S60 Platform [5]).

1.3 Naming Conventions

Most names of the type `ESomething` typically indicate a constant defined by the Symbian SDK. More information about these constants can be found in the Symbian SDK documentation.
CHAPTER TWO

API Summary

All built-in object types of the Python language are supported in the S60 environment. The rest of the programming interfaces are implemented by various library modules as summarized in this chapter.

2.1 Python Standard Library

Python for S60 platform distribution does not include all of the Python’s standard and optional library modules to save storage space in the phone. Nevertheless, many of the excluded modules also work in the S60 Python environment without any modifications. Some modules are included in the SDK version but not installed in the phone. For a summary of supported library modules, see Chapter 8.

When Python, available at [http://www.python.org/](http://www.python.org/), is installed on a PC, the library modules are by default located in ‘\Python22\Lib’ on Windows and in ‘/usr/lib/python2.2’ on Linux. The Python library modules’ APIs are documented in [1].

Python for S60 extends some standard modules. These extensions are described in this document, see Chapter 8.2.

2.2 Python for S60 Extensions

There are two kinds of native C++ extensions in the Python for S60 Platform: built-in extensions and dynamically loadable extensions.

2.2.1 Built-in extensions

There are two built-in extensions in the Python for S60 package:

- The e32 extension module is built into the Python interpreter on Symbian OS, and implements interfaces to special Symbian OS Platform services that are not accessible via Python standard library modules.

- The appuifw module for Python for S60 Platform offers UI application framework related Python interfaces.

2.2.2 Dynamically loadable extensions

These dynamically loadable extension modules provide proprietary APIs to S60 Platform’s services:

- **graphics**: see Chapter 5.2
- **e32db**: see Chapter 7.4
- **messaging**: see Chapter 6.3
2.3 Third-Party Extensions

It is also possible to write your own Python extensions. S60 related extensions to Python/C API are described in Chapter 9.1. For some further guidelines on writing extensions in C/C++, see Chapter 9.2. In addition, for an example on porting a simple extension to S60, see [6].
CHAPTER
THREE

Selected Issues on Python Programming for S60

The following issues must be considered when using Python on S60.

3.1 Concurrency Aspects

The thread that initializes the Python interpreter becomes the main Python thread. This is usually the main thread of a UI application. When an application written in Python launches, the Symbian platform infrastructure creates the main UI thread that starts the Python environment. If a Python program is started as a server with `e32.start_server`, then the Python main thread is not a UI thread.

It is possible to launch new threads via the services of `thread` module. Examples of such situations could be to overcome eventual problems with the fixed, relatively small stack size of the main UI application thread; or to perform some background processing while still keeping the UI responsive. These new threads are not allowed to directly manipulate the UI; in other words, they may not use the `appuifw` module.

Because of the limitations of the Python interpreter’s final cleanup, Python applications on the Symbian OS should be designed in such a way that the main thread is the last thread alive.

A facility called active object is used extensively on the Symbian OS to implement co-operative, non-preemptive scheduling within operating system threads. This facility is also utilized with native APIs. A Python programmer is exposed to related concurrency issues particularly in UI programming. Preserving the responsiveness of the UI with the help of active objects needs to be considered when designing the application logic. At the same time it is necessary to take into account the resulting concurrent behavior within the application when active objects are used. While the main execution path of a UI script is blocked in wait for an active object to complete – either explicitly as a result of using `e32.Ao_lock`, or indirectly within some other Python API implementation – the UI-related callbacks may still get called.

The standard `thread.lock` cannot normally be used for synchronization in the UI application main thread, as it blocks the UI event handling that takes place in the same thread context. The Symbian active object based synchronization service called `e32.Ao_lock` has been implemented to overcome this problem. The main thread can wait in this lock, while the UI remains responsive.

Python for S60 tries to minimize the unwanted exposure of a Python programmer to the active objects of the Symbian OS. The programmer may choose to implement the eventual concurrent behavior of the application with normal threads. However, certain active object based facilities are offered as an option in the `e32` module.

3.2 Running Python for S60 Scripts

The current options for installing Python scripts to a S60 device are: a stand-alone installation to the device’s main application menu, and an installation to a folder hierarchy maintained by the Python script
shell. For more details on this topic, see Programming with Python for S60 Platform [6]. In the first case the script application is launched via application menu, and it executes in its own process context. The latter case is suitable for development, testing, and trying out new scripts.

The Python script shell delivered with Python for S60 package has itself been written in Python. It is a collection of scripts that offer an interactive Python console and a possibility to execute scripts located in the directory of the script shell. Due to this kind of design the scripts are not fully isolated from each other. This means that any changes a script makes in the script shell namespace are visible to other scripts as well. This may be helpful during the development of a script suite, as long as care is taken to avoid unwanted interference between scripts.

For some special issues to consider when writing Python scripts to be run in the current Python script shell, see Programming with Python for S60 Platform [6]. These include the arrangements for standard output and the maintenance of the Options menu contents.

**Note:** Note that unlike some previous releases, the current version of the Python for S60 script shell takes care of restoring `appuifw.app.menu`, `appuifw.app.title`, `appuifw.app.exit_key_handler`, `appuifw.app.screen`, `appuifw.app.body`, `sys.stderr` and `??` after a script has been run, and The application programmer doesn’t need to save and restore these settings.

### 3.3 Standard I/O Streams

The standard Python I/O streams in the `sys` module are by default connected to underlying C STDLIB’s `stdio` streams that in turn are terminated by dummy file descriptors. Usually Python scripts set the I/O streams suitably by manipulating them at Python level via `sys` module interface. The `e32` extension module offers a Python interface for attaching to C STDLIB’s output streams, but this service is only recommended for debugging purposes. The `e32._stdo` function takes as its argument the name of the file where C STDLIB’s `stdout` and `stderr` are to be redirected. This makes it possible to capture the low-level error output when the Python interpreter has detected a fatal error and aborts.

### 3.4 Usage of Unicode

No changes have been made to the standard library modules with regard to string argument and return value types. S60 extensions generally accept both plain strings and Unicode strings as arguments, but they return only Unicode strings. APIs that take string arguments for the purpose of showing them on the UI expect Unicode strings. Giving something else may result in garbled appearance of the text on the screen.

### 3.5 Date and Time

Unix time, seconds since January 1, 1970, 00:00:00 UTC (Coordinated Universal Time), is generally used as the time format in the Python for S60 APIs described in this document. The float type is used for storing time values.

### 3.6 Limitations of Thread Support

Python for S60 supports starting native threads via the standard `thread` module. However, the native APIs Python for S60 uses have certain limitations that a Python programmer must be aware of.

Objects that wrap native resources can typically be used only in the thread they are created in. This is because native resources cannot be shared between native threads. Examples:

**Note:**
• Symbian OS STDLIB implementation has some limitations that are reflected at OS module support (see S60 SDK documentation [4]). For example, STDLIB file descriptors cannot be shared between threads, and for that reason, Python file objects cannot either.

• Sockets as implemented in the S60 version of the socket module.

Warning: Trying to use native objects from the wrong thread can crash the interpreter. If display of panic codes is enabled, a typical panic code displayed in this case is “KERN-EXEC 3”.

3.7 Scalable User Interface

Note: S60 2nd Edition FP3 and further releases.

S60 2nd Edition FP3 enables a new feature called scalable user interface. For Python developers this feature is currently visible in new APIs supporting the scalable UI, icon loading, and new screen resolutions. For more information on scalable user interface, see Section 5.1.8, Icon Type of this document, as well as Programming with Python for S60 Platform [6].

3.8 Error Handling

The APIs described in this document may raise any standard Python exceptions. In situations where a Symbian error code is returned, its symbolic name is given as the value parameter of a SymbianError exception.

In case where the functions have nothing special to return, they return None on success.

3.9 Limitations and Areas of Development

Some OS level concepts to which the standard os library module offers an interface do not exist as such in Symbian OS environment. An example of this is the concept of current working directory.

Reference cycle garbage collection is not in use. Because of this, special care needs to be taken to dismantle cyclic references when a Python program exits. This prevents error messages related to native resources that are left open. The problem could be removed by developing support for collection of cyclic garbage or by performing a special cleanup action on interpreter exit. The gc module has been ported to the Symbian OS, and it has been verified to work. However, the current distribution has been built without gc support.
Operating System Services and Information

4.1 e32 — A Symbian OS related services package

The e32 module offers Symbian OS related utilities that are not related to the UI and are not provided by the standard Python library modules.

4.1.1 Module Level Functions

The following free functions - functions that do not belong to any class - are defined in the e32 module:

`ao_yield()`
Yields to the active scheduler to have ready active objects with priority above normal scheduled for running. This has the effect of flushing the eventual pending UI events. Note that the UI callback code may be run in the context of the thread that performs an `ao_yield`. For information on active scheduler, see S60 SDK documentation [4].

`ao_sleep(interval[, callback])`
Sleeps for the given interval without blocking the active scheduler. When the optional callback is given, the call to `ao_sleep` returns immediately and the callback gets called after interval. See also Section 4.1.3, Ao_timer Type.

`ao_callgate(wrapped_callable)`
Wraps `wrapped_callable` into returned callable object `callgate` that can be called in any thread. As a result of a call to `callgate`, `wrapped_callable` gets called in the context of the thread that originally created the callgate. Arguments can be given to the call. This is actually a simple wrapping of the Symbian active object facility.

`drive_list()`
Returns a list of currently visible drives as a list of Unicode strings '<driveletter>:'

`file_copy(target_name, source_name)`
Copies the file `source_name` to `target_name`. The names must be complete paths.

`in_emulator()`
Returns 1 if running in an emulator, or 0 if running on a device.

`set_home_time(time)`
Set the device’s time to `time` (see Section 3.5).

`pys60_version`
A string containing the version number of the Python for S60 and some additional information. Example:

```python
>>> import e32
>>> e32.pys60_version
'1.2 final'
```

`pys60_version_info`
A tuple containing the five components of the Python for S60 version number: major, minor, micro, release tag, and serial. All values except release level are integers; the release tag is a string. A value other than 'final' for the release tag signifies a development release. The \texttt{pys60\_version\_info} value corresponding to the Python for S60 version 1.2 is (1, 2, 0, 'final', 0).

\texttt{s60\_version\_info}
The SDK version with which this Python was compiled (tuple). The following values are possible:

- (1, 2) for S60 1st Edition
- (2, 0) for S60 2nd Edition
- (2, 6) S60 2nd Edition Feature Pack 2
- (2, 8) S60 2nd Edition Feature Pack 3
- (3, 0) S60 3rd Edition

Examples:

\begin{verbatim}
>>> import e32
>>> e32.pys60_version
'1.2.0 final'
>>> e32.pys60_version_info
(1, 2, 0, 'final', 0)
>>> e32.s60_version_info
(2, 0)
\end{verbatim}

\texttt{is\_ui\_thread()}
Returns True if the code that calls this function runs in the context of the UI thread; otherwise returns False.

\texttt{start\_exe(filename, command \[,wait\])}
Launches the native Symbian OS executable \texttt{filename} (Unicode) and passes it the \texttt{command} string. When \texttt{wait} is set, the function synchronously waits for the exit of the executable and returns a value that describes the exit type. Possible values are 0 for normal exit and 2 for abnormal exit.

\texttt{start\_server(filename)}
Starts the Python script in file \texttt{filename} (Unicode) as a server in its own process. Note that \texttt{appui\_fw} module is not available to a server script.

\texttt{reset\_inactivity()}
Resets the timers since the user was last active. As a consequence, the device backlight is normally turned on when this function is invoked.

\texttt{inactivity()}
Returns the time in seconds since the user of the device was last active.

4.1.2 Ao\_lock Type

class Ao\_lock()
Creates an Ao\_lock instance. A Symbian active object based synchronization service. This can be used in the main thread without blocking the handling of UI events. The application should not exit while a thread is waiting in Ao\_lock. If Ao\_lock.wait is called while another wait call is already in progress, an AssertionError is raised.

Instances of Ao\_lock type have the following methods:

\texttt{wait()}
If the lock has already been signaled, returns immediately. Otherwise blocks in wait for the lock to be signaled. Only one waiter is allowed, so you should avoid recursive calls to this service. \texttt{wait} can only be called in the thread that created the lock object. During the wait, other Symbian-active objects are being served, so the UI will not freeze. This may result in the UI callback code
being run in the context of the thread that is waiting in Ao_lock. This must be considered when designing the application logic.

\texttt{signal()}
Signals the lock. The waiter is released.

4.1.3 Ao_timer Type

The rationale for the \texttt{Ao_timer} type is that you cannot cancel a pending \texttt{e32.ao.sleep}. This is problematic if e.g. the user exits an application which is sleeping. In this case a panic would occur since the sleep is not cancelled - this is the reason you should avoid using \texttt{e32.ao.sleep} and instead use the \texttt{Ao_timer} with appropriate \texttt{cancel} calls if there is for example a possibility for the user to exit the application during a sleep.

\texttt{class Ao_timer()}
Creates an \texttt{Ao_timer} instance. A Symbian active object based sleeping service. This can be used in the main thread without blocking the handling of UI events. The application should not exit while a thread has a pending \texttt{after} call in \texttt{Ao_timer}. Only one \texttt{after} invocation can be pending at time for each instance of this type.

Instances of \texttt{Ao_timer} type have the following methods:

\texttt{after(interval \[,callback\] )}
Sleeps for the given interval without blocking the active scheduler. When the optional callback is given, the call to \texttt{after} returns immediately and the callback gets called after interval.

\texttt{cancel()}
Cancels a pending \texttt{after} call.

4.2 \texttt{sysinfo} — Access to system information

The \texttt{sysinfo} module offers an API for checking the system information of a S60 mobile device.

\textbf{Note:} The method \texttt{ring_type} is not available for S60 1st Edition.

The \texttt{sysinfo} module has the following functions:

\texttt{active_profile()}
Returns the current active profile as a string, which can be one of the following: \texttt{'general'}, \texttt{'silent'}, \texttt{'meeting'}, \texttt{'outdoor'}, \texttt{'pager'}, \texttt{'offline'}, \texttt{'drive'}, or \texttt{'user <profile value>'}.

\texttt{battery()}
Returns the current battery level. On devices based on S60 2nd Edition Feature Pack 1 (S60 2.1) or earlier the value ranges from 0 (empty) to 7 (full). On newer devices the value ranges from 0 (empty) to 100 (full). On the emulator the value is always 0.

\textbf{Note:} The returned value may be incorrect while the device is being charged.

\texttt{display_twips()}
Returns the width and height of the display in twips. For a definition of a twip, see Chapter 10, Terms and Abbreviations.

\texttt{display_pixels()}
Returns the width and height of the display in pixels.

\texttt{free_drivespace()}
Returns the amount of free space left on the drives in bytes, for example \{u'C:' 100\}. The keys in the dictionary are the drive letters followed by a colon (:).

\texttt{imei()}
Returns the IMEI code of the device as a Unicode string or, if running on the emulator, the hardcoded string \texttt{u’00000000000000’}.
max_ramdrive_size()
Returns the maximum size of the RAM drive on the device.

total_ram()
Returns the amount of RAM memory on the device.

free_ram()
Returns the amount of free RAM memory available on the device.

total_rom()
Returns the amount of read-only ROM memory on the device.

ring_type()
Not supported in 1st Edition! Returns the current ringing type as a string, which can be one of the following: 'normal', 'ascending', 'ring_once', 'beep', or 'silent'.

os_version()
Returns the operating system version number of the device as a three element tuple (major version, minor version, build number). The elements are as follows:

- The major version number, ranging from 0 to 127 inclusive
- The minor version number, ranging from 0 to 99 inclusive
- The build number, ranging from 0 to 32767 inclusive.

signal_bars()
Returns the current network signal strength ranging from 0 to 7, with 0 meaning no signal and 7 meaning a strong signal. If using an emulator, value 0 is always returned.

signal_dbm()
Returns the current network signal strength in dBm. This is available SDK 2.8 onwards. If using an emulator value 0 is always returned.

sw_version()
Returns the software version as a Unicode string. On the emulator, returns the hardcoded string u' emulator'. For example, a software version can be returned as u'V 4.09.1 26-02-04 NHL-10 (c) NMP'.

---

1 Descriptions for these values are based on information found in S60 SDK documentation [4].
5.1 appuifw — Interface to the S60 GUI framework

The appuifw module offers an interface to the S60 UI application framework. Figure 5.1 provides an overview of the Python for S60 environment for UI application programming.

Note: The services of this interface may only be used in the context of the main thread, that is, the initial thread of a UI application script.

5.1.1 Basics of appuifw Module

Figure 5.2 shows the layout of a S60 application UI in the normal screen mode and a summary of how it relates to the services available at the appuifw API. For alternative layouts, see Figure 5.3.

The main application window may be set up to be occupied by a UI control.

A multi-view application can show the different views as tabs in the navigation pane and react as the users navigate between tabs.

Dialogs always take precedence over the usual UI controls and appear on top of them.

UI controls are implemented as Python types. These types are available:

- Text
- Listbox
- Canvas

UI controls appear on the screen as soon as an instance of the corresponding Python type is set to the body field (app.body) of the current application UI.

Form is a versatile dialog implemented as a type.

The Content_handler type facilitates interfacing to other UI applications and common high-level UI components. It is based on the notion that designated handlers can reduce UI application interaction to operations on MIME-type content.

The following dialogs are implemented as functions:

- note
- query
- multi_query
- selection_list
- multi_selection_list
Figure 5.1: Python for S60 UI environment overview

Figure 5.2: The different parts of the screen when using the 'normal' layout
A dialog becomes visible as soon as the corresponding Python function has been called. The function returns with the eventual user input or information on the cancellation of the dialog. Form is an exception; it is shown when its execute method is called.

5.1.2 Softkeys

The softkeys are managed by the underlying S60 Platform. When no dialog is visible, the right softkey is bound to application exit and the left one represents an Options menu. Python for S60 offers an interface for manipulating the menu and for binding the Exit key to a Python-callable object (see Section 5.1.4).

The native code that implements a dialog also manages the softkeys of the dialog, typically OK and Cancel. When the user input needs to be validated before accepting it and dismissing the dialog, it is best to use Form.

5.1.3 Module Level Functions

The following free functions - functions that do not belong to any class - are defined in the appuifw module:

available_fonts()

Returns a list (Unicode) of all fonts available in the device.

query(label, type[, initial_value])

Performs a query with a single-field dialog. The prompt is set to label, and the type of the dialog is defined by type. The value of type can be any of the following strings:

- 'text'
- 'code'
- 'number'
- 'date'
- 'time'
- 'query'
- 'float'

The type of the optional initial_value parameter and the returned input depend on the value of type:
• For text fields, (‘text’, ‘code’) it is Unicode
• For number fields, it is numeric
• For date fields, it is seconds since epoch rounded down to the nearest local midnight

A simple confirmation query and time query take no initial value and return True/None and seconds since local midnight, correspondingly. All queries return None if the users cancel the dialog.

For ‘float’ query the initial_value setting has no effect.

multi_query(label_1, label_2)
A two-field text (Unicode) input dialog. Returns the inputted values as a 2-tuple. Returns None if the users cancel the dialog.

note(text, type=[], global=[])
Displays a note dialog of the chosen type with text (Unicode). The default value for type is ‘info’, which is automatically used if type is not set. type can be one of the following strings: ‘error’, ‘info’, or ‘conf’.

If global (integer) is any other value than zero a global note is displayed. A global note is displayed even if the Python application calling this function is in background. The same set of types is supported as in standard note.

popup_menu(list, label)
A pop-up menu style dialog. list representing the menu contents can be a list of Unicode strings or a list of Unicode string pairs (tuples). The resulting dialog list is then a single-style or a double-style list. A single-style list is shown in full; whereas a double-style list shows the items one at a time. Returns None if the user cancels the operation.

selection_list(choices, search_field=0)
Executes a dialog that allows the users to select a list item and returns the index of the chosen item, or None if the selection is cancelled by the users. choices is a list of Unicode strings. search_field is 0 (disabled) by default and is optional. Setting it to 1 enables a search field (find pane) that facilitates searching for items in long lists. If enabled, the search field appears after you press a letter key.

multi_selection_list(choices, style='checkbox', search_field=0)
Executes a dialog that allows the users to select multiple list items. Returns a tuple of indexes (a pair of Unicode strings) of the chosen items, or empty tuple if the selection is cancelled by the users. choices is a list of Unicode strings. style is an optional string; the default value being ‘checkbox’. If ‘checkbox’ is given, the list will be a checkbox list, where empty checkboxes indicate what items can be marked. The other possible value that can be set for style is ‘checkmark’. If ‘checkmark’ is given, the list will be a markable list, which lists items but does not indicate specifically that items can be selected. To select items on a markable list, use the Navigation key to browse the list and the Edit key to select an item. For example views on checkbox and markable lists, see Figure 5.4. search_field is 0 (disabled) by default and is optional. Setting it to 1 enables a search field (find pane) that facilitates searching for items in long lists. If enabled, the search field is always visible with checkbox lists; with markable lists it appears by pressing a letter key.

Example:

tuple = appuifw.multi_selection_list(L, style='checkmark', search_field=1)

5.1.4 Application Type

A single implicit instance of this type always exists when appuifw module is present and can be referred to with the name app. New instances cannot be created by a Python program.

class Application

Instances of Application type have the following attributes:
Figure 5.4: Examples of a checkbox list (left) and a markable list (right)

body
The UI control that is visible in the application’s main window. Currently either Text, a Listbox object, Canvas, or None.

exit_key_handler
A callable object that is called when the user presses the Exit softkey. Setting exit_key_handler to None sets it back to the default value.

menu
This is a list of the following kinds of items:
- \((title, \text{callback})\) which creates a regular menu item
- \((title, ((title, \text{callback})[...]))\) which creates a submenu

*title* (Unicode) is the name of the item and *callback* the associated callable object. The maximum allowed number of items in a menu, or items in a submenu, or submenus in a menu is 30.

Example:

```python
appuifw.app.menu = [(u"Item 1", item1),
                    (u"Submenu 1",
                     [(u"Subitem 1", subitem1),
                      (u"Subitem 2", subitem2)])]
```

screen
The screen area used by an application. See Figure 5.3 for example screens. The appearance of the application on the screen can be affected by setting one of the following values: ‘normal’, ‘large’, and ‘full’.

Examples:

```python
appuifw.app.screen='normal' # (a normal screen with title pane and softkeys)
appuifw.app.screen='large'  # (only softkeys visible)
appuifw.app.screen='full'   # (a full screen)
```

title
The title the application that is visible in the application’s title pane. Must be Unicode.

focus
A callable object that is called with integer as parameter \((0 =\text{focus lost}, 1 = \text{focus regained})\) when the application receives focus or it is switched to background. Focus is received e.g. when the application is switched from background to foreground or when the focus is regained from screensaver. Similarly when the screensaver is displayed, focus is lost.

Examples:

```python
>>> import appuifw
>>> def cb(fg):
```

5.1. appuifw — Interface to the S60 GUI framework
... if(fg):
...     print "foreground"
... else:
...     print "background"
...
>>> appuifw.app.focus=cb
>>> # switch to background, following text is printed from callback:
>>> background
>>> # switch to foreground, following text is printed from callback:
>>> foreground

Note: An improper callback can cause adverse effects. If you, for example, define a callback
which takes no parameters you will receive never-ending TypeError exceptions on the Nokia
6600.

orientation
Available only for S60 3rdEd. The orientation of the application. The orientation of the appli-
cation can be one of the following values: 'automatic' (this is the default value), 'portrait'
or 'landscape'.

Instances of Application type have the following methods:
activate_tab(index)
    Activates the tab index counting from zero.

full_name()
    Returns the full name, in Unicode, of the native application in whose context the current
    Python interpreter session runs.

uid()
    Returns the UID, in Unicode, of the native application in whose context the current Python
    interpreter session runs.

set_exit()
    Requests a graceful exit from the application as soon as the current script execution returns.

set_tabs(tab_texts[,callback=None ])
    Sets tabs with given names on them in the navigation bar; tab_texts is a list of Unicode
    strings. When the users navigate between tabs, callback gets called with the index of the
    active tab as an argument. Tabs can be disabled by giving an empty or one-item tab_texts
    list.

layout(layout_id)
    Note: Available from S60 2ndEd FP3 onwards (inclusive).
    Returns as a tuple the size and the position of the requested layout_id. The logical lay-
    outs are outlined partly in Figure 5.2. The position is given from the top left corner. The
    layout_id can be one of the constants defined in module appuifw:
    EScreen
        Screen.
    EApplicationWindow
        Window that fills the entire screen.
    EStatusPane
        Indicates common components for most of the applications.
    EMainPane
        The application main pane is used in all the applications.
    EControlPane
        Control pane.
    ESignalPane
        The signal pane is used to indicate signal strength.
    EContextPane
        The context pane is used to indicate an active application.
    ETitlePane
        Used to indicate the subject or the name of the main pane content.

1Descriptions of the values are from the S60 SDK documentation [4].
EBatteryPane
The battery pane is used to indicate battery strength.

EUniversalIndicatorPane
The universal indicator pane is used to indicate items that require the user’s attention while browsing applications.

ENaviPane
The navi pane is used to indicate navigation within an application, to provide context sensitive information to the user while entering or editing data, or to show additional information.

EFindPane
A fixed find pane is used with lists instead of the find pop-up window.

EWallpaperPane
Wallpaper pane.

EIndicatorPane
The universal indicator pane is used to indicate items that require the user’s attention while browsing applications.

EAColumn
Used generally to display small sized graphics or heading texts.

EBColumn
Used generally to display large sized icons or heading texts.

ECColumn
Used generally to display data entered by the user. Overlaps with the D column.

EDColumn
Used generally to display additional icons. Overlaps with the C column.

ESTaconTop
Top part of status and control panes in landscape layout.

ESTaconBottom
Bottom part of status and control panes in landscape layout.

EStatusPaneBottom
Bottom part of status pane in landscape layout.

EControlPaneBottom
Bottom part of control pane in landscape layout.

EControlPaneTop
Top part of control pane in landscape layout.

EStatusPaneTop
Top part of status pane in landscape layout.

Example:

```python
>>> import appuifw
>>> appuifw.app.layout(appuifw.EMainPane)
((176, 144), (0, 44))
>>> # size and position (x, y) of the main pane in Nokia N70
```

5.1.5 Form Type

Form implements a dynamically configurable, editable multi-field dialog. Form caters for advanced dialog use cases with requirements such as free selectability of the combination of fields, possibility of validating the user input, and automatically producing the contents of some dialog fields before allowing the closing of the dialog.

```python
class Form(fields=[], flags=0)

Creates a Form instance. fields is a list of field descriptors: (label, type[, value]) where label is a Unicode string

type is one of the following strings: 'text', 'number', 'date', 'time', 'combo' or 'float'
```
value, depending on type: Unicode string, numeric, float (seconds since Unix epoch rounded down to the nearest local midnight), float (seconds since local midnight), ([choice_label ...], index) of float. For ‘float’ type the initial value setting might not be shown in the UI.

Form can also be configured and populated after construction. The configuration flags are visible as an attribute. Form implements the list protocol that can be used for setting the form fields, as well as obtaining their values after the dialog has been executed.

Instances of Form type have the following attributes:

flags
This attribute holds the values of the various configuration flags. Currently supported flags are:

FFormEditModeOnly
When this flag is set, the form remains in edit mode while execute runs.

FFormViewModeOnly
When this flag is set, the form cannot be edited at all.

FFormAutoLabelEdit
This flag enables support for allowing the end-users to edit the labels of the form fields.

FFormAutoFormEdit
This flag enables automatic support for allowing the end-users to add and delete the form fields. Note that this is an experimental feature and is not guaranteed to work with all SDK versions.

FFormDoubleSpaced
When this flag is set, double-spaced layout is applied when the form is executed: one field takes two lines, as the label and the value field are on different lines.

menu
A list of (title, callback) pairs, where each pair describes an item in the form’s menu bar that is active while the dialog is being executed. title (Unicode) is the name of the item and callback the associated callable object.

save_hook
This attribute can be set to a callable object that receives one argument and returns a Boolean value. It gets called every time the users want to save the contents of an executing Form dialog. A candidate list for new form content - a list representing the currently visible state of the UI - is given as an argument. The list can be modified by save_hook. If save_hook returns True, the candidate list is set as the new contents of the form. Otherwise, the form UI is reset to reflect the field list contained in Form object.

Instances of Form type have the following methods:

execute()
Executes the dialog by making it visible on the UI.

insert(index, field_descriptor)
Inserts the field descriptor into the Form before the given index.

pop()
Removes the last field descriptor from the Form and returns it.

length()
The number of field descriptors in the form.

The subscript notation f[i] can be used to access or modify the i-th element of the form f. Same limitations as discussed above in the context of the flag FFormAutoFormEdit apply to modifying a form while it is executing. The ability to change the schema of a form while it is executing is an experimental feature.

5.1.6 Text Type

Text is a text editor UI control. For examples on the options available with Text, see Figure 5.5.
Instances of **Text** type have the following attributes:

**color**
The color of the text. **color** supports the same color representation models as the **graphics** module. For the supported color representation models, see Section 5.2.

**focus**
A Boolean attribute that indicates the focus state of the control. Editor control also takes the ownership of the navigation bar, and this feature is needed to enable the usage of this control in applications that use the navigation bar - for example, navigation tabs.

**font**
The font of the text. There are two possible ways to set this attribute:

- Using a supported Unicode font, for example `u"Latin12"`. Trying to set a font which is not supported by the device has no effect. A list of supported fonts can be retrieved by using `appuifw.available_fonts`.

  Example, setting font:
  ```python
  t = appuifw.Text()
  t.font = u"albi17b" # sets font to Albi 17 bold
  t.font = u"LatinPlain12" # sets font to Latin Plain 12
  ```

- Using one of the default device fonts that are associated with the following labels (plain strings): 'annotation', 'title', 'legend', 'symbol', 'dense', 'normal'

  Example, setting font:
  ```python
  t.font = "title" # sets font to the one used in titles
  ```

  Example, checking the currently set font:
  ```python
  unicodeFont = t.font
  ```

  The attribute value retrieved is always a Unicode string. If the font has been set with a label, for example, 'title', the attribute will retrieve the font associated with that label.

**highlight_color**
The highlight color of the text. **highlight_color** supports the same color representation models as the **graphics** module. For the supported color representation models, see Section 5.2.

**style**
The style of the text. The flags for this attribute are defined in the **appuifw** module. These flags can be combined by using the binary operator `|`. The flags can be divided into two types: text style and text highlight. Text style flags can be freely combined with each other. However, one or more text style flags can be combined with only one text highlight flag. The flags are:

  - Text style:
  - Text highlight:
STYLE_BOLD
    Enables bold text.
STYLE_UNDERLINE
    Enables underlined text.
STYLE_ITALIC
    Enables italic text.
STYLE_STRIKETHROUGH
    Enables strikethrough.

Text highlight:

HIGHLIGHT_STANDARD
    Enables standard highlight.
HIGHLIGHT_ROUNDED
    Enables rounded highlight.
HIGHLIGHT_SHADOW
    Enables shadow highlight.

Only one highlight is allowed to be used at once. Therefore, it is possible to combine only one highlight with one or more text styles.

Examples:

    t = appuifw.Text()

    # These and other similar values and combinations are valid:
    t.style = appuifw.STYLE_BOLD
    t.style = appuifw.STYLE_UNDERLINE
    t.style = appuifw.STYLE_ITALIC
    t.style = appuifw.STYLE_STRIKETHROUGH
    t.style = (appuifw.STYLE_BOLD|
               appuifw.STYLE_ITALIC|
               appuifw.STYLE_UNDERLINE)

    # These values are valid:
    t.style = appuifw.HIGHLIGHT_STANDARD
    t.style = appuifw.HIGHLIGHT_ROUNDED
    t.style = appuifw.HIGHLIGHT_SHADOW

    # This combination is NOT valid:
    # Invalid code, do not try!
    t.style = (appuifw.HIGHLIGHT_SHADOW|appuifw.HIGHLIGHT_ROUNDED)

Instances of Text type have the following methods:

add(text)
    Inserts the Unicode string text to the current cursor position.

bind(event_code, callback)
    Binds the callable Python object callback to event event_code. The key codes are defined in the
dependent_events library module. The call bind(event_code, None) clears an existing binding. In the
current implementation the event is always passed also to the underlying native UI control.

clear()
    Clears the editor.

delete([pos=0, length=len()])
    Deletes length characters of the text held by the editor control, starting from the position pos.

get_pos()
    Returns the current cursor position.

len()
    Returns the length of the text string held by the editor control.
get([pos=0, length=len()])

Retrieves length characters of the text held by the editor control, starting from the position pos.

set(text)

Sets the text content of the editor control to Unicode string text.

set_pos(cursor_pos)

Sets the cursor to cursor_pos.

5.1.7 Listbox Type

An instance of this UI control type is visible as a listbox, also known as a list in Symbian, that can be configured to be a single-line item or a double-item listbox. Figure 5.6 shows a single-line item Listbox with icons. For more information on the MBM and MIF formats, see Section 5.1.8.

class Listbox(list, callback)

Creates a Listbox instance. A callable object callback gets called when a listbox selection has been made. list defines the content of the listbox and can be one of the following:

- A normal (single-line item) listbox: a list of Unicode strings, for example [unicode_string item1, unicode_string item2]
- A double-item listbox: a two-element tuple of Unicode strings, for example [(unicode_string item1, unicode_string item1description), (unicode_string item2, unicode_string item2description)]
- A normal (single-line item) listbox with graphics: a two-element tuple consisting of a Unicode string and an Icon object, for example [(unicode_string item1, icon1), (unicode_string item2, icon2)].
- A double-item listbox with graphics: a three-element tuple consisting of two Unicode strings and one Icon object, for example [(unicode_string item1, unicode_string item1description, icon1), (unicode_string item2, unicode_string item2description, icon2)]

Example: To produce a normal (single-line item) listbox with graphics:

```
icon1 = appuifw.Icon(u"z:\system\data\avkon.mbm", 28, 29)
icon2 = appuifw.Icon(u"z:\system\data\avkon.mbm", 40, 41)
entries = [(u"Signal", icon1),
           (u"Battery", icon2)]
lb = appuifw.Listbox(entries, lbox_observe)
```

Instances of Listbox type have the following methods:
**bind(event_code, callback)**

Binds the callable Python object `callback` to event `event_code`. The key codes are defined in the `key_codes` library module. The call `bind(event_code, None)` clears an existing binding. In the current implementation the event is always passed also to the underlying native UI control.

**current()**

Retruns the currently selected item’s index in the `Listbox`.

**set_list(list[, current])**

Sets the `Listbox` content to a list of Unicode strings or a list of tuples of Unicode strings. The accepted structures of `list` are the same as in the `Listbox` constructor. The optional argument `current` is the index of the focused list item.

### 5.1.8 Icon Type

An instance of `Icon` type encapsulates an icon to be used together with a `Listbox` instance. Note that currently `Icon` can only be used with `Listbox` (see Section 5.1.7).

MBM is the native Symbian OS format used for pictures. It is a compressed file format where the files can contain several bitmaps and can be referred to by a number. An `.mbg` file is the header file usually associated with an `.mbm` file, which includes symbolic definitions for each bitmap in the file. For example, an ‘avkon.mbm’ file has an associated index file called ‘avkon.mbg’, which is included in S60 SDKs. For more information on the MBM format and the bitmap converter tool, see [4] and search the topics with the key term ”How to provide Icons”; this topic also points you to the Bitmap Converter tool that can be used for converting bitmaps into the MBM format.

S60 2nd Edition FP3 introduces a new format for icons called Multi-Image File (MIF). This format is very similar to the MBM format and also contains several compressed files. The files to be compressed should be in Scalable Vector Graphics Tiny (SVG-T) format. For more information on the SVG format, see Scalable Vector Graphics (SVG) 1.1 Specification [10].

**class Icon(filename, bitmap, bitmapMask)**

Creates an icon. `filename` is a Unicode file name and must include the whole path. Note that MBM and MIF (MIF only in S60 2nd Edition FP3) are the only file formats supported. `bitmap` and `bitmapMask` are integers that represent the index of the icon and icon mask inside that file respectively.

Example: The following builds an icon with the standard signal symbol:

```python
icon = appuifw.Icon(u"z:\system\data\avkon.mbm", 28, 29)
```

### 5.1.9 Content_handler Type

An instance of `Content_handler` handles data content by its MIME type.

**class Content_handler([callback])**

Creates a `Content_handler` instance. A `Content_handler` handles data content by its MIME type. The optional `callback` is called when the embedded handler application started with the `open` method finishes.

Instances of `Content_handler` type have the following methods:

**open(filename)**

Opens the file `filename` (Unicode) in its handler application if one has been registered for the particular MIME type. The handler application is embedded in the caller’s thread. The call to this function returns immediately. When the handler application finishes, the `callback` that was given to the `Content_handler` constructor is called.

**open_standalone(filename)**

Opens the file `filename` (Unicode) in its handler application if one has been registered for the particular MIME type. The handler application is started in its own process. The call to this function returns immediately. Note that `callback` is not called for applications started with this
method.

5.1.10 Canvas Type

Canvas is a UI control that provides a drawable area on the screen and support for handling raw key events. Canvas supports the standard drawing methods that are documented in Section 5.2.

```
class Canvas([redraw_callback=None, event_callback=None, resize_callback=None])
```

Constructs a Canvas. The optional parameters are callbacks that are called when specific events occur.

**Note:** Watch out for cyclic references here. For example, if the callbacks are methods of an object that holds a reference to the Canvas, a reference cycle is formed that must be broken at cleanup time or the Canvas will not be freed.

*redraw_callback* is called whenever a part of the Canvas has been obscured by something, is then revealed, and needs to be redrawn. This can typically happen, for example, when the user switches away from the Python application and back again, or after displaying a pop-up menu. The callback takes as its argument a four-element tuple that contains the top-left and the bottom-right corner of the area that needs to be redrawn. In many cases redrawing the whole Canvas is a reasonable option.

*event_callback* is called whenever a raw key event is received. There are three kinds of key events: EEventKeyDown, EEventKey, and EEventKeyUp. When a user presses a key down, events EEventKeyDown and EEventKey are generated. When the key is released, an EEventKeyUp event is generated.

The argument to the *event_callback* is a dictionary that contains the following data for key events:

- `'type'`: one of EEventKeyDown, EEventKey, or EEventKeyUp
- `'keycode'`: the keycode of the key
- `'scancode'`: the scancode of the key
- `'modifiers'`: the modifiers that apply to this key event

Each key on the keyboard has one or more scancodes and zero or more keycodes associated with it. A scancode represents the physical key itself and a keycode is the result of state-related operating system defined processing done on the key. For keys that correspond to a symbol in the current character set of the phone, the keycode is equal to the code of the corresponding symbol in that character set. For example, if you are using the Nokia Wireless Keyboard (SU-8W), pressing the key A will always produce the scan code 65 (ASCII code for an upper case A), but the keycode could be either 65 or 91 (ASCII code for a lower case A) depending on whether or not the Shift key is pressed or Caps Lock is active.

The *key_codes* module contains definitions for the keycodes and scancodes. See Figure 5.7 for the codes of the most common keys on the phone keypad.

Some keys are handled in a special way:

- A short press of the Edit key causes it to stay down, meaning that no EEventKeyUp event is sent. The event is only sent after a long press.
- Detecting presses of the Voice tags key or the Power key is not supported.
- If the right softkey is pressed, the `appuifw.app.exit_key_handler` callback is always executed.

There is no way to prevent the standard action of the Hang-up key, the Menu key, the Power key or the Voice tags key from taking place.

*resize_callback* is called when screen size is changed when the Canvas rect size has been changed. The callback takes as its argument a two-element tuple that contains the new clientRect width and height.
<table>
<thead>
<tr>
<th>Key</th>
<th>Keycode</th>
<th>Scancode</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>EKeyLeftSoftkey</td>
<td>EScancodeLeftSoftkey</td>
</tr>
<tr>
<td>2.</td>
<td>EKeyYes</td>
<td>EScancodeYes</td>
</tr>
<tr>
<td>3.</td>
<td>EKeyMenu</td>
<td>EScancodeMenu</td>
</tr>
<tr>
<td>4.</td>
<td>EKey0...9</td>
<td>EScancode0...9</td>
</tr>
<tr>
<td>5.</td>
<td>EKeyStar</td>
<td>EScancodeStar</td>
</tr>
<tr>
<td>6.</td>
<td>EKeyLeftArrow</td>
<td>EScancodeLeftArrow</td>
</tr>
<tr>
<td>7.</td>
<td>EKeyUpArrow</td>
<td>EScancodeUpArrow</td>
</tr>
<tr>
<td>8.</td>
<td>EKeySelect</td>
<td>EScancodeSelect</td>
</tr>
<tr>
<td>9.</td>
<td>EKeyRightArrow</td>
<td>EScancodeRightArrow</td>
</tr>
<tr>
<td>10.</td>
<td>EKeyDownArrow</td>
<td>EScancodeDownArrow</td>
</tr>
<tr>
<td>11.</td>
<td>EKeyRightSoftkey</td>
<td>EScancodeRightSoftkey</td>
</tr>
<tr>
<td>12.</td>
<td>EKeyNo</td>
<td>EScancodeNo</td>
</tr>
<tr>
<td>13.</td>
<td>EKeyBackspace</td>
<td>EScancodeBackspace</td>
</tr>
<tr>
<td>14.</td>
<td>EKeyEdit</td>
<td>EScancodeEdit</td>
</tr>
<tr>
<td>15.</td>
<td>EKeyHash</td>
<td>EScancodeHash</td>
</tr>
</tbody>
</table>

Figure 5.7: Keycodes and scancodes for phone keys usable from Python applications
Instances of Canvas type have the following attribute:

**size**

A two-element tuple that contains the current width and height of the Canvas as integers.

Instances of Canvas type have the same standard drawing methods that are documented in Section 5.2.

### 5.2 graphics — A graphics related services package

The graphics module provides access to the graphics primitives and image loading, saving, resizing, and transformation capabilities provided by the Symbian OS.

The module is usable from both graphical Python applications and background Python processes. However, background processes have some restrictions, namely that plain string symbolic font names are not supported in background processes since background processes have no access to the UI framework (see also Section 5.2.4).

For an example on using this module, see [6].

Functions Image.open and Image.inspect and Image object methods load, save, resize, and transpose are not available for S60 1st Edition.

#### 5.2.1 Module Level Functions

The following free functions - functions that do not belong to any class - are defined in the graphics module:

**screenshot()**

Takes a screen shot and returns the image in Image format.

#### 5.2.2 Image Class Static Methods

The following Image class static methods are defined in the graphics module:

**Image.new(size[, mode='RGB16'])**

Creates and returns a new Image object with the given size and mode. size is a two-element tuple. mode specifies the color mode of the Image to be created. It can be one of the following:

- '1': Black and white (1 bit per pixel)
- 'L': 256 gray shades (8 bits per pixel)
- 'RGB12': 4096 colors (12 bits per pixel)
- 'RGB16': 65536 colors (16 bits per pixel)
- 'RGB': 16.7 million colors (24 bits per pixel)

**Image.open(filename)**

*Note:* Not supported in S60 1st Edition!

Returns a new Image object (mode RGB16) that contains the contents of the named file. The supported file formats are JPEG and PNG. The file format is automatically detected based on file contents. filename should be a full path name.

**Image.inspect(filename)**

*Note:* Not supported in S60 1st Edition!

Examines the given file and returns a dictionary of the attributes of the file. At present the dictionary contains only the image size in pixels as a two-element tuple, indexed by key 'size'. filename should be a full path name.
5.2.3 Image Objects

An Image object encapsulates an in-memory bitmap.

Note on asynchronous methods: Methods `resize`, `transpose`, `save`, and `load` have an optional callback argument. If the callback is not given, the method call is synchronous; when the method returns, the operation is complete or an exception has been raised. If the callback is given, the method calls are asynchronous. If all parameters are valid and the operation can start, the method call will return immediately. The actual computation then proceeds in the background. When it is finished, the callback is called with an error code as the argument. If the given code is 0, the operation completed without errors, otherwise an error occurred.

It is legal to use an unfinished image as a source in a blit operation; this will use the image data as it is at the moment the blit is made and may thus show an incomplete result.

Image objects have the following methods:

```markdown
resize(newsize[, callback=None, keepaspect=0])
```

**Note:** Not supported in S60 1st Edition!

Returns a new image that contains a resized copy of this image. If `keepaspect` is set to 1, the resize will maintain the aspect ratio of the image, otherwise the new image will be exactly the given size. If `callback` is given, the operation is asynchronous, and the returned image will be only partially complete until `callback` is called.

```markdown
transpose(direction[, callback=None])
```

**Note:** Not supported in S60 1st Edition!

Creates a new image that contains a transformed copy of this image. The `direction` parameter can be one of the following:

- **FLIP_LEFT_RIGHT:** Flips the image horizontally, exchanging left and right edges.
- **FLIP_TOP_BOTTOM:** Flips the image vertically, exchanging top and bottom edges.
- **ROTATE_90:** Rotates the image 90 degrees counterclockwise.
- **ROTATE_180:** Rotates the image 180 degrees.
- **ROTATE_270:** Rotates the image 270 degrees counterclockwise.

If `callback` is given, the operation is asynchronous and the returned image will be only partially complete until `callback` is called.

```markdown
load(filename[, callback=None])
```

**Note:** Not supported in S60 1st Edition!

Replaces the contents of this Image with the contents of the named file, while keeping the current image mode. This Image object must be of the same size as the file to be loaded.

If `callback` is given, the operation is asynchronous and the loaded image will be only partially complete until `callback` is called. `filename` should be a full path name.

```markdown
save(filename[, callback=None, format=None, quality=75, bpp=24, compression='default'])
```

**Note:** Not supported in S60 1st Edition!

Saves the image into the given file. The supported formats are JPEG and PNG. If `format` is not given or is set to `None`, the format is determined based on the file name extension: `.jpg` or `.jpeg` are interpreted to be in JPEG format and `.png` to be in PNG format. `filename` should be a full path name.

When saving in JPEG format, the `quality` argument specifies the quality to be used and can range from 1 to 100.

When saving in PNG format, the `bpp` argument specifies how many bits per pixel the resulting file should have, and `compression` specifies the compression level to be used.

Valid values for `bpp` are:
1: Black and white, 1 bit per pixel
8: 256 gray shades, 8 bits per pixel
24: 16.7 million colors, 24 bits per pixel

Valid values for compression are:

- 'best': The highest possible compression ratio, the slowest speed
- 'fast': The fastest possible saving, moderate compression
- 'no': No compression, very large file size
- 'default': Default compression, a compromise between file size and speed

If callback is given, the operation is asynchronous. When the saving is complete, the callback is called with the result code.

stop()
Stops the current asynchronous operation, if any. If an asynchronous call is not in progress, this method has no effect.

Image objects have the following attribute:

size
A two-element tuple that contains the size of the Image. Read-only.

5.2.4 Common Features of Drawable Objects

Objects that represent a surface that can be drawn on support a set of common drawing methods, described in this section. At present there are two such objects: Canvas from the appuifw module and Image from the graphics module.

Options

Many of these methods support a set of standard options. This set of options is as follows:

- outline: The color to be used for drawing outlines of primitives and text. If None, the outlines of primitives are not drawn.
- fill: The color to be used for filling the insides of primitives. If None, the insides of primitives are not drawn. If pattern is also specified, fill specifies the color to be used for areas where the pattern is white.
- width: The line width to be used for drawing the outlines of primitives.
- pattern: Specifies the pattern to be used for filling the insides of primitives. If given, this must be either None or a 1-bit (black and white) Image.

Coordinate representation

The methods accept an ordered set of coordinates in the form of a coordinate sequence. Coordinates can be of type int, long, or float. A valid coordinate sequence is a non-empty sequence of either

- Alternating x and y coordinates. In this case the sequence length must be even, or
- Sequences of two elements, that specify x and y coordinates.

Examples of valid coordinate sequences:

- (1, 221L, 3, 4, 5.85, -3): A sequence of three coordinates
• \([(1,221L),(3,4),(5,12,6)]\): A sequence of three coordinates
• (1,5): A sequence of one coordinate
• \([(1,5)]\): A sequence of one coordinate
• \([1,5]\): A sequence of one coordinate

Examples of invalid coordinate sequences:

Invalid code, do not use!

• []: An empty sequence
• (1,2,3): Odd number of elements in a flat sequence
• \([(1,2),(3,4),\text{None}]\): Contains an invalid element
• \([(1,2),3,4]\): Mixing the flat and nested form is not allowed

Color representation

All methods that take color arguments accept the following two color representations:

• A three-element tuple of integers in the range from 0 to 255 inclusive, representing the red, green, and blue components of the color.
• An integer of the form 0xrrggb, where rr is the red, gg the green, and bb the blue component of the color.

For 12 and 16 bit color modes the color component values are simply truncated to the lower bit depth. For the 8-bit grayscale mode images the color is converted into grayscale using the formula \((2*r+5*g+b)/8\), rounded down to the nearest integer. For 1-bit black and white mode images the color is converted into black (0) or white (1) using the formula \((2*r+5*g+b)/1024\).

Examples of valid colors:

• 0xffff00: Bright yellow
• 0x004000: Dark green
• (255,0,0): Bright red
• 0: Black
• 255: Bright blue
• (128,128,128): Medium gray

Examples of invalid colors:

Invalid code, do not use!

• (0,0.5,0.9): Floats are not supported
• ‘#ff80c0’: The HTML color format is not supported
• (-1,0,1000): Out-of-range values
• (1,2): The sequence is too short
• [128,128,192]: This is not a tuple
Font specifications

A font can be specified in three ways:

- None, meaning the default font
- a Unicode string that represents a full font name, such as u’LatinBold19’
- a plain string symbolic name that refers to a font setting currently specified by the UI framework
- as a two or three element tuple, where
  - the first element is the font name (unicode or string) or None for default font
  - the second element is the font height in pixels or None for default size
  - the third (optional) element is the flags applied to the font or None for default options.

The flags are the following:

- FONT_BOLD bold
- FONT_ITALIC italic
- FONT_SUBSCRIPT subscript
- FONT_SUPERSCRIPT superscript
- FONT_ANTIALIAS forces the font to be antialiased
- FONT_NO_ANTIALIAS forces the font to not be antialiased

You can combine the flags with the binary or operator “—”. For example, the flags setting FONT_BOLD|FONT_ITALIC will produce text that is both bold and italic.

Note: Antialiasing support is only available for scalable fonts.

You can obtain a list of all available fonts with the appuifw module function available_fonts.

The symbolic names for UI fonts are:

- 'normal'
- 'dense'
- 'title'
- 'symbol'
- 'legend'
- 'annotation'

Since background processes have no access to the UI framework, these symbolic names are not supported in them. You need to specify the full font name.

Common Methods of Drawable Objects

\begin{align*}
\text{line}(\textit{coordseq}[\text{, } \langle \textit{options} \rangle]) & \quad \text{Draws a line connecting the points in the given coordinate sequence. For more information about the choices available for options, see Section 5.2.4.} \\
\text{polygon}(\textit{coordseq}[\text{, } \langle \textit{options} \rangle]) & \quad \text{Draws a line connecting the points in the given coordinate sequence, and additionally draws an extra line connecting the first and the last point in the sequence. If a fill color or pattern is specified, the polygon is filled with that color or pattern. For more information about the choices available for options, see Section 5.2.4.}
\end{align*}
rectangle(coordseq[, <options>])

Draws rectangles between pairs of coordinates in the given sequence. The coordinates specify the
top-left and the bottom-right corners of the rectangle. The sequence must have an even number
of coordinates. For more information about the choices available for options, see Section 5.2.4.

ellipse(coordseq[, <options>])

Draws ellipses between pairs of coordinates in the given sequence. The coordinates specify the
top-left and bottom-right corners of the rectangle inside which the ellipse is contained. The sequence
must have an even number of coordinates. For more information about the choices available for
options, see Section 5.2.4.

pieslice(coordseq, start, end[, <options>])

Draws pie slices contained in ellipses between pairs of coordinates in the given sequence. The start
and end parameters are floats that specify the start and end points of pie slice as the starting and
ending angle in radians. The angle 0 is to the right, the angle pi/2 is straight up, pi is to the left
and-pi/2 is straight down. coordseq is interpreted the same way as for the ellipse method. For
more information about the choices available for options, see Section 5.2.4.

arc(coordseq, start, end[, <options>])

Draws arcs contained in ellipses between pairs of coordinates in the given sequence. The start
and end parameters are floats that specify the start and end points of pie slice as the starting and
ending angle in radians. The angle 0 is to the right, the angle pi/2 is straight up, pi is to the left
and-pi/2 is straight down. coordseq is interpreted the same way as for the ellipse method. For
more information about the choices available for options, see Section 5.2.4.

point(coordseq[, <options>])

Draws points in each coordinate in the given coordinate sequence. If the width option is set to
greater than 1, draws a crude approximation of a circle filled with the outline color in the locations.
Note that the approximation is not very accurate for large widths; use the ellipse method if you
need a precisely formed circle. For more information about the choices available for options, see
Section 5.2.4.

clear([color=0xffffff])

Sets the entire surface of the drawable to the given color, white by default.

text(coordseq, text[fill=0, font=None])

Draws the given text in the points in the given coordinate sequence with the given color (default
value is black) and the given font. The font specification format is described above.

measure_text(text[font=None, maxwidth=-1, maxadvance=-1])

Measures the size of the given text when drawn using the given font. Optionally you can specify
the maximum width of the text or the maximum amount the graphics cursor is allowed to move
(both in pixels).

Returns a tuple of three values:

• the bounding box for the text as a 4-tuple: (topleft-x, topleft-y, bottomright-x, bottomright-y)
• the number of pixels the graphics cursor would move to the right
• the number of characters of the text that fits into the given maximum width and advance

blit(image[, target=(0,0), source=((0,0),image.size), mask=None, scale=0])

Copies the source area from the given image to the target area in this drawable. The source area
is copied in its entirety if mask is not given or is set to None. If the mask is given, the source area
is copied where the mask is white. mask can be either None, a 1-bit (black and white) Image or
(on S60 2nd edition FP2 and later) a grayscale Image, and must be of the same size as the source
image. A grayscale mask acts as an alpha channel, i.e. partial transparency.

target and source specify the target area in this image and the source area in the given source. They
are coordinate sequences of one or two coordinates. If they specify one coordinate, it is interpreted
as the upper-left corner for the area; if they specify two coordinates, they are interpreted as the
top-left and bottom-right corners of the area.

If scale is other than zero, scaling is performed on the fly while copying the source area to the
target area. If scale is zero, no scaling is performed, and the size of the copied area is clipped to the smaller of source and target areas.

Note that a blit operation with scaling is slower than one without scaling. If you need to blit the same Image many times in a scaled form, consider making a temporary Image of the scaling result and blitting it without scaling. Note also that the scaling performed by the blit operation is much faster but of worse quality than the one done by the resize method, since the blit method does not perform any antialiasing.

5.3 camera — Interface for taking photographs

Note: Not available for S60 1st Edition.

The camera module enables taking photographs.

The camera module has the following functions:

- cameras_available()
  Returns the number of cameras available in the device.

- image_modes()
  Returns the image modes supported in the device as a list of strings, for example: ['RGB12', 'RGB', 'JPEG_IfExif', 'RGB16'].

- image_sizes()
  Returns the image sizes (resolution) supported in the device as a list of (x, y) tuples, for example: [(640, 480), (160, 120)].

- flash_modes()
  Returns the flash modes available in the device as a list of strings.

- max_zoom()
  Returns the maximum digital zoom value supported in the device as an integer.

- exposure_modes()
  Returns the exposure settings supported in the device as a list of strings.

- white_balance_modes()
  Returns the white balance modes available in the device as a list of strings.

- take_photo([mode, size, flash, zoom, exposure, white_balance, position])
  Takes a photograph and returns the image in:

  1. Image format (for more information on Image format, see Chapter 5.2 graphics Module) or
  2. Raw JPEG data.

The settings listed below describe all settings that are supported by the camera module. You can retrieve the mode settings available for your device by using the appropriate functions listed at the beginning of this chapter.

- mode is the display mode of the image. The default value is 'RGB16'. The following display modes are supported for the Image format pictures taken:
  - 'RGB12': 4096 colors (12 bits per pixel)
  - 'RGB16': 65536 colors (16 bits per pixel). Default value, always supported
  - 'RGB': 16.7 million colors (24 bits per pixel)

  For the JPEG data format images the following modes are supported:
  - 'JPEG_Exif': JPEG Exchangeable image file format
  - 'JPEG_JFIF': JPEG File Interchange Format

Descriptions for some of the values are based on information found in S60 SDK documentation.

For more information, see e.g. http://en.wikipedia.org/wiki/JPEG.
Note that there is variety between the devices and the supported formats.

- **size** is the resolution of the image. The default value is (640, 480). The following sizes are supported, for example, in Nokia 6630: (1280, 960), (640, 480) and (160, 120).

- **flash** is the flash mode setting. The default value is ‘none’. The following flash mode settings are supported:
  - ‘none’
    No flash. Default value, always supported
  - ‘auto’
    Flash will automatically fire when required
  - ‘forced’
    Flash will always fire
  - ‘fill_in’
    Reduced flash for general lighting
  - ‘red_eye_reduce’
    Red-eye reduction mode

- **zoom** is the digital zoom factor. It is assumed to be on a linear scale from 0 to the maximum zoom value allowed in the device. The default value is 0, meaning that zoom is not used.

- **exposure** is the exposure adjustment of the device. Exposure is a combination of lens aperture and shutter speed used in taking a photograph. The default value is ‘auto’. The following exposure modes are supported:
  - ‘auto’
    Sets exposure automatically. Default value, always supported
  - ‘night’
    Night-time setting for long exposures
  - ‘backlight’
    Backlight setting for bright backgrounds
  - ‘center’
    Centered mode for ignoring surroundings

- **white_balance** can be used to adjust white balance to match the main source of light. The term white balance refers to the color temperature of the current light. A digital camera requires a reference point to represent white. It will then calculate all the other colors based on this white point. The default value for **white_balance** is ‘auto’ and the following white balance modes are supported:
  - ‘auto’
    Sets white balance automatically. Default value, always supported
  - ‘daylight’
    Sets white balance to normal daylight
  - ‘cloudy’
    Sets white balance to overcast daylight
  - ‘tungsten’
    Sets white balance to tungsten filament lighting
  - ‘fluorescent’
    Sets white balance to fluorescent tube lighting
  - ‘flash’
    Sets white balance to flash lighting

- **position** is the camera used if the device, such as Nokia 6680, has several cameras. In Nokia 6680, the camera pointing to the user of the device is located in position 1, whereas the one pointing away from the user is located in position 0. The default **position** is 0.

If some other application is using the camera, this operation fails, with error SymbianError: KErrInUse. Invoking this function right after the device boot, might result in SymbianError: KErrNotReady error.
start_finder(*callable[, backlight_on=1, size=main-pane-size]*)

Starts the camera viewfinder and binds a callback to receive Image format feed. When a new viewfinder frame is ready the callback is invoked with the Image as parameter.

The optional parameter backlight_on determines whether the device backlight is kept on when the camera view finder is in operation. By default, the backlight is on (1 = on, 0 = off).

The optional parameter size (of type tuple, e.g. (176, 144)) can be used to change the size of the image received in the callback. The default size is the same as the application’s main pane size.

Example view finder code:

```python
>>> import appuifw
>>> import camera
>>> def cb(im):
...   appuifw.app.body.blit(im)
... >>> import graphics
>>> appuifw.app.body=appuifw.Canvas()
>>> camera.start_finder(cb)
```

stop_finder()

Stops the viewfinder.

release()

Releases the camera – After invocation other applications can access the camera hardware.

5.4 keycapture — Interface for global capturing of key events.

The keycapture module offers an API for global capturing of key events. The keycapture module provides the KeyCapturer object as a tool for listening the events.

The KeyCapturer object uses a callback method to report the key events. The callback method is called each time any of the specified keys is pressed.

Currently the keycapture module does not support capturing separate key-up or key-down events.

Note: Keycapture module requires capability SwEvent to work in 3rd Edition devices.

5.4.1 Module Level Constants

The following constants are defined in the keycapture module:

all_keys

A list of all key codes defined in the key_codes module.

5.4.2 KeyCapturer objects

KeyCapturer object takes a callback method as a mandatory parameter to its constructor. The callback method must have one single parameter for forwarding the key code of the captured key.

There can be several KeyCapturer objects existing at the same time.

KeyCapturer object has following methods and properties:

keys

List of keys to be captured. Can be read and written.

Example:
```python
defines (key_codes.EkeyUpArrow,)
defines keycapture.all_keys

forwarding
    Specifies whether captured key events are forwarded to other applications or not. Either has value 1 or 0. Can be read and written.

start()
    Starts the actual capturing of key events.

stop()
    Stops the actual capturing of key events.

last_key()
    Returns last key code that is captured.

5.5 topwindow — Interface for creating windows that are shown on top of other applications.

The topwindow module offers an API for creating windows that are shown on top of other applications and managing the content of these windows. Images can be inserted into the windows and the background color, visibility, corner type and shadow of the window can be manipulated.

topwindow extension does not provide sophisticated drawing capabilities by any means but rather relies on services provided by the graphics extension: topwindow allows graphics Image objects to be put into the windows that are represented by TopWindow objects.

TopWindow object provides mainly only two services: TopWindow objects can be shown or hidden and Images can be put into the windows. However, several images can be added into one TopWindow object and several TopWindow objects can be created and shown. Since the images can be manipulated using the graphics extension this makes it possible to create many kind of content to the TopWindow objects.

5.5.1 TopWindow objects

class TopWindow()
    Create a TopWindow object.

TopWindow objects have the following methods and properties:

show()
    Shows the window. The window is not shown until show() is called.

hide()
    Hides the window.

add_image(image, position)
    Inserts an image object graphics.Image into the window. The position of the image is specified by the (position) parameter. If only the coordinates of the top left corner are specified, like (x1, y1) the image is not resized. If four coordinates are given, like (x1, y1, x2, y2), the image is resized to fit to the specified area. Example:

```
remove_image(image)
remove_image(image, (10,10))
remove_image(image, (10,10,20,20))

position
Specifies the coordinates of the top left corner of the window. Can be read and written.
Example:

    position = (10, 20)

size
Specifies the size of the window. Can be read and written.
Example:

    size = (100, 200)

images
The images inserted into the window. Defined as a list of tuple objects. Each tuple contains a
graphics.Image object and the position of the image. The position may specify the top-left coordinate
of the image and optionally also the bottom-right coordinate of the image. Parameter (x,y) specifies
the top-left coordinate, but does not resize the image while parameter like (x1,y1,x2,y2) specifies
both the top-left and bottom-right coordinates and possibly also resizes the image. Can
be read and written. Also see the add_image() and remove_image() methods.
Example:

    images = [(image1,(x1,y1)), (image2,(x1,y1,x2,y2)), (image3,(50,50,100,100))]
sets the window content to be 3 images. image2 and image3 are possibly resized while the image1
is not)

shadow
Specifies if the shadow of the window is shown and the length of the shadow. Can be read and
can be written. Setting shadow = 0 makes the shadow invisible.
Example: shadow = 5

corner_type
Specifies the corner type of the window. Can be read and written. Corner type can be one of the
following values:

- square
- corner1
- corner2
- corner3
- corner5

Example: corner_type = square

maximum_size
Returns the maximum size of the window as a tuple (width, height). Read only property.

background_color
The background color of the window as an integer (e.g. 0xabcdef). The two greatest hexadecimal
digits specify the red, the next two specify the blue and the last ones specify the green color. Can
be read and written.
Example: background_color = 0xffffff (sets the white color)

visible
Can be set to 0 or 1. 1 means that window is visible, 0 means that it is not. Can be read and
written. Also see the show and hide methods.

5.5. topwindow — Interface for creating windows that are shown on top of other applications. 37
5.6 gles — Bindings to OpenGL ES

The gles module provides Python bindings to OpenGL ES 2D/3D graphics C API. OpenGL ES is a standard defined by Khronos Group (www.khronos.org). Currently S60 Python supports OpenGL ES version 1.0 from Series 60 version 2.6 onwards. Support for OpenGL ES version 1.1 should also become available in the near future, and both versions are documented here. OpenGL ES 1.1 will require Series 60 version 3.0 or newer.

For detailed description of the OpenGL ES API see the official specifications at http://www.khronos.org/opengles. This documentation contains only information that is specific to the S60 Python bindings to OpenGL ES. Where possible, the conventions of the PyOpenGL desktop OpenGL bindings (http://pyopengl.sourceforge.net) have been followed.

The display of OpenGL ES graphics is handled by separate module, glcanvas. See glcanvas module documentation for more information.

5.6.1 array type

gles module defines array type for representing numerical data of specific GL type. array objects are convenient when numerical data for OpenGL ES calls is specified in Python code. Class array also defines the standard Python sequence methods so its instances can be iterated and individual items in arrays can be manipulated easily.

class array(type, dimension, sequence)

Constructs a new array object that contains the given type of data that is taken from sequence. Parameter dimension specifies how many items there are in each array element. The dimension information is stored with the array and is used by those functions that need to know the element size of the input data, for example, if colors are specified with three or four components. The dimension does not affect the length of an array or its indexing: both are based on individual items.

Value of type must be one of the following: GL_FLOAT, GL_BYTE, GL_UNSIGNED_BYTE, GL_SHORT, GL_UNSIGNED_SHORT, or GL_FIXED.

The data in sequence is flattened before it is used to fill the array. When type is GL_FLOAT, the sequence can contains floats or integers. With all other types, sequence must only contain integers. Values in sequence are casted in C to the requested type, so if the requested type cannot properly represent all the values the results can be unexpected.

__len__()  
Returns the number of items in the array. Note that array dimension does not affect the calculation of the length.

__getitem__(index)  
Returns the item in array with index. Note that array dimension does not affect indexing.

__setitem__(index, value)  
Sets the value of the item in position index to value. Note that array dimension does not affect indexing.

5.6.2 Error handling

Errors generated by the API calls are handled similarly as in PyOpenGL: all GL errors are reported as Python exceptions of type gles.GLError. The wrapper code checks GL error status after each call automatically. There is no Python binding for glGetError call.

5.6.3 Differences to OpenGL ES C API

Certain OpenGL ES functions require special handling in Python, mainly because of the pointer parameters in the C API. Additionally, special Python versions for some OpenGL ES functions have been
added. Both sets of functions are documented below. If a function is not listed here its Python version should exactly match the C version defined in the official OpenGL ES 1.0 and 1.1 specifications.

OpenGL ES 1.0

`glColorPointer(size, type, stride, sequence)`
Parameter `sequence` must be either a `gles.array` object or some other Python sequence object. `gles.array` objects require less processing and can be therefore slightly faster. If `gles.array` object is used, the type and dimension of its data are ignored and `size` and `type` are used instead.

`glColorPointerub(sequence)`
Special Python version of `glColorPointer` that accepts either a `gles.array` object or some other Python sequence object. Other parameters of `glColorPointer` will be determined as follows:
- `size` If `sequence` is an instance of `gles.array`, its dimension is used; otherwise the length of `sequence` is used.
- `type` `GL_UNSIGNED_BYTE`
- `stride` 0

`glColorPointerf(sequence)`
Special Python version of `glColorPointer` that behaves exactly as `glColorPointerub` except `GL_FLOAT` is used as `type`.

`glColorPointerr(sequence)`
Special Python version of `glColorPointer` that behaves exactly as `glColorPointerub` except `GL_FIXED` is used as `type`.

`glCompressedTexImage2D(target, level, internalformat, width, height, border, imageSize, data)`
Parameter `data` must be either a `gles.array` or a Python string.

`glCompressedTexImage2D(target, level, internalformat, width, height, border, imageSize, data)`
Parameter `data` must be either a `gles.array` or a Python string.

`glDeleteTextures(sequence)`
Parameter `sequence` must be a Python sequence containing integers.

`glDrawElements(mode, count, type, indices)`
Parameter `indices` must be either a `gles.array` or some other Python sequence object. `gles.array` objects require less processing and can be therefore slightly faster. If `gles.array` object is used, the type of its data is ignored and `type` is used instead.

`glDrawElementsSub(mode, indices)`
Special Python version of `glDrawElements` that uses length of the sequence `indices` as `count` and `GL_UNSIGNED_BYTE` as `type`.

`glDrawElementsus(mode, indices)`
Special Python version of `glDrawElements` that uses length of the sequence `indices` as `count` and `GL_UNSIGNED_SHORT` as `type`.

`glFogv pname, params`  
Parameter `params` must be a Python sequence containing float values.

`glFogxv pname, params`  
Parameter `params` must be a Python sequence containing integer values.

`glGenTextures(n)`  
The generated texture names are returned in a Python tuple.

`glGetIntegerv pname`  
The values are returned in a Python tuple.

`glGetString name`  
The value is return as a Python string.

`glLightModelfv pname, params`  
Parameter `params` must be a Python sequence containing float values.
Parameter `params` must be a Python sequence containing integer values.

Parameter `params` must be a Python sequence containing float values.

Parameter `params` must be a Python sequence containing integer values.

Parameter `m` must be a Python sequence containing float values. The sequence is flattened before its items are read.

Parameter `m` must be a Python sequence containing integer values. The sequence is flattened before its items are read.

Parameter `params` must be a Python sequence containing float values.

Parameter `params` must be a Python sequence containing integer values.

Parameter `m` must be a Python sequence containing float values. The sequence is flattened before its items are read.

Parameter `m` must be a Python sequence containing integer values. The sequence is flattened before its items are read.

Parameter `sequence` must be either a `gles.array` object or some other Python sequence object. `gles.array` objects require less processing and can be therefore slightly faster. If `gles.array` object is used, the type of its data is ignored and `type` is used instead.

Special Python version of `glNormalPointer` that uses `type GL_BYTE` and `stride 0`.

Special Python version of `glNormalPointer` that uses `type GL_SHORT` and `stride 0`.

Special Python version of `glNormalPointer` that uses `type GL_FLOAT` and `stride 0`.

Special Python version of `glNormalPointer` that uses `type GL_FIXED` and `stride 0`.

The pixel data read is returned in a Python string.

Parameter `sequence` must be either a `gles.array` object or some other Python sequence object. `gles.array` objects require less processing and can be therefore slightly faster. If `gles.array` object is used, the dimension and type of its data are ignored and `size` and `type` are used instead.

Special Python version of `glTexCoordPointer` that accepts either a `gles.array` object or some other Python sequence object. Other parameters of `glTexCoordPointer` will be determined as follows:

- `size` If `sequence` is an instance of `gles.array`, its dimension is used; otherwise the length of `sequence` is used.
- `type` `GL_BYTE`
- `stride 0`

Special Python version of `glTexCoordPointers` that accepts either a `gles.array` object or some other Python sequence object. Other parameters of `glTexCoordPointers` will be determined as follows:

- `size` If `sequence` is an instance of `gles.array`, its dimension is used; otherwise the length of `sequence` is used.
- `type` `GL_BYTE`
- `stride 0`
Special Python version of `glTexCoordPointer` that behaves exactly as `glTexCoordPointerb` except `GL_SHORT` is used as `type`.

`glTexCoordPointerf(sequence)`  
Special Python version of `glTexCoordPointer` that behaves exactly as `glTexCoordPointerb` except `GL_FLOAT` is used as `type`.

`glTexCoordPointerv(sequence)`  
Special Python version of `glTexCoordPointer` that behaves exactly as `glTexCoordPointerb` except `GL_FIXED` is used as `type`.

`glTexCoordPointerf(sequence)`  
Special Python version of `glTexCoordPointer` that behaves exactly as `glTexCoordPointerb` except `GL_FLOAT` is used as `type`.

`glTexCoordPointerv(sequence)`  
Special Python version of `glTexCoordPointer` that behaves exactly as `glTexCoordPointerb` except `GL_FIXED` is used as `type`.

`glTexCoordPointerb(sequence)`  
Special Python version of `glTexCoordPointer` that behaves exactly as `glTexCoordPointerb` except `GL_SHORT` is used as `type`.

`glTexCoordPointerv(sequence)`  
Special Python version of `glTexCoordPointer` that behaves exactly as `glTexCoordPointerb` except `GL_FIXED` is used as `type`.

Parameter `params` must be a Python sequence containing float values.

Parameter `params` must be a Python sequence containing integer values.

`glTexImage2D(target, level, internalformat, width, height, border, format, type, pixels)`  
Parameter `pixels` must be either a Python string, a `gles.array` object, or `graphics.Image` object. Python strings are taken as literal data with no conversion. The dimension and type of data in `gles.array` objects are ignored: the raw data in the array is used.

Use of `graphics.Image` objects is limited to only some combinations of `format` and `type`. Table 5.6.3 below shows the accepted combinations. To get the best results and performance, the `CFbsBitmap` object in the `graphics.Image` object should be in the equivalent display mode, also shown in the table below. Otherwise, the `CFbsBitmap` object will be first converted to the equivalent display mode before reading its pixel data, which can degrade the visual quality in some cases.

<table>
<thead>
<tr>
<th><code>format</code></th>
<th><code>type</code></th>
<th>The equivalent display mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>GL_LUMINANCE, GL_ALPHA</td>
<td>GL_UNSIGNED_BYTE</td>
<td>EGray256</td>
</tr>
<tr>
<td>GL_RGB</td>
<td>GL_UNSIGNED_BYTE</td>
<td>EColor16M</td>
</tr>
<tr>
<td>GL_RGB</td>
<td>GL_UNSIGNED_SHORT_5_6_5</td>
<td>EColor64K</td>
</tr>
</tbody>
</table>

`glTexImage2D(target, level, internalformat, width, height, border, format, type, pixels)`  
The handling of `pixels` is the same as with `glTexImage2D`.

`glVertexPointer(size, type, stride, sequence)`  
Parameter `sequence` must be either a `gles.array` object or some other Python sequence object. `gles.array` objects require less processing and can be therefore slightly faster. If `gles.array` object is used, the dimension and type of its data are ignored and `size` and `type` are used instead.

`glVertexPointerv(sequence)`  
Special Python version of `glVertexPointer` that accepts either a `gles.array` object or some other Python sequence object. Other parameters of `glVertexPointerv` will be determined as follows:

- `size` If `sequence` is an instance of `gles.array`, its dimension is used; otherwise the length of `sequence` is used.
- `type` `GL_BYTE`
- `stride` 0

`glVertexPointers(sequence)`  
Special Python version of `glVertexPointer` that behaves exactly as `glVertexPointerv` except `GL_SHORT` is used as `type`.

`glVertexPointerf(sequence)`  
Special Python version of `glVertexPointer` that behaves exactly as `glVertexPointerv` except `GL_FLOAT` is used as `type`.

`glVertexPointerv(sequence)`  
Special Python version of `glVertexPointer` that behaves exactly as `glVertexPointerv` except `GL_FIXED` is used as `type`.

5.6. `gles` — Bindings to OpenGL ES
GL_FIXED is used as type.

OpenGL ES 1.1

glBufferData(target, size, data, usage)
Parameter data must be a gles.array object. If size is -1, the in-memory size of data is used in its place.

glBufferData(target, data, usage)
Special Python version of glBufferData that accepts either a gles.array object or some other Python sequence object for data. If gles.array object is used, its in-memory size in bytes is used as size. Other sequences are first converted to flat lists of GL_BYTE data by casting. The length of the resulting sequence in bytes is used as size.

glBufferDataub(target, data, usage)
Special Python version of glBufferData that works exactly like glBufferData except GL_UNSIGNED_BYTE is used instead of GL_BYTE.

glBufferDatas(target, data, usage)
Special Python version of glBufferData that works exactly like glBufferData except GL_SHORT is used instead of GL_BYTE.

glBufferDataf(target, data, usage)
Special Python version of glBufferData that works exactly like glBufferData except GL_FLOAT is used instead of GL_BYTE.

glBufferSubData(target, size, data, usage)
Parameter data must be a gles.array object. If size is -1, the in-memory size of data is used in its place.

glBufferSubDatab(target, data, usage)
Special Python version of glBufferSubData that accepts either a gles.array object or some other Python sequence object for data. If gles.array object is used, its in-memory size (in bytes) is used as size. Other sequences are first converted to flat lists of GL_BYTE data by casting. The length of the resulting sequence is used as size.

glBufferSubDataub(target, data, usage)
Special Python version of glBufferSubData that works exactly like glBufferSubDatab except GL_UNSIGNED_BYTE is used instead of GL_BYTE.

glBufferSubDatas(target, data, usage)
Special Python version of glBufferSubData that works exactly like glBufferSubDatab except GL_SHORT is used instead of GL_BYTE.

glBufferSubDataf(target, data, usage)
Special Python version of glBufferSubData that works exactly like glBufferSubDatab except GL_FLOAT is used instead of GL_BYTE.

glBufferSubDatax(target, data, usage)
Special Python version of glBufferSubData that works exactly like glBufferSubDatab except GL_FIXED is used instead of GL_BYTE.

glClipPlanef(plane, equation)
Parameter equation must be a Python sequence that contains four float values.

```python
glClipPlanex(plane, equation)
```
Parameter equation must be a Python sequence that contains four integer values.

```python
glDeleteBuffers(buffers)
```
Parameter buffers must be a Python sequence that contains integer values.

```python
glDrawTexsvOES(coords)
```
Parameter coords must be a Python sequence that contains integer values.

```python
glDrawTexivOES(coords)
```
Parameter coords must be a Python sequence that contains integer values.

```python
glDrawTexfvOES(coords)
```
Parameter coords must be a Python sequence that contains integer values.

```python
gluDeleteBuffers(n)
```
The generated buffer names are returned in a Python tuple.

```python
glGetBooleanv(pname)
```
The values are returned in a Python tuple.

```python
glGetBufferParameteriv(target, pname)
```
The value is returned as an integer.

```python
glGetClipPlanef(plane)
```
The values are returned in a Python tuple.

```python
glGetClipPlanef(plane)
```
The values are returned in a Python tuple.

```python
glGetFixedv(pname)
```
The values are returned in a Python tuple.

```python
glGetFloatv(pname)
```
The values are returned in a Python tuple.

```python
glGetLightfv(light, pname)
```
The values are returned in a Python tuple.

```python
glGetLightxv(light, pname)
```
The values are returned in a Python tuple.

```python
glGetMaterialfv(face, pname)
```
The values are returned in a Python tuple.

```python
glGetMaterialxv(face, pname)
```
The values are returned in a Python tuple.

```python
glGetTexParameterf(target, pname)
```
The value is returned as a float.

```python
glGetTexParameterx(target, pname)
```
The value is returned as an integer.

```python
glMatrixIndexPointerOES(size, type, stride, sequence)
```
Parameter sequence must be either a `gles.array` object or some other Python sequence object. `gles.array` objects require less processing and can be therefore slightly faster. If `gles.array` object is used, the dimension and type of its data are ignored and size and type are used instead.

5.6. `gles` — Bindings to OpenGL ES
glMatrixIndexPointerOESub(sequence)
Special Python version of glMatrixIndexPointerOES that accepts either a gles.array object or some other Python sequence object. Other parameters of glMatrixIndexPointerOES will be determined as follows:
• size If sequence is an instance of gles.array, its dimension is used; otherwise the length of sequence is used.
• type GL_UNSIGNED_BYTE
• stride 0

glPointParameterfv(pname, params)
Parameter params must be a Python sequence containing float values.

glPointParameterxv(pname, params)
Parameter params must be a Python sequence containing integer values.

glPointSizePointerOES(type, stride, sequence)
Parameter sequence must be either a gles.array object or some other Python sequence object. gles.array objects require less processing and can be therefore slightly faster. If gles.array object is used, the type of its data is ignored and type is used instead.

glPointSizePointerOESf(sequence)
Special Python version of glPointSizePointerOES uses GL_FLOAT as type and 0 as stride.

glPointSizePointerOESx(target, data, usage)
Special Python version of glPointSizePointerOES uses GL_FIXED as type and 0 as stride.

glWeightPointerOES(size, type, stride, sequence)
Parameter sequence must be either a gles.array object or some other Python sequence object. gles.array objects require less processing and can be therefore slightly faster. If gles.array object is used, the dimension and type of its data are ignored and size and type are used instead.

glWeightPointerOESf(sequence)
Special Python version of glWeightPointerOES that accepts either a gles.array object or some other Python sequence object. Other parameters of glWeightPointerOES will be determined as follows:
• size If sequence is an instance of gles.array, its dimension is used; otherwise the length of sequence is used.
• type GL_FLOAT
• stride 0

glWeightPointerOESx(sequence)
Special Python version of glWeightPointerOES that behaves exactly as glWeightPointerOESf except GL_FIXED is used as type.

5.7 glcanvas — UI Control for Displaying OpenGL ES Graphics

The glcanvas module provides a UI control, GLCanvas, for displaying OpenGL ES graphics. GLCanvas component is similar to the appuiw Canvas component that supports Symbian OS-level drawing.

Internally GLCanvas uses EGL for displaying the OpenGL ES graphics. EGL, as OpenGL ES, is a standard API defined by the Khronos Group (www.khronos.org). Specifically, GLCanvas uses an EGL window surface, which supports double-buffered rendering. It is possible to affect selection of the EGL config that is used to create the window surface; for details, see the documentation of the GLCanvas constructor.

GLCanvas instances also hold the OpenGL ES context object, which together with the surface, are needed for rendering. When one wants to render with a specific OpenGL ES context to a specific surface, they need to be made current. This also applies to GLCanvas, which has the makeCurrent method for this purpose. Generally, calling makeCurrent has to be done only if multiple GLCanvas objects are used in
the same program, as each GLCanvas object is automatically made current when it is created and it remains current until it is destroyed or makeCurrent of some other GLCanvas object is called.

```python
class GLCanvas(redraw_callback, [event_callback=None, resize_callback=None, attributes=None])
```

Constructs a new GLCanvas object that can be used as a UI control for displaying OpenGL ES graphics. Parameters `redraw_callback`, `event_callback`, and `resize_callback` have the same meaning as with `appuifw` module `Canvas`. Using `redraw_callback` to specify the OpenGL ES drawing is preferred as it will be automatically called by `drawNow` method.

Parameter `attributes` can be used to specify attributes used in EGL config selection. It must be a Python dictionary where keys are EGL attribute names (which are defined in the `glcanvas` module) and values are integers defining the desired attribute values. Unless specified in `attributes`, `EGL_BUFFER_SIZE` is set to value based on the display mode of the window owned by the underlying `CCoeControl` object and `EGL_DEPTH_SIZE` is set to 16. Attributes specified in `attributes` are given to `eglChooseConfig`. Refer to the EGL specification for a detailed list of config attributes and explanation of how the selection of EGL configs works.

The new GLCanvas object will be made current when the constructor returns so `makeCurrent` does not have to be called before starting to use OpenGL ES calls.

```python
bind(key_code, c)
```

Sets a callback to be called when a specific key is pressed. Parameter `key_code` should be one of the standard Symbian key codes defined in `key_codes`. Parameter `c` must be a callable object.

```python
drawNow()
```

Calls the redraw callback (if set) and then calls `eglSwapBuffers` to render and display the OpenGL ES graphics.

```python
makeCurrent()
```

Makes this GLCanvas object current, meaning that it will be used to display the results of the subsequent OpenGL ES calls. In EGL terms this means that the EGL context and surface held by this object will be passed to `eglMakeCurrent`. Using `makeCurrent` makes it possible to use several GLCanvas objects in a single application: the receiver of the OpenGL ES calls can be switched with `makeCurrent` easily.

5.7. glcanvas — UI Control for Displaying OpenGL ES Graphics
CHAPTER

SIX

Audio and Communication Services

6.1 audio — An audio related services package

The audio module enables recording and playing audio files and access to device text-to-speech engine. The audio module supports all the formats supported by the device, typically: WAV, AMR, MIDI, MP3, AAC, and Real Audio\(^1\). For more information on the audio types supported by different devices, see the Forum Nokia Web site [7] and S60 Platform Web site [8].

The following Sound class static methods are defined in the audio module:

```python
Sound.open(filename)
```

Returns a new initialized Sound object with the named file opened. Note that filename should be a full Unicode path name and must also include the file extension, for example u’c:\foo.wav’.

The following data items for state information are available in audio:

- ENotReady
  The Sound object has been constructed but no audio file is open.

- EOpen
  An audio file is open but no playing or recording operation is in progress.

- EPlaying
  An audio file is playing.

- ERecording
  An audio file is being recorded.

The following data item is provided for continuous playback of an audio file:

- KMdaRepeatForever
  Possible value for times parameter in open.

The following method is available in the audio module:

```python
say(text, prefix=audio.TTS_PREFIX)
```

Passes the text to the device text-to-speech engine. The default prefix is the text-to-speech prefix "(tts)".

6.1.1 Sound Objects

**Note:** The method current_volume is not available for S60 1st Edition.

```python
class Sound
    Sound objects have the following functions:
    play([times=1, interval=0, callback=None])
    Starts playback of an audio file from the beginning. Without the parameters times and
```

\(^1\)The dynamically loaded audio codec for the sound file is based on the MIME-type information inside the audio file and file extension.
interval it plays the audio file one time. times defines the number of times the audio file is played, the default being 1. If the audio file is played several times, interval gives the time interval between the subsequent plays in microseconds.

The optional callback is called when the playing starts and when the end of the sound file is reached. The callback should take three parameters: the previous state, the current state and the possible error code. The possible states given as parameters to the callback are data items in the module audio.

Other issues:

- Calling **play(audio.KMdaRepeatForever)** will repeat the file forever.
- If an audio file is played but not stopped before exiting, the Python script will leave audio playing on; therefore **stop** needs to be called explicitly prior to exit.
- Currently the module does not support playing simultaneous audio files, calling **play** to a second Sound instance while another audio file is playing, stops the earlier audio file and starts to play the second Sound instance.
- Calling **play** while a telephone call is ongoing plays the sound file to uplink. In some devices the sound file is also played to the device speaker.
- Calling **play** when already playing or recording results in **RuntimeError**. Calling **stop** prior to **play** will prevent this from happening.

**stop()**

Stops playback or recording of an audio file.

**record()**

Starts recording audio data to a file. If the file already exists, the operation appends to the file. For Nokia devices, WAV is typically supported for recording. For more information on the audio types supported by different devices, see the Forum Nokia Web site [7] and S60 Platform Web site [8]. Other issues:

- Calling **record** while a telephone call is ongoing starts the recording of the telephone call.
- Calling **record** when already playing or recording results in **RuntimeError**. Calling **stop** prior to **record** will prevent this from happening.

**close()**

Closes an opened audio file.

**state()**

Returns the current state of the Sound type instance. The different states (constants) are defined in the audio module. The possible states\(^2\) are:

- **ENotReady**
  The Sound object has been constructed but no audio file is open.
- **EOpen**
  An audio file is open but no playing or recording operation is in progress.
- **EPlaying**
  An audio file is playing.
- **ERecording**
  An audio file is being recorded.

**max_volume()**

Returns the maximum volume of the device.

**set_volume(volume)**

Sets the volume. If the given volume is negative, then the volume is set to zero which mutes the device. If the volume is greater than **max_volume**, then **max_volume** is used.

**current_volume()**

Returns the current volume set.

**duration()**

Returns the duration of the file in microseconds.

**set_position(microseconds)**

Set the position for the playhead.

\(^2\)Descriptions for these options are based on information found in S60 SDK documentation [4].
current_position()
Returns the current playhead position in microseconds.

6.2 telephone — Telephone services

This module provides an API to a telephone.

Since the users of the device can also hang-up the phone explicitly, they might affect the current status of the call. In addition, using this extension in an emulator has no effect since no calls can be connected.

The telephone module has the following functions:

dial(number)
Dials the number set in number. number is a string, for example u '+358501234567' where '+' is the international prefix, '358' is the country code, '50' is the mobile network code (or the area code), and '1234567' is the subscriber number. If there is an ongoing phone call prior to calling dial from Python, then the earlier call is put on hold and a new call is established. Calling dial multiple times when, for example, the first call has been answered and a line has been established results in subsequent calls not being connected.

hang_up()
Hangs up if a call initiated by dial is in process. If this call has already been finished, SymbianError: KErrNotReady is raised.

6.3 messaging — A messaging services package

The messaging module offers APIs to messaging services. Currently, the messaging module has functions:

sms_send(recipient, message, [encoding='7bit', callback=None])
Sends an SMS message with body text message (Unicode) to telephone number recipient (string). The optional parameter encoding is used to define encoding in the message. The parameter values can be '7bit', '8bit' or 'UCS2'.

The optional parameter callback is invoked with the current status of the send operation as parameter. The possible states are data items in the module messaging. Invoking another send while a previous send request is ongoing will result in RuntimeError being raised.

If the callback is not given, the sms_send function will block until the message in the queue is either deleted or the sending has failed.

mms_send(recipient, message, [attachment=None])
Note: Available from S60 3.0 onwards (inclusive).
Sends an MMS message with body text message (Unicode) to telephone number recipient (string). The optional parameter attachment is full path to e.g. image file attached to the message.

The following data items for SMS sending state information are available in the module messaging:
ECreated
EMovedToOutBox
EScheduledForSend
ESent
The SMS message has been sent.
EDeleted
The SMS message has been deleted from device’s outbox queue. The sms_send operation has finalized and subsequent SMS sending is possible.

3Please note that this blocking might last for several minutes and hence supplying the callback might be more suitable in many cases.

6.2. telephone — Telephone services 49
EScheduleFailed

ESendFailed
This state information is returned when the SMS subsystem has tried to send the message several
times in vain. The \texttt{sms\_send} operation has finalized and subsequent SMS sending is possible.

ENoServiceCentre
This state information is returned by the SMS subsystem in S60 3.x emulator. In emulator this
indicates that the \texttt{sms\_send} operation has finalized and subsequent SMS sending is possible.

EFatalServerError
The underlying messaging subsystem in S60 devices might give error messages to the user if the device
is not connected to a network while trying to send a message – An "SMS send failed!" note is a common
error message.

When sending messages in offline-mode or with no network connection these messages are actually added
to an outgoing message queue and they might be sent if the device is later on connected to a suitable
network\textsuperscript{4}. This occurs despite the possibly misleading error messages. The current network conditions
can be checked e.g. with \texttt{sysinfo.active\_profile()} and \texttt{sysinfo.signal\_bars()} invocations.

The following is example code for state information processing with \texttt{sms\_send} operation:


global import messaging

def cb(state):
    if state==messaging.ESent:
        print "**Message was sent**"
    if state==messaging.ESendFailed:
        print "**Something went wrong - Truly sorry for this**"

>>> messaging.sms_send("1234567", "Hello from PyS60!", '7bit', cb)
>>> **Message was sent** # This is printed from the callback

6.4 \texttt{inbox} — Interface to device inbox

The \texttt{inbox} module offers APIs to device inbox, outbox, sent and drafts folders. Currently, the \texttt{inbox}
module supports only SMS handling and notifications of incoming messages to the device inbox.

\texttt{class Inbox([folder\_type])}
Create an \texttt{Inbox} object.

The optional parameter \texttt{folder\_type} defines the type of the folder to which the created \texttt{Inbox}
object has access to. The default is the device’s inbox folder, \texttt{inbox.EInbox}.

The following data items are available in the \texttt{inbox} module to define the type of the folder for \texttt{Inbox}
objects:

\texttt{EInbox}
The device’s inbox folder.

\texttt{EOutbox}
The device’s outbox folder.

\texttt{ESent}
The sent messages folder.

\texttt{EDraft}
The draft messages folder.

\textsuperscript{4}Note also that prior this the user of the device can explicitly delete the messages from the native messaging application.
The amount of resending is approx. 4 times – After this the sending operation is cancelled and the user of the device will
see a visual cue of the failure in the status pane.
6.4.1 Inbox Objects

Inbox objects have the following functions:

**sms_messages()**
Returns a list of SMS message IDs in device inbox.

**content(sms_id)**
Retrieve the SMS message content in Unicode.

**time(sms_id)**
Retrieve the SMS message time of arrival in seconds since epoch.

**address(sms_id)**
Retrieve the SMS message sender address in Unicode.

**delete(sms_id)**
Delete the SMS message from inbox.

**unread()**

sms_id Returns the status (1=unread, 0=read) of the SMS with id.

**bind(callable)**

Bind a callback to receive new message events in device inbox. When a new message arrives to the
device inbox the callback gets called with the received message ID. The received message can be
other than an SMS message.

If the message received is deleted immediately after e.g. checking the message content, the "new
message" sound and dialog are not activated. This functionality might be useful in notification
type of applications.

Examples:

```python
>>> import inbox
>>> i=inbox.Inbox() # Give inbox.ESent as parameter for sent SMSes
>>> m=i.sms_messages()
>>> i.content(m[0])
'u'foobar'
>>> i.time(m[0])
1130267365.03125
>>> i.address(m[0])
u'John Doe'
>>> i.delete(m[0])

>>> import inbox
>>> id=0
>>> def cb(id cb):
... global id
... id=id cb
...
>>> i=inbox.Inbox()
>>> i.bind(cb)
>>> # Send an SMS to your inbox here. The "id" gets updated
>>> i.address(id)
u'John Doe'
>>> i.content(id)
u'print 1'
```

6.5 location — GSM location information
The location module offers APIs to location information related services. Currently, the location has one function:

**Note:** Location module requires capabilities ReadDeviceData, ReadUserData and Location in 3rd Edition devices.

gsm_location()

Retrieves GSM location information: Mobile Country Code, Mobile Network Code, Location Area Code, and Cell ID. A location area normally consists of several base stations. It is the area where the terminal can move without notifying the network about its exact position. mcc and mnc together form a unique identification number of the network into which the phone is logged.

### 6.5.1 Examples

Here is an example of how to use the location package to fetch the location information:

```python
>>> import location
>>> print location.gsm_location()
```

### 6.6 position — Simplified interface to the Location Acquisition API

The position provides basic access to the S60 Location Acquisition API. The Location Acquisition API gathers different positioning technologies together to be used through a consistent interface. Location Acquisition API offers quite a large amount of information (cost of service, device power consumption etc.) about accessible positioning devices (like GPS-modules), position, course, accuracy and satellite information (depending on the position device used) and much more. The Location Acquisition API can also be used to obtain device/vendor specific extended information.

The position module has the following functions:

- `modules()`
  - get information about available positioning modules
- `default_module()`
  - get default module id
- `module_info(module_id)`
  - get detailed information about the specified module
- `select_module(module_id)`
  - select a module
- `set_requestors(requestors)`
  - set the requestors of the service (at least one must be set)
- `position(course=0,satellites=0)`
  - get the position information

### 6.6.1 Example

The following example demonstrates how to use the python position module.

```python
# information about available positioning modules
print "***available modules***"
print positioning.modules()
print ""

# id of the default positioning module
print "***default module***"
```
print positioning.default_module()
print ""

# detailed information about the default positioning module
print "***detailed module info***"
print positioning.module_info(positioning.default_module())
print ""

# select a module (however, selecting default module has no
# relevance..this has been added just to show the functionality).
positioning.select_module(positioning.default_module())

# set requestors.
# at least one requestor must be set before requesting the position.
# the last requestor must always be service requestor
# (whether or not there are other requestors).
positioning.set_requestors([{
    "type" : "service",
    "format" : "application",
    "data" : "test_app"}])

# get the position.
# note that the first position()-call may take a long time
# (because of gps technology).
print "***position info***"
print positioning.position()
print ""

# re-get the position.
# this call should be much quicker.
# ask also course and satellite information.
print "***course and satellites***"
print positioning.position(course=1, satellites=1)
print ""

To run the script in the emulator you must configure PSY emulation (SimPSYConfigurator->Select Config File -><some config file s>).

6.6. position — Simplified interface to the Location Acquisition API 53
7.1 contacts — A contacts related services package

The contacts module offers an API to address book services allowing the creation of contact information databases. The contacts module represents a Symbian contact database as a dictionary-like ContactDb object, which contains Contact objects and which is indexed using the unique IDs of those objects. A Contact object is itself a list-like object, which contains ContactField objects and which is indexed using the field indices. Unique IDs and field indices are integers. The ContactDb object supports a limited subset of dictionary functionality. Therefore, only __iter__, __getitem__, __delitem__, __len__, keys, values, and items are included.

ContactDb objects represent a live view into the database. If a contact is changed outside your Python application, the changes are visible immediately, and conversely any changes you commit into the database are visible immediately to other applications. It is possible to lock a contact for editing, which will prevent other applications from modifying the contact for as long as the lock is held. This can be done in, for example, a contacts editor application when a contact is opened for editing, very much like with the Contacts application in your Nokia device. If you try to modify a contact without locking it for editing, the contact is automatically locked before the modification and released immediately afterwards.

7.1.1 Module Level Functions

The following free functions - functions that do not belong to any class - are defined in the Contact module:

```
open([filename[, mode]])
```

Opens a contacts database and returns a ContactDb object. filename should be a full Unicode path name. If filename is not given, opens the default contacts database. If mode is not given, the database must exist. If mode is 'c', the database is created if it does not already exist. If mode is 'n', a new, empty database is created, overwriting the possible previous database.

**Warning:** Using open together with the additional parameters filename or mode is intended for testing purposes only. Due to S60 SDK functionality, the open method can sometimes be unreliable with these parameters.

7.1.2 ContactDb Object

There is one default contact database, but it is possible to create several databases with the open function.

```
class ContactDb
    ContactDb objects have the following methods:

    add_contact()
    Adds a new contact into the database. Returns a Contact object that represents the new contact. The returned object is already locked for modification. Note that a newly created contact will contain some empty default fields. If you do not want to use the default fields for anything, you can ignore them.
```
find(searchterm)
Finds the contacts that contain the given Unicode string as a substring and returns them as a list.

import_vcards(vcards)
Imports the vCard(s) in the given string into the database.

export_vcards(ids)
Converts the contacts corresponding to the ID’s in the given tuple ids to vCards and returns them as a string.

keys()
Returns a list of unique IDs of all Contact objects in the database.

compact_required()
Verifies whether compacting is recommended. Returns an integer value indicating either a true or false state. Returns True if more than 32K of space is unused and if this comprises more than 50 percent of the database file, or if more than 256K is wasted in the database file.

compact()
Compacts the database to its minimum size.

__delitem__(id)
Deletes the given contact from the database.

field_types()
Returns a list of dictionary objects that contains information on all supported field types. The list contains dictionary objects, which each describe one field type. The most important keys in the dictionary are 'type' and 'location' which together indentify the field type. 'type' can have string values such as 'email_address'. 'location' can have the string values 'none', 'home', or 'work'. Another important key is 'storage_type', which defines the storage type of the field. 'storage_type' can have the string values 'text', 'datetime', 'item_id', or 'binary'. Note that the Contacts extension does not support adding, reading, or modifying fields of any other type than 'text' or 'datetime'. The other content returned by field_types is considered to be advanced knowledge and is not documented here.

groups
Returns contact groups of the database. Read-only.

7.1.3 Contact Object

A Contact object represents a live view into the state of a single contact in the database. You can access the fields either with a contact’s numeric field ID as contact[fieldid], or using the find method. Attempting to modify a contact while it has been locked for editing in another application will raise the exception ContactBusy.

class Contact
Contact objects have the following attributes:

id
The unique ID of this Contact. Read-only.

title
The title of this Contact. Read-only.

is_group
Returns 1 if this contact is a contact group. Returns 0 if normal contact entry. Read-only.

Contact objects have the following methods:

begin()
Locks the contact for editing. This prevents other applications from modifying the contact for as long as the lock is held. This method will raise the exception ContactBusy if the contact has already been locked.

commit()
Releases the lock and commits the changes made into the database.
rollback()
Releases the lock and discards all changes that were made. The contact remains in the state it was before begin.

as_vcard()
Returns the contact as a string in vCard format.

add_field(type [, value [, label=field_label] [, location=location_spec]]
Adds a new field into this Contact. This method raises ContactBusy if the contact has been locked by some other application. type can be one of the supported field types as a string. In Series 60 editions older than the 3rd one the following field types can be added:

- city
- company_name
- country
- date
- dtmf_string
- email_address
- extended_address
- fax_number
- first_name
- job_title
- last_name
- mobile_number
- note
- pager_number
- phone_number
- po_box
- postal_address
- postal_code
- state
- street_address
- url
- video_number
- wvid

The following field types are recognized but cannot be created:

- first_name_reading
- last_name_reading
- picture
- speed_dial
- thumbnail_image
- voicetag

If 3rd edition of Series 60 is used the following field types can be added:

- city
- company_name
- country
- date
- dtmf_string
- email_address
- extended_address
- fax_number
- first_name
- job_title
• last_name
• mobile_number
• note
• pager_number
• phone_number
• po_box
• postal_address
• postal_code
• state
• street_address
• url
• video_number
• picture
• second_name
• voip
• sip_id
• personal_ringtone
• share_view
• prefix
• suffix
• push_to_talk
• locationid

The following field types are recognized but cannot be created at present:
• first_name_reading
• last_name_reading
• speed_dial
• thumbnail_image
• voice_tag
• wvid

All supported field types are passed as strings or Unicode strings, except for ‘date’ which is a float that represents Unix time. For more information on Unix time, see Section 3.5, Date and Time.

field_label is the name of the field shown to the user. If you do not pass a label, the default label for the field type is used.

location_spec, if given, must be ‘home’ or ‘work’. Note that not all combinations of type and location are valid. The settings of the current contacts database in use determine which ones are valid.

find([[type=field_type], location=field_location])
Finds the fields of this contact that match the given search specifications. If no parameters are given, all fields are returned.

__delitem__(fieldindex)
Deletes the given field from this contact. Note that since this will change the indices of all fields that appear after this field in the contact, and since the ContactField objects refer to the fields by index, old ContactField objects that refer to fields after the deleted field will refer to different fields after this operation.

7.1.4 ContactField Object

A ContactField represents a field of a Contact at a certain index. A ContactField has attributes, some of which can be modified. If the parent Contact has not been locked for editing, modifications are committed immediately to the database. If the parent Contact has been locked, the changes are committed only when commit is called on the Contact.
class ContactField

ContactField objects have the following attributes:

- label
  The user-visible label of this field. Read-write.
- value
  The value of this field. Read-write.
- type
  The type of this field. Read-only.
- location
  The location of this field. This can be 'none', 'work', or 'home'.
- schema
  A dictionary that contains some properties of this field. The contents of this dictionary correspond to those returned by the ContactDb method field_types.

7.1.5 Groups Object

A Groups object represents Symbian contact groups as a dictionary like object with limited subset of dictionary functionality. Each group can be accessed using the group’s unique id as a key. The Groups object returns a list like Group object as the value matching the given key.

The following common methods are supported: __iter__, __getitem__, __delitem__ and __len__.

class Groups

Groups objects have the following attributes:

- add_group(name)
  Creates new contact group and returns corresponding Group object. Group name can be given as an optional parameter.

7.1.6 Group Object

A Group object represents single Symbian contact group as a list object with limited subset of list functionality. The Group object lists Contact entry ids that belong to the group.

The native Symbian group objects are represented as Symbian contact entries in the database. Therefore they can also be accessed as Python Contact objects, but this way their group handling properties cannot be used from Python. Use Groups and Group objects to access group functionalities.

The following common methods are supported: __iter__, __getitem__, __delitem__ and __len__.

class Group

Group objects have the following attributes:

- id
  The unique id of the Group object. Read-only.
- name
  The name of the Group object. Read-write.
7.2 calendar — Access to calendar related services

The calendar module offers an API to calendar services. The calendar module represents a Symbian agenda database as a dictionary-like `CalendarDb` object, which contains `Entry` objects and which is indexed using the unique IDs of those objects. There are four types of entry objects: `AppointmentEntry`, `EventEntry`, `AnniversaryEntry`, and `TodoEntry`.

`CalendarDb` objects represent a live view into the database. If an entry is changed outside your Python application, the changes are visible immediately, and conversely any changes you commit into the database are visible immediately to other applications.

In addition to entries, there are todo lists which contain todo entries. Todo lists are accessed using the dictionary-like `TodoListDict` and `TodoList` objects.

All time parameters use Unix time unless stated otherwise. For more information on Unix time, see Section 3.5, Date and Time.

Figure 7.1 demonstrates the relationships of the calendar module objects.

### 7.2.1 Module Level Functions

The following free functions - functions that do not belong to any class - are defined in the calendar module:

```python
open(filename=None, mode=None)
```

Opens a calendar database and returns a new `CalendarDb` object.

If `filename` is `None`, the default database is opened.

If `filename` is given, it should be a full, absolute path name in Unicode that specifies the calendar database to open.

`mode` can be:

- `None`: Opens an existing calendar database.
- `'c'`: Opens an existing calendar database, or creates it if it doesn’t exist.
- `'n'`: Creates a new, empty calendar database. If `filename` exists, the previous contents are erased.
7.2.2 CalendarDb Objects

Calendar entries and todo lists are stored in a calendar database. There is one default calendar database but more calendar databases can be created by invoking open with parameters 'n' or 'c'.

**class CalendarDb**

CalendarDb objects have the following methods:

- **add_appointment()**
  Creates and returns a new appointment entry AppointmentEntry. The entry is not added and saved into the database until Entry.commit is called.

- **add_event()**
  Creates and returns a new event entry EventEntry. The entry is not added and saved into the database until Entry.commit is called.

- **add_anniversary()**
  Creates and returns a new anniversary entry AnniversaryEntry. The entry is not added and saved into the database until Entry.commit is called.

- **add_todo()**
  Creates and returns new todo entry TodoEntry. The entry is not added and saved into the database until Entry.commit is called.

- **find_instances(start_date, end_date, search_str='[appointments=0,events=0,anniversaries=0,todos=0]')**
  The parameters for this function include the start date, end date, search string, and optional parameters. The optional parameters define the entry types to be included into the search. By default all entry types are included. Returns a list that contains Entry instances found in the search. An instance is a dictionary that contains the entry ID and the datetime value. An entry may have several instances if it is repeated, for example once every week, etc. However, all the returned instances occur on the same day, i.e. on the first day between the start and end datetime values that contains instances. To search all instances between the initial start and end datetime values, you may have to execute several searches and change the start datetime value for each search. A match is detected if the search string is a substring of an entry's content.

- **monthly_instances(month, appointments=0, events=0, anniversaries=0, todos=0)**
  The parameters for this function include month (float) and optional parameters. The optional parameters define the entry types to be returned. Returns a list that contains entry instances occurring during the specified calendar month.

- **daily_instances(day, appointments=0, events=0, anniversaries=0, todos=0)**
  The parameters for this function include day (float) and optional parameters. The optional parameters define the entry types to be returned. Returns a list that contains entry instances occurring on the specified day.

- **add_todo_list([name=None])**
  Creates a new todo list. name sets the name of the todo list (Unicode). Returns the ID of the created todo list.

- **export_vcalendars((int,...))**
  Returns a vcalendar string that contains the specified entries in vCalendar format. The parameter for this function is a tuple that contains the entry IDs of the exported entries.

- **import_vcalendars(string)**
  Imports vcalendar entries, given in the string parameter, to the database. Returns a tuple that contains the unique IDs of the imported entries.

- **todo_lists**
  Contains a dictionary-like TodoListDict object for accessing the todo lists of this database.

- **__delitem__(id)**
  Deletes the given calendar Entry from the database. id is the unique ID of the calendar Entry.

- **__getitem__(id)**
  Returns a calendar Entry object indicated by the unique ID. The returned object can be one
of the following: AppointmentEntry, EventEntry, AnniversaryEntry, or TodoEntry. id is the unique ID of the calendar Entry.

compact()
Compacts the database file. The returned value (integer) indicates the success of compaction; a value other than zero means that the compaction was successful.

7.2.3 Entry Objects

An Entry object represents a live view into the state of a single entry in the database. You can access the entries with an entry’s unique ID. If you create a new entry using db.add_appointment etc., it is saved into the database only if you call the entry’s commit method. In case an entry is already saved into the database, the autocommit mode is on by default and all the changes are automatically saved into the database, unless you call the entry’s begin method. If you call the entry’s begin method, the changes are not saved into the database until you call the entry’s commit method.

Database entries cannot be locked. In other words, other applications are able to make changes to the database entries you are using (not directly to the EntryObjects you are using, but to their representation in the database) at the same time you are modifying them, even if you use begin and commit methods.

class Entry
Entry objects have the following methods and properties:

content
Sets or returns the entry’s content text (Unicode).

commit()
Saves the entry or in case of a new entry adds the entry into the database. Note that this can be called only in case of a new entry, created with db.add_appointment etc., or after begin is called.

rollback()
Undoes the changes made after last commit.

set_repeat(dictionary)
Sets the repeat data of the entry. dictionary is a repeat data dictionary that contains all the repeat rules. For more information on repeat rules, see Section 7.3.4, Repeat Rules.

get_repeat()
Returns the repeat data dictionary of the entry.

location
Sets or returns the entry’s location data (Unicode), for example meeting room information.

set_time(start, end)
Sets the start and end datetime values of the entry (floats). If only one parameter is given, the other will have the same value.
In case of events, anniversaries, and todo entries the datetime values are truncated to corresponding date values.
TodoEntries can be made undated with TodoEntry.set_time(None). Making the todo entry undated means removing the start and end date and all the repeat rules.

start_time
The start datetime value (float) of the entry or None if the start datetime of the entry is not set.

date_time
The end datetime value (float) of the entry or None if the end datetime of the entry is not set.

id
The unique ID of the entry.

last_modified
The datetime value (float) of the entry’s last modification in universal time.
alarm
The alarm datetime value (float) for the entry. None if alarm is not set. Alternatively removes
the alarm if the value is set to None.
Alarms can be set to all Entry types. However, only alarms set to Appointments and Anniversaries
will actually cause an alarm; this is similar to the Calendar application in your Nokia
device, which allows you to set an alarm only for Meetings and Anniversaries. In addition,
alarm set to any entries residing in a database other than the default database do not cause
actual alarms either.

date
The priority of the entry, which can be an integer ranging from 0 to 255. Native Phonebook
and Calendar applications in Nokia devices use value 1 for high priority, 2 for normal priority,
and 3 for low priority.

crossed_out
The crossed out value of an entry. A value that is interpreted as false means that the entry is
not crossed out, whereas a value that is interpreted as true means that the entry is crossed out.
Note that TodoEntries must also have a cross-out time while the other entry types cannot
have one. If TodoEntry is crossed out using this method, the moment of crossing out is set
to the cross-out time of the TodoEntry. See also Section 7.3.3, TodoEntry, cross_out_time.

replication
Sets or returns the entry’s replication status, which can be one of the following: 'open',
'private', or 'restricted'.
as_vcalendar()
Returns this entry as a vCalendar string.

AppointmentEntry Objects
class AppointmentEntry
AppointmentEntry class contains no additional methods compared to the Entry class from which it is
derived.

EventEntry
class EventEntry
EventEntry class contains no additional methods compared to the Entry class from which it is derived.

AnniversaryEntry
class AnniversaryEntry
AnniversaryEntry class contains no additional methods compared to the Entry class from which it is
derived.

TodoEntry
todoEntryobjects represent todo entry types. They have additional properties compared to the Entry
class from which they are derived.
class TodoEntry
TodoEntry objects have the following additional properties:
cross_out_time
The cross-out date value of the entry. The value can be None meaning that the entry is
not crossed out, or the cross-out date (float). The set value must be date (float). Setting a
cross-out time also crosses out the entry. See also Section 7.3.3, Entry Object, crossed_out.
The ID of the todo list to which this entry belongs.

TodoListDict

TodoListDict objects are dictionary-like objects that enable accessing todo lists.

class TodoListDict
    TodoListDict objects have the following property:
    
    default_list
        The ID of the default todo list.

TodoList

TodoList objects are dictionary-like objects that enable accessing todo lists.

class TodoList
    TodoList objects have the following properties:

    name
        The name of the todo list as a Unicode string.

    id
        Returns the ID of the todo list as an integer.

7.2.4 Repeat Rules

Repeat rules specify an entry's repeat status, that is, the recurrence of the entry. There are six repeat types:

- **daily**: repeated daily
- **weekly**: repeat on the specified days of the week, such as Monday and Wednesday, etc.
- **monthly_by_dates**: repeat monthly on the specified dates, such as the 15th and 17th day of the month
- **monthly_by_days**: repeat monthly on the specified days, such as the fourth Wednesday of the month, or the last Monday of the month
- **yearly_by_date**: repeat yearly on the specified date, such as December 24
- **yearly_by_day**: repeat yearly on the specified day, such as every third Tuesday of May

There are exceptions to repeat rules. For example, you can specify the datetime value (float) in such a way that the entry is not repeated on a specific day even if the repeat rule would specify otherwise.

You must set the start and end dates (floats) of the repeat. The end date can also be set to `None` to indicate that the repeating continues forever. You can set `interval` defining how often the repeat occurs, for example in a daily repeat: 1 means every day, 2 means every second day, etc. You can also set the `days` specifier which lets you explicitly specify the repeat days; for example in a weekly repeat you can set "days": [0, 2] which sets the repeat to occur on Mondays and Wednesdays. If you do not set the `days` specifier, the repeat days are calculated automatically based on the start date.

You can modify repeat data by calling `rep_data = entry.get_repeat()`, then making changes to `rep_data` dictionary, and then calling `entry.set_repeat(rep_data)`.

Repeating can be cancelled by calling `entry.set_repeat` with a parameter that is interpreted to be false, such as `entry.set_repeat(None)`.

Repeat definition examples:
repeat = {"type":"daily", #repeat type
  "exceptions":[exception_day, exception_day+2*24*60*60],
  #no appointment on those days
  "start":appt_start_date, #start of the repeat
  "end":appt_start_date+30*24*60*60, #end of the repeat
  "interval":1} #interval (1=every day, 2=every second day etc.)

repeat = {"type":"weekly", #repeat type
  "days":[0,1], #which days in a week (Monday, Tuesday)
  "exceptions":[exception_day], #no appointment on that day
  "start":appt_start_date, #start of the repeat
  "end":appt_start_date+30*24*60*60, #end of the repeat
  "interval":1} #interval (1=every week, 2=every second week etc.)

repeat = {"type":"monthly_by_days", #repeat type
  # appointments on second Tuesday and last Monday of the month
  "days": [{"week":1, "day":1}, {"week":4, "day":0}],
  "exceptions":[exception_day], #no appointment on that day
  "start":appt_start_date, #start of the repeat
  "end":appt_start_date+30*24*60*60, #end of the repeat
  "interval":1} #interval (1=every month, 2=every second month etc.)

repeat = {"type":"monthly_by_dates", #repeat type
  "days": [0,15],
  # appointments on the 1st and 16th day of the month
  "exceptions":[exception_day], #no appointment on that day
  "start":appt_start_date, #start of the repeat
  "end":appt_start_date+30*24*60*60, #end of the repeat
  "interval":1} #interval (1=every month, 2=every second month etc.)

repeat = {"type":"yearly_by_date", #repeat type
  "exceptions":[exception_day], #no appointment on that day
  "start":appt_start_date, #start of the repeat
  "end":appt_start_date+3*365*24*60*60, #end of the repeat
  "interval":1} #interval (1=every year, 2=every second year etc.)

repeat = {"type":"yearly_by_day", #repeat type
  # appointments on the second Tuesday of February
  "days": {"day":1, "week":1, "month":1},
  "exceptions":[exception_day], #no appointment on that day
  "start":appt_start_date, #start of the repeat
  "end":appt_start_date+3*365*24*60*60, #end of the repeat
  "interval":1} #interval (1=every year, 2=every second year etc.)

7.3 calendar for EKA2 — Access to calendar related services

The calendar module offers an API to calendar services. The calendar module represents a Symbian agenda database as a dictionary-like CalendarDb object, which contains Entry objects and which is indexed using the unique IDs of those objects. There are five types of entry objects: AppointmentEntry, EventEntry, AnniversaryEntry, ReminderEntry, and TodoEntry.

CalendarDb objects represent a live view into the database. If an entry is changed outside your Python application, the changes are visible immediately, and conversely any changes you commit into the database are visible immediately to other applications.
All time parameters use Unix time unless stated otherwise. For more information on Unix time, see Section 3.5, Date and Time.

7.3.1 Module Level Functions

The following free functions - functions that do not belong to any class - are defined in the calendar module:

```python
open(filename=None, mode=None)
```

Opens a calendar database and returns a new CalendarDb object.

- If `filename` is `None`, the default database is opened.
- If `filename` is given, it should contain drive letter, colon and file's name, but no absolute path.
- `mode` can be:
  - `None`: Opens an existing calendar database.
  - `'c'`: Opens an existing calendar database, or creates it if it doesn't exist.
  - `'n'`: Creates a new, empty calendar database. If `filename` exists, the previous contents are erased.

7.3.2 CalendarDb Objects

Calendar entries are stored in a calendar database. There is one default calendar database but more calendar databases can be created by invoking `open` with parameters `'n'` or `'c'`.

```python
class CalendarDb
```

CalendarDb objects have the following methods:

- `add_appointment()`
  - Creates and returns a new appointment entry AppointmentEntry. The entry is not added and saved into the database until `Entry.commit` is called.

- `add_event()`
  - Creates and returns a new event entry EventEntry. The entry is not added and saved into the database until `Entry.commit` is called.

- `add_anniversary()`
  - Creates and returns a new anniversary entry AnniversaryEntry. The entry is not added and saved into the database until `Entry.commit` is called.

- `add_todo()`
  - Creates and returns new todo entry TodoEntry. The entry is not added and saved into the database until `Entry.commit` is called.

- `add_reminder()`
  - Creates and returns new reminder entry ReminderEntry. The entry is not added and saved into the database until `Entry.commit` is called.

- `find_instances(start_date, end_date, search_str=u"", appointments=0, events=0, anniversaries=0, todos=0, reminders=0)`
  - The parameters for this function include the start date, end date, search string, and optional parameters. The optional parameters define the entry types to be included into the search. By default all entry types are included. Returns a list that contains Entry instances found in the search. An instance is a dictionary that contains the entry ID and the datetime value. An entry may have several instances if it is repeated, for example once every week, etc.

- `monthly_instances(month, appointments=0, events=0, anniversaries=0, todos=0, reminders=0)`
  - The parameters for this function include `month` (float) and optional parameters. The optional parameters define the entry types to be returned. Returns a list that contains entry instances occurring during the specified calendar month.
daily_instances(day, appointments=0, events=0, anniversaries=0, todos=0)

The parameters for this function include day (float) and optional parameters. The optional parameters define the entry types to be returned. Returns a list that contains entry instances occurring on the specified day.

export_vcalendars((int,...))

Returns a vcalendar string that contains the specified entries in vCalendar format. The parameter for this function is a tuple that contains the entry IDs of the exported entries.

import_vcalendars(string)

Imports vcalendar entries, given in the string parameter, to the database. Returns a list that contains the unique IDs of the imported entries.

__delitem__(id)

Deletes the given calendar Entry from the database. id is the unique ID of the calendar Entry.

__getitem__(id)

Returns a calendar Entry object indicated by the unique ID. The returned object can be one of the following: AppointmentEntry, EventEntry, AnniversaryEntry, ReminderEntry, or TodoEntry. id is the unique ID of the calendar Entry.

7.3.3 Entry Objects

An Entry object represents a live view into the state of a single entry in the database. You can access the entries with an entry’s unique ID. If you create a new entry using db.add_appointment etc., it is saved into the database only if you call the entry’s commit method. In case an entry is already saved into the database, the autocommit mode is on by default and all the changes are automatically saved into the database, unless you call the entry’s begin method. If you call the entry’s begin method, the changes are not saved into the database until you call the entry’s commit method.

Database entries cannot be locked. In other words, other applications are able to make changes to the database entries you are using (not directly to the EntryObjects you are using, but to their representation in the database) at the same time you are modifying them, even if you use begin and commit methods.

class Entry

Entry objects have the following methods and properties:

content

Sets or returns the entry’s content text (Unicode).

commit()

Saves the entry or in case of a new entry adds the entry into the database. Note that this can be called only in case of a new entry, created with db.add_appointment etc., or after begin is called.

rollback()

Undoes the changes made after last commit.

set_repeat(dictionary)

Sets the repeat data of the entry. dictionary is a repeat data dictionary that contains all the repeat rules. For more information on repeat rules, see Section 7.3.4, Repeat Rules.

get_repeat()

Returns the repeat data dictionary of the entry.

location

Sets or returns the entry’s location data (Unicode), for example meeting room information.

set_time(start[, end])

Sets the start and end datetime values of the entry (floats). If only one parameter is given, the other will have the same value.

In case of events, anniversaries, and todo entries the datetime values are truncated to corresponding date values.

TodoEntries can be made undated with TodoEntry.set_time(None). Making the todo entry undated means removing the start and end date and all the repeat rules.
The start datetime value (float) of the entry or None if the start datetime of the entry is not set.

The end datetime value (float) of the entry or None if the end datetime of the entry is not set.

The unique ID of the entry.

The datetime value (float) of the entry’s last modification in universal time.

An integer value indicating if the entry is an originating entry or a modifying entry.

The alarm datetime value (float) for the entry. None if alarm is not set. Alternatively removes the alarm if the value is set to None.

Alarms can be set to all Entry types. However, only alarms set to Appointments and Anniversaries will actually cause an alarm; this is similar to the Calendar application in your Nokia device, which allows you to set an alarm only for Meetings and Anniversaries. In addition, alarms set to any entries residing in a database other than the default database do not cause actual alarms either.

The priority of the entry, which can be an integer ranging from 0 to 255. Native Phonebook and Calendar applications in Nokia devices use value 1 for high priority, 2 for normal priority, and 3 for low priority.

The crossed out value of an entry. Only valid for todo entries. A value that is interpreted as false means that the entry is not crossed out, whereas a value that is interpreted as true means that the entry is crossed out. Note that TodoEntries must also have a cross-out time. If TodoEntry is crossed out using this method, the moment of crossing out is set to the cross-out time of the TodoEntry. See also Section 7.3.3, TodoEntry, cross_out_time.

Sets or returns the entry’s replication status, which can be one of the following: ‘open’, ‘private’, or ‘restricted’.

Returns this entry as a vCalendar string.

AppointmentEntry Objects

class AppointmentEntry

AppointmentEntry class contains no additional methods compared to the Entry class from which it is derived.

EventEntry

class EventEntry

EventEntry class contains no additional methods compared to the Entry class from which it is derived.

AnniversaryEntry

class AnniversaryEntry

AnniversaryEntry class contains no additional methods compared to the Entry class from which it is derived.
ReminderEntry

class ReminderEntry

ReminderEntry class contains no additional methods compared to the Entry class from which it is derived.

TodoEntry

TodoEntry objects represent todo entry types. They have additional properties compared to the Entry class from which they are derived.

class TodoEntry

TodoEntry objects have the following additional properties:

cross_out_time

The cross-out date value of the entry. The value can be None meaning that the entry is not crossed out, or the cross-out date (float). The set value must be date (float). Setting a cross-out time also crosses out the entry. See also Section 7.3.3, Entry Object, crossed_out.

7.3.4 Repeat Rules

Repeat rules specify an entry’s repeat status, that is, the recurrence of the entry. There are six repeat types:

* daily: repeated daily
* weekly: repeat on the specified days of the week, such as Monday and Wednesday, etc.
* monthly_by_dates: repeat monthly on the specified dates, such as the 15th and 17th day of the month
* monthly_by_days: repeat monthly on the specified days, such as the fourth Wednesday of the month, or the last Monday of the month
* yearly_by_date: repeat yearly on the specified date, such as December 24
* yearly_by_day: repeat yearly on the specified day, such as every third Tuesday of May

There are exceptions to repeat rules. For example, you can specify the datetime value (float) in such a way that the entry is not repeated on a specific day even if the repeat rule would specify otherwise.

You must set the start and end dates (floats) of the repeat. The end date can also be set to None to indicate that the repeating continues forever. You can set interval defining how often the repeat occurs, for example in a daily repeat: 1 means every day, 2 means every second day, etc. You can also set the days specifier which lets you explicitly specify the repeat days; for example in a weekly repeat you can set "days":[0,2] which sets the repeat to occur on Mondays and Wednesdays. If you do not set the days specifier, the repeat days are calculated automatically based on the start date.

You can modify repeat data by calling rep_data = entry.get_repeat(), then making changes to rep_data dictionary, and then calling entry.set_repeat(rep_data).

Repeating can be cancelled by calling entry.set_repeat(False), such as entry.set_repeat(None).

Repeat definition examples:

```python
repeat = {"type":"daily", #repeat type
  "exceptions":[exception_day, exception_day+2*24*60*60],
  #no appointment on those days
  "start":appt_start_date, #start of the repeat
```
repeat = {"type":"weekly", #repeat type
  "days":[0,1], #which days in a week (Monday, Tuesday)
  "exceptions":[exception_day], #no appointment on that day
  "start":appt_start_date, #start of the repeat
  "end":appt_start_date+30*24*60*60, #end of the repeat
  "interval":1} #interval (1=every day, 2=every second day etc.)

repeat = {"type":"monthly_by_days", #repeat type
  # appointments on second Tuesday and last Monday of the month
  "days": [{"week":1, "day":1}, {"week":4, "day":0}],
  "exceptions":[exception_day], #no appointment on that day
  "start":appt_start_date, #start of the repeat
  "end":appt_start_date+30*24*60*60, #end of the repeat
  "interval":1} #interval (1=every month, 2=every second month etc.)

repeat = {"type":"monthly_by_dates", #repeat type
  "days": [0,15],
  # appointments on the 1st and 16th day of the month.
  "exceptions":[exception_day], #no appointment on that day
  "start":appt_start_date, #start of the repeat
  "end":appt_start_date+30*24*60*60, #end of the repeat
  "interval":1} #interval (1=every month, 2=every second month etc.)

repeat = {"type":"yearly_by_day", #repeat type
  # appointments on the second Tuesday of February
  "days":{"day":1, "week":1, "month":1},
  "exceptions":[exception_day], #no appointment on that day
  "start":appt_start_date, #start of the repeat
  "end":appt_start_date+3*365*24*60*60, #end of the repeat
  "interval":1} #interval (1=every year, 2=every second year etc.)

repeat = {"type":"yearly_by_date", #repeat type
  "exceptions":[exception_day], #no appointment on that day
  "start":appt_start_date, #start of the repeat
  "end":appt_start_date+3*365*24*60*60, #end of the repeat
  "interval":1} #interval (1=every year, 2=every second year etc.)

7.4 e32db — Interface to the Symbian native DB

The e32db module provides an API for relational database manipulation with a restricted SQL syntax. For details of DBMS support, see the S60 SDK documentation. For examples on using this module, see [6].

The e32db module defines the following functions:

format_rawtime(timevalue)
  Formats timevalue (Symbian time) according to the current system’s date/time formatting rules and returns it as a Unicode string.

format_time(timevalue)
  Formats timevalue according to the current system’s date/time formatting rules and returns it as a Unicode string.
7.4.1 Dbms Objects

class Dbms()
    Creates a Dbms object. Dbms objects support basic operations on a database.

Dbms objects have the following methods:

begin()
    Begins a transaction on the database.

close()
    Closes the database object. It is safe to try to close a database object even if it is not open.

commit()
    Commits the current transaction.

compact()
    Compacts the database, reclaiming unused space in the database file.

create(dbname)
    Creates a database with path dbname.

execute(query)
    Executes an SQL query. On success, returns 0 if a DDL (SQL schema update) statement was executed. Returns the number of rows inserted, updated, or deleted, if a DML (SQL data update) statement was executed.

open(dbname)
    Opens the database in file dbname. This should be a full Unicode path name, for example, u'c:\foo.db'.

rollback()
    Rolls back the current transaction.

7.4.2 DB_view Objects

class Db_view()
    Creates a Db_view object. DB_view objects generate rowsets from a SQL query. They provide functions to parse and evaluate the rowsets.

Db_view objects have the following methods:

col(column)
    Returns the value in column. The first column of the rowset has the index 1. If the type of the column is not supported, a TypeError is raised. See Table 7.1 for a list of supported data types.

col_count()
    Returns the number of columns defined in the rowset.

col_length(column)
    Gets the length of the value in column. Empty columns have a length of zero; non-empty numerical and date/time columns have a length of 1. For text columns, the length is the character count, and for binary columns, the length is the byte count.

col_raw(column)
    Extracts the value of column as raw binary data, and returns it as a Python string. The first column of the rowset has the index 1. See Table 7.1 for a list of supported data types.

col_rawtime(column)
    Extracts the value of a date/time column at index column as a long integer, which represents the raw Symbian time value. The first column of the rowset has the index 1. See Table 7.1 for a list of the supported data types.

col_type(column)
    Returns the numeric type of the given column as an integer from a Symbian-specific list of types. This function is used in the implementation of method col.
count_line()
Returns the number of rows available in the rowset.

first_line()
Positions the cursor on the first row in the rowset.

get_line()
Gets the current row data for access.

is_col_null(column)
Tests whether column is empty. Empty columns can be accessed like normal columns. Empty numerical columns return a 0 or an equivalent value, and text and binary columns have a zero length.

next_line()
Moves the cursor to the next row in the rowset.

prepare(db, query)
Prepares the view object for evaluating an SQL select statement. db is a Dbms object and query the SQL query to be executed.

7.4.3 Mapping Between SQL and Python Data Types

See Table 7.1 for a summary of mapping between SQL and Python data types. The col function can extract any value except LONG VARBINARY and return it as the proper Python value. In addition, the col_raw function can extract any column type except LONG VARCHAR and LONG VARBINARY as raw binary data and return it as a Python string.

Inserting, updating, or searching for BINARY, VARBINARY, or LONG VARBINARY values is not supported. BINARY and VARBINARY values can be read with col or col_raw.

<table>
<thead>
<tr>
<th>SQL type</th>
<th>Symbian column type (in the DBMS C++ API)</th>
<th>Python type</th>
<th>Supported</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIT</td>
<td>EDbColBit</td>
<td>int</td>
<td>yes</td>
</tr>
<tr>
<td>TINYINT</td>
<td>EDbColInt8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>UNSIGNED TINYINT</td>
<td>EDbColUint8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SMALLINT</td>
<td>EDbColInt16</td>
<td></td>
<td></td>
</tr>
<tr>
<td>UNSIGNED SMALLINT</td>
<td>EDbColUint16</td>
<td></td>
<td></td>
</tr>
<tr>
<td>INTEGER</td>
<td>EDbColInt32</td>
<td></td>
<td></td>
</tr>
<tr>
<td>UNSIGNED INTEGER</td>
<td>EDbColUint32</td>
<td></td>
<td></td>
</tr>
<tr>
<td>COUNTER</td>
<td>EDbColUint32 (with the TDbCol::EAutoIncrement attribute)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BIGINT</td>
<td>EDbColInt64</td>
<td>long</td>
<td></td>
</tr>
<tr>
<td>REAL</td>
<td>EDbColReal32</td>
<td>float</td>
<td></td>
</tr>
<tr>
<td>FLOAT</td>
<td>EDbColReal64</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DOUBLE</td>
<td>EDbColReal64</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DOUBLE PRECISION</td>
<td>EDbColReal64</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DATE</td>
<td>EDbColDateTime</td>
<td>float (or long, with col_rawtime())</td>
<td></td>
</tr>
<tr>
<td>TIME</td>
<td>EDbColDateTime</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIMESTAMP</td>
<td>EDbColDateTime</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VARCHAR(n)</td>
<td>EDbColText</td>
<td>Unicode</td>
<td></td>
</tr>
<tr>
<td>LONG VARCHAR</td>
<td>EDbColLongText</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BINARY(n)</td>
<td>EDbColBinary</td>
<td>str</td>
<td>read only</td>
</tr>
<tr>
<td>VARBINARY(n)</td>
<td>EDbColLongBinary</td>
<td>n/a</td>
<td>no</td>
</tr>
<tr>
<td>LONG VARBINARY</td>
<td>EDbColLongBinary</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 7.1: Mapping between SQL and Python types
7.4.4 Date and Time Handling

The functions `col` and `format_time` use Unix time, seconds since January 1, 1970, 00:00:00 UTC, as the time format. Internally the database uses the native Symbian time representation that provides greater precision and range than the Unix time. The native Symbian time format is a 64-bit value that represents microseconds since January 1st 0 AD 00:00:00 local time, nominal Gregorian. BC dates are represented by negative values. Since converting this format to Unix time and back may cause slight round-off errors, you have to use the functions `col_rawtime` and `format_rawtime` if you need to be able to handle these values with full precision.

The representation of date and time literals in SQL statements depends on the current system date and time format. Note that the only accepted ordering of day, month, and year is the one that the system is currently configured to use. Dates in other order are rejected. The recommended way to form date/time literals for SQL statements is to use the functions `format_time` or `format_rawtime` that format the given date/time values properly according to the current system’s date/time format settings.

7.5 e32dbm — DBM implemented using the Symbian native DBMS

The `e32dbm` module provides a DBM API that uses the native Symbian RDBMS as its storage back-end. The module API resembles that of the `gdbm` module. The main differences are:

- The `firstkey()` - `nextkey()` interface for iterating through keys is not supported. Use the "for key in db" idiom or the `keys` or `keysiter` methods instead.
- This module supports a more complete set of dictionary features than `gdbm`
- The values are always stored as Unicode, and thus the values returned are Unicode strings even if they were given to the DBM as normal strings.

7.5.1 Module Level Functions

The `e32dbm` defines the following functions:

`open(dbname[,flags, mode])`

Opens or creates the given database file and returns an `e32dbm` object. Note that `dbname` should be a full path name, for example, `u'c:\foo.db'`. Flags can be:

- `'r'`: opens an existing database in read-only mode. This is the default value.
- `'w'`: opens an existing database in read-write mode.
- `'c'`: opens a database in read-write mode. Creates a new database if the database does not exist.
- `'n'`: creates a new empty database and opens it in read-write mode.

If the character 'f' is appended to flags, the database is opened in fast mode. In fast mode, updates are written to the database only when one of these methods is called: `sync`, `close`, `reorganize`, or `clear`.

Since the connection object destructor calls `close`, it is not strictly necessary to close the database before exiting to ensure that data is saved, but it is still good practice to call the `close` method when you are done with using the database. Closing the database releases the lock on the file and allows the file to be reopened or deleted without exiting the interpreter.

If you plan to do several updates, it is highly recommended that you open the database in fast mode, since inserts and updates are more efficient when they are bundled together in a larger transaction. This is especially important when you plan to insert large amounts of data, since inserting records to e32db is very slow if done one record at a time.
7.5.2 e32dbm Objects

The e32dbm objects returned by the open function support most of the standard dictionary methods. The supported dictionary methods are:

- `__getitem__`
- `__setitem__`
- `__delitem__`
- `has_key`
- `update`
- `__len__`
- `__iter__`
- `iterkeys`
- `iteritems`
- `itervalues`
- `get`
- `setdefault`
- `pop`
- `popitem`
- `clear`

These work the same way as the corresponding methods in a normal dictionary.

In addition, e32dbm objects have the following methods:

**close()**
Closes the database. In fast mode, commits all pending updates to disk. close raises an exception if called on a database that is not open.

**reorganize()**
Reorganizes the database. Reorganization calls compact on the underlying e32db database file, which reclaims unused space in the file. Reorganizing the database is recommended after several updates.

**sync()**
In fast mode, commits all pending updates to disk.
## Standard Library Support and Extensions

### 8.1 Support for Python Standard Library

The standard library support in Python for S60 is summarized in Table 8.1. For API descriptions, see [1].

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Status</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>_testcapi</td>
<td>PYD</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>anydbm</td>
<td>PY</td>
<td>X</td>
<td>DBM API is implemented by PY e32dbm that relies on PYD e32db (see Chapter 7.5, e32dbm Module)</td>
</tr>
<tr>
<td>atexit</td>
<td>PY</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>base64</td>
<td>PY</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>bdb</td>
<td>PY</td>
<td>(X)</td>
<td></td>
</tr>
<tr>
<td>binascii</td>
<td>built-in</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>cmd</td>
<td>PY</td>
<td>(X)</td>
<td></td>
</tr>
<tr>
<td>code</td>
<td>PY</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>codecs</td>
<td>PY</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>codeop</td>
<td>PY</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>copy</td>
<td>PY</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>copy_reg</td>
<td>PY</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>cStringIO</td>
<td>built-in</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>dis</td>
<td>PY</td>
<td>(X)</td>
<td></td>
</tr>
<tr>
<td>errno</td>
<td>built-in</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>exceptions</td>
<td>built-in</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td><strong>future</strong></td>
<td>PY</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>httplib</td>
<td>PY</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>imp</td>
<td>built-in</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>keyword</td>
<td>PY</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>linecache</td>
<td>PY</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>marshal</td>
<td>built-in</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>math</td>
<td>built-in</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>md5¹</td>
<td>built-in</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>mimetools</td>
<td>PY</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>operator</td>
<td>built-in</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>os, os.path</td>
<td>PY</td>
<td>X</td>
<td>Wraps built-in e32posix. Limitations discussed in Section 3.9, Limitations and Areas of Development.</td>
</tr>
<tr>
<td>pdb</td>
<td>PY</td>
<td>(X)</td>
<td></td>
</tr>
<tr>
<td>quopri</td>
<td>PY</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

¹Derived from the RSA Data Security, Inc. MD5 Message-Digest Algorithm.
Table 8.1: Status of library module support.

<table>
<thead>
<tr>
<th>Module</th>
<th>Type</th>
<th>Support Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>random</td>
<td>PY</td>
<td>X</td>
</tr>
<tr>
<td>re</td>
<td>PY</td>
<td>X</td>
</tr>
<tr>
<td>repr</td>
<td>PY</td>
<td>Uses PY _sre as its engine.</td>
</tr>
<tr>
<td>rfc822</td>
<td>PY</td>
<td>X</td>
</tr>
<tr>
<td>select</td>
<td>PY</td>
<td>A minimal implementation: select is supported only for input from sockets.</td>
</tr>
<tr>
<td>sockets</td>
<td>PY</td>
<td>X</td>
</tr>
<tr>
<td>sre</td>
<td>PY</td>
<td>Uses PY _sre.</td>
</tr>
<tr>
<td>string</td>
<td>PY</td>
<td>X</td>
</tr>
<tr>
<td>StringIO</td>
<td>PY</td>
<td>X</td>
</tr>
<tr>
<td>struct</td>
<td>built-in</td>
<td>X</td>
</tr>
<tr>
<td>sys</td>
<td>built-in</td>
<td>X</td>
</tr>
<tr>
<td>thread</td>
<td>built-in</td>
<td>X</td>
</tr>
<tr>
<td>threading</td>
<td>PY</td>
<td>(X)</td>
</tr>
<tr>
<td>time</td>
<td>built-in</td>
<td>X</td>
</tr>
<tr>
<td>traceback</td>
<td>PY</td>
<td>X</td>
</tr>
<tr>
<td>types</td>
<td>PY</td>
<td>X</td>
</tr>
<tr>
<td>urllib</td>
<td>PY</td>
<td>X</td>
</tr>
<tr>
<td>urllib.parse</td>
<td>PY</td>
<td>(urlsplit only)</td>
</tr>
<tr>
<td>uu</td>
<td>PY</td>
<td>X</td>
</tr>
<tr>
<td>warnings</td>
<td>PY</td>
<td>X</td>
</tr>
<tr>
<td>whichdb</td>
<td>PY</td>
<td>X</td>
</tr>
<tr>
<td>xreadlines</td>
<td>built-in</td>
<td>X</td>
</tr>
<tr>
<td>zipfile</td>
<td>PY</td>
<td>X</td>
</tr>
<tr>
<td>zlib</td>
<td>PYD</td>
<td>X</td>
</tr>
</tbody>
</table>

Table 8.1 uses the following coding for module types:

- **PY** – module is implemented in Python.
- **Built-in** – module is a built-in C/C++ module.
- **PYD** – module is a dynamically loadable C/C++ module.

For support status, the following codes are used:

- **X** – included to the Series 60 Python distribution.
- **(X)** – not included to the Series 60 Python distribution, but works both on phone and SDK.
- **Y** – included only to the SDK distribution.

### 8.2 Extensions to Standard Library Modules

The following standard modules have been extended.
8.2.1 thread — S60 extensions to standard thread module

The following function has been added to the standard thread module:

\[
\text{ao\_waittid(thread\_id)}
\]

Synchronizes with the end of the execution of the thread identified by the given thread\_id. The implementation is based on a Symbian OS active object. For the blocking behavior, see Section 4.1.2, Ao\_lock Type.

8.2.2 socket — S60 extensions to standard socket module

Bluetooth (BT) support has been added to the standard socket module. The following related constants and functions are defined:

**Note:** In release 1.0 the functions `bt\_advertise\_service`, `bt\_obex\_receive`, and `bt\_rfcomm\_get\_available\_server\_channel` incorrectly expected to be given the internal e32socket.socket object as the socket parameter instead of the proper socket object. Now the functions work correctly. The old calling convention is still supported but it is deprecated and may be removed in a future release.

**AF\_BT**

Represents the Bluetooth address family.

**BTPROTO\_RFCOMM**

This constant represents the Bluetooth protocol RFCOMM.

**RFCOMM**

**OBEX**

Bluetooth service classes supported by `bt\_advertise\_service`.

**AUTH**

**ENCRYPT**

**AUTHOR**

Bluetooth security mode flags.

**bt\_advertise\_service(name, socket, flag, class)**

Sets a service advertising the service name (Unicode) on local channel that is bound to socket. If flag is True, the advertising is turned on, otherwise it is turned off. The service class to be advertised is either RFCOMM or OBEX.

**bt\_discover\([address]\)**

Performs the Bluetooth device discovery (if the optional BT device address is not given) and the discovery of RFCOMM class services on the chosen device. Returns a pair: BT device address, dictionary of services, where Unicode service name is the key and the corresponding port is the value.

**bt\_obex\_discover\([address]\)**

Same as discover, but for discovery of OBEX class services on the chosen device.

**bt\_obex\_send\_file(address, channel, filename)**

Sends file filename (Unicode) wrapped into an OBEX object to remote address, channel.

**bt\_obex\_receive(socket, filename)**

Receives a file as an OBEX object, unwraps and stores it into filename (Unicode). socket is a bound OBEX socket.

**bt\_rfcomm\_get\_available\_server\_channel(socket)**

Returns an available RFCOMM server channel for socket.

**set\_security(socket, mode)**

Sets the security level of the given bound socket. The mode is an integer flag that is formed using a binary or operation of one or more of: AUTH (authentication), ENCRYPT, AUTHOR (authorization). Example: `set\_security(s, AUTH | AUTHOR)`.

8.2. Extensions to Standard Library Modules 77
Note: When listening to a Bluetooth socket on the phone, it is necessary to set the security level.

Note: SSL is not supported in S60 1st Edition. SSL client certificates are not supported at all.

For examples on the usage of these functions, see Programming with Python for S60 Platform [6].

Setting default Access Point (AP) has been added to the standard socket module. The following related constants and functions are defined:

**select_access_point()**
This opens popup selection where access points are listed and can be selected. Returns selected access point id.

**access_point(apid)**
This creates access point object by given apid. Returns access point object.

**set_default_access_point(apo)**
This sets the default access point that is used when socket is opened. Setting apo to "None" will clear default access point.

**access_points()**
This lists access points id’s and names that are available.

Example 1:

```python
#access point is selected from the list
apid = select_access_point()
apo = access_point(apid)
set_default_access_point(apo)

s = socket(AF_INET, SOCK_STREAM)
print apo.ip()
s.connect(('www.sourceforge.net',80))
s.send('GET /r\n\n\r
\n
s.recv(100)
s.close()
apo.stop()
```

Example 2:

```python
#Access point id is already known
apo = access_point(1)
set_default_access_point(apo)

s = socket(AF_INET, SOCK_STREAM)
s.connect(('www.sourceforge.net',80))
s.send('GET /r\n\n\r
\n
s.recv(100)
s.close()
apo.stop()
```

Example 3:

```python
#display interface ip.
#access point is selected from the list
apid = select_access_point()
apo = access_point(apid)
apo.start()

#Note that ip-address is given by operator, if static ip-address is not defined,
#when connection is started
print apo.ip()

#When connection is closed dynamic ip-address is released
apo.stop()
```
9.1 Python/C API Extensions

The native API exported by the interpreter in S60 environment consists of class `CSPyInterpreter`, Python/C API (see [3]) and a small set of extensions to Python/C API.

9.1.1 class CSPyInterpreter

The class `CSPyInterpreter` offers an interface for initializing the interpreter and for running scripts. It exports the following public interface:

```c

static CSPyInterpreter* NewInterpreterL(TBool aCloseStdlib = ETrue,
                                           void(*aStdioInitFunc)(void*) = NULL,
                                           void* aStdioInitCookie = NULL);
TInt RunScript(int argc, char** argv);
void PrintError();
void (*iStdI)(char* buf, int n);
void (*iStdO)(const char* buf, int n);
```

The caller of the constructor `CSPyInterpreter::NewInterpreterL()` may provide its own function `aStdioInitFunc` for initializing Symbian OS STDLIB's standard I/O descriptors. It gets called with the argument `aStdioInitCookie`. The `CSPyInterpreter` class can also be requested to leave STDLIB open at its destruction.

The `RunScript` method establishes a Python interpreter context and runs the script file whose full path name is in `argv[0]` with the given argument vector. After completion, it leaves the interpreter context and returns a Symbian error code to indicate success or failure.

The `CSPyInterpreter::PrintError` method can be used to print current Python exception information to the standard error output.

9.1.2 Extensions to Python/C API

Defined in `symbian_python_ext_util.h`

```python
PyObject* SPyErr_SetFromSymbianOSErr(int error)
    Sets Python exception of type `PyExc_SymbianError` with the value field set to symbolic name of the Symbian OS enumeration value `error` and returns NULL. In case `error` has the special value `KErrPython`, it assumes that a Python exception has already been set and returns NULL.
```

The following functions can be used for storing the global data in a module implementation. They are thin wrappers around `PyDict_SetItem`, `PyDict_SetItemString`, `PyDict_GetItem`, `PyDict_GetItemString`, `PyDict_DelItem` and `PyDict_DelItemString`, respectively, and can be used in the same way. The data is stored in a special completely global dictionary shared by all modules and threads in the current interpreter.
int SPyAddGlobal(PyObject *key, PyObject *value)
int SPyAddGlobalString(char *key, PyObject *value)
PyObject* SPyGetGlobal(PyObject *key)
PyObject* SPyGetGlobalString(char *key)
void SPyRemoveGlobal(PyObject *key)
void SPyRemoveGlobalString(char *key)

Defined in python_globals.h

PyThreadState* PYTHON_TLS->thread_state
Current thread state.

Thread state and interpreter lock management must be performed according to the instructions; see [3]. Python for S60 Platform extends the Python/C API by offering a facility for querying the related Python thread state (PYTHON_TLS->thread_state) from the context of the currently running thread. This can be used to re-establish the interpreter context with PyEval_RestoreThread in C/C++ code.

To save/restore the interpreter context:

    Py_BEGIN_ALLOW_THREADS
    /* ...your code... */
    Py_END_ALLOW_THREADS

To restore/save the interpreter context:

    PyEval_RestoreThread(PYTHON_TLS->$thread_state)
    /* ...your code... */
    PyEval_SaveThread()

Defined in pythread.h

int PyThread_AtExit(void(*)())
An extension to the standard thread module’s C API that can be used for registering thread-specific exit functions. In the main thread calling this function has the same effect as calling Py_AtExit. For more information, see [1].

9.2 Extending Python for S60

The general rules and guidelines for writing Python extensions apply in the S60 Python environment as well; for more information, see [2]. The Python/C API is available, see [3] In addition, for an example on porting a simple extension to S60, see [6].

The issues that need to be considered in the implementation of the extension modules include:

• Preparation of the data structures that make the C/C++ coded extensions visible to the Python interpreter and make it possible to perform calls from Python to C/C++ code
• Conversions between C/C++ representations of the Python objects and object types used in the extension code
• Maintenance of the reference counts of the C/C++ representations of the Python objects
• Passing of exceptions between C/C++ code and Python
• Management of interpreter’s thread state and the interpreter lock
In addition to the concerns common for all Python C extensions, the following principles should be considered when implementing new Python interfaces in the S60 environment:

- Maximize the usage of Python’s built-in types at the interfaces.
- Related to the above: design interfaces in such a way that information can be passed between them with minimal conversions.
- Convert Symbian operating system exceptions / errors to Python exceptions.
- Unicode strings are used at the interfaces to represent text that gets shown on the GUI. They can be passed to and from Symbian operating system without conversions.
- While performing potentially long-lasting / blocking calls from an extension implementation to services outside the interpreter, the interpreter lock must be released and then re-acquired after the call.
- Rather than always implementing a thin wrapper on top of a Symbian OS facility, consider the actual task for which the script writer needs the particular interface. For example, if the task involves interaction with the users using the GUI, the script writer’s interest may well be limited to performing the interaction / information exchange in a way that is compatible with the UI style rather than having full control of the low-level details of the GUI implementation.
- The C/C++ implementation of a Python interface should be optimized for performance and covering access to the necessary features of the underlying Platform. Where necessary, the Python programming interface can be further refined by wrapper modules written in Python.

An extension module is packaged in its own dynamically loadable library that must be installed into ‘\system\libs’ directory and named ‘module_name.pyd’. The module initialization function must be exported at ordinal 1. The module identification is based on the filename only. As a special feature of PyS60, an optional module finalizer function may be exported at ordinal 2.

The macro versions of memory-management functions PyMem_MALLOC and PyObject_NEW are not included. Use the functions PyMem_Malloc and PyObject_New instead.

9.2.1 Services for Extensions

S60 Python Platform implements an adaptation layer between S60 UI application framework and script language UI extensions to simplify UI extension development. This API is used by the implementation of the appuifw module but not exported in the current release. Some general utility services for extensions are also provided, see Chapter 9.1.

9.2.2 Example

This extension code snippet demonstrates some of the issues mentioned in this chapter, such as:

- Conversion from Python data types, usage of built-in data types at extension interface, usage of Unicode strings (lines 8-12)
- Maintenance of the reference counts (line 36)
- Passing of exceptions between C/C++ code and Python (line 34)
- Releasing the interpreter lock while performing a blocking call to a service outside the interpreter (lines 29, 31)
- Simplifying the API to the note facility of the Platform
extern "C" PyObject *
note(PyObject * /*self*/, PyObject *args)
{
    TInt error = KErrNone;
    int l_tx, l_ty;
    char *b_tx, *b_ty;

    if (!PyArg_ParseTuple(args, "u#s#", &b_tx, &l_tx, &b_ty, &l_ty))
        return NULL;

    TPtrC8 stype((TUint8*)b_ty, l_ty);
    TPtrC note_text((TUint16 *)b_tx, l_tx);
    CAknResourceNoteDialog* dlg = NULL;

    if (stype.Compare(KErrorNoteType) == 0)
        dlg = new CAknErrorNote(ETrue);
    else if (stype.Compare(KInfoNoteType) == 0)
        dlg = new CAknInformationNote(ETrue);
    else if (stype.Compare(KConfNoteType) == 0)
        dlg = new CAknConfirmationNote(ETrue);
    else {
        PyErr_BadArgument();
        return NULL;
    }

    if (dlg == NULL)
        return PyErr_NoMemory();

    Py_BEGIN_ALLOW_THREADS
    TRAP(error, dlg->ExecuteLD(note_text));
    Py_END_ALLOW_THREADS

    if (error != KErrNone)
        return SPyErr_SetFromSymbianOSErr(error);
    else {
        Py_INCREF(Py_None);
        return Py_None;
    }
}
The following list defines the terms and abbreviations used in this document:

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>AAC; Adaptive Audio Coding</td>
<td>AAC provides basically the same sound quality as MP3 while using a smaller bit rate. AAC is mainly used to compress music.</td>
</tr>
<tr>
<td>Advertise</td>
<td>Advertise service in Bluetooth makes it known that a certain Bluetooth service is available.</td>
</tr>
<tr>
<td>AMR</td>
<td>Adaptive Multi-rate Codec file format.</td>
</tr>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>Bluetooth</td>
<td>Bluetooth is a technology for wireless communication between devices that is based on a low-cost short-range radio link.</td>
</tr>
<tr>
<td>BPP</td>
<td>Bits Per Pixel</td>
</tr>
<tr>
<td>C STDLIB</td>
<td>Symbian OS’s implementation of the C standard library</td>
</tr>
<tr>
<td>Dialog</td>
<td>A temporary user interface window for presenting context-specific information to the user, or prompting for information in a specific context.</td>
</tr>
<tr>
<td>Discovery</td>
<td>Discovery is a process where Bluetooth finds other nearby Bluetooth devices and their advertised services.</td>
</tr>
<tr>
<td>DLL</td>
<td>Dynamic link library</td>
</tr>
<tr>
<td>GSM; Global System for Mobile communication</td>
<td>GSM is a digital mobile telephone system that uses a variation of time division multiple access. It digitizes and compresses data, then sends it down a channel with two other streams of user data, each in its own time slot.</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User Interface</td>
</tr>
<tr>
<td>I/O</td>
<td>input/output</td>
</tr>
<tr>
<td>IP</td>
<td>Internet Protocol</td>
</tr>
<tr>
<td>MBM; Multi-BitMap</td>
<td>The native Symbian OS format used for pictures. MBM files can be generated with the bmconv.exe tool included in the S60 SDK.</td>
</tr>
<tr>
<td>MIDI; Musical Instrument Digital Interface</td>
<td>A protocol and a set of commands for storing and transmitting information about music.</td>
</tr>
<tr>
<td>MIF; Multi-Image File</td>
<td>MIF files are similar to MBM files and can contain compressed SVG-T files. This file type can be generated with the MifConv.exe tool.</td>
</tr>
<tr>
<td>MIME; Multi-purpose Internet Mail Extensions</td>
<td>MIME is an extension of the original Internet e-mail protocol that can be used to exchange different kinds of data files on the Internet.</td>
</tr>
<tr>
<td>MP3</td>
<td>A standard technology and format for compressing a sound sequence into a very small file while preserving the original level of sound quality when it is played.</td>
</tr>
<tr>
<td>OS</td>
<td>Operating System</td>
</tr>
<tr>
<td>Real Audio</td>
<td>An audio format developed by Real Networks.</td>
</tr>
<tr>
<td>RDBMS</td>
<td>Relational database management system</td>
</tr>
<tr>
<td>SMS; Short Message System (within GSM)</td>
<td>SMS is a service for sending messages of up to 160 characters, or 224 characters if using a 5-bit mode, to mobile phones that use GSM communication.</td>
</tr>
<tr>
<td><strong>Term</strong></td>
<td><strong>Definition</strong></td>
</tr>
<tr>
<td>---------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Softkey</td>
<td>Softkey is a key that does not have a fixed function nor a function label printed on it. On a phone, selection keys reside below or above on the side of the screen, and derive their meaning from what is presently on the screen.</td>
</tr>
<tr>
<td>SQL</td>
<td>Structured Query Language</td>
</tr>
<tr>
<td>SVG, SVG-T;</td>
<td>XML-based vector graphics format for describing two-dimensional graphics and graphical applications.</td>
</tr>
<tr>
<td>Scalable</td>
<td></td>
</tr>
<tr>
<td>Vector</td>
<td></td>
</tr>
<tr>
<td>Graphics</td>
<td></td>
</tr>
<tr>
<td>(-Tiny)</td>
<td></td>
</tr>
<tr>
<td>Twip</td>
<td>Twips are screen-independent units to ensure that the proportion of screen elements are the same on all display systems. A twip is defined as 1/1440 of an inch, or 1/567 of a centimeter.</td>
</tr>
<tr>
<td>UI</td>
<td>User Interface</td>
</tr>
<tr>
<td>UI control</td>
<td>UI control is a GUI component that enables user interaction and represents properties or operations of an object.</td>
</tr>
<tr>
<td>WAV</td>
<td>A file format for recording sound, especially in multimedia applications.</td>
</tr>
</tbody>
</table>


[8] Developers section on the S60 Platform Web site (for all S60 devices), http://www.s60.com/


In order to improve the quality of Python for S60 the developers would like to know of any deficiencies you find in Python for S60 or its documentation.

Before submitting a report, you will be required to log into SourceForge; this will make it possible for the developers to contact you for additional information if needed. It is not possible to submit a bug report anonymously.

All bug reports should be submitted via the project PyS60 Bug Tracker on SourceForge (http://sourceforge.net/tracker/?group_id=154155). The bug tracker offers a Web form which allows pertinent information to be entered and submitted to the developers.

The first step in filing a report is to determine whether the problem has already been reported. The advantage in doing so, aside from saving the developers time, is that you learn what has been done to fix it; it may be that the problem has already been fixed for the next release, or additional information is needed (in which case you are welcome to provide it if you can!). To do this, search the bug database using the search box near the bottom of the page.

If the problem you’re reporting is not already in the bug tracker, go back to the project PyS60 Bug Tracker (http://sourceforge.net/tracker/?group_id=154155). Select the “Submit a Bug” link at the top of the page to open the bug reporting form.

The submission form has a number of fields. The only fields that are required are the “Summary” and “Details” fields. For the summary, enter a very short description of the problem; less than ten words is good. In the Details field, describe the problem in detail, including what you expected to happen and what did happen. Be sure to include the version of Python for S60 you used, whether any extension modules were involved and what hardware (the S60 device model or emulator) you were using, including version information of the S60 SDK and your device firmware version as appropriate. You can see the device firmware version by entering *#0000# on the device keypad - please include all information that is shown by this code.

The only other field that you may want to set is the “Category” field, which allows you to place the bug report into a broad category (such as “Documentation” or “Library”).

Each bug report will be assigned to a developer who will determine what needs to be done to correct the problem. You will receive an update each time action is taken on the bug.

See Also:

* How to Report Bugs Effectively  
  (http://www-mice.cs.ucl.ac.uk/multimedia/software/documentation/ReportingBugs.html)
  Article which goes into some detail about how to create a useful bug report. This describes what kind of information is useful and why it is useful.

* Bug Writing Guidelines  
  (http://www.mozilla.org/quality/bug-writing-guidelines.html)  
  Information about writing a good bug report. Some of this is specific to the Mozilla project, but describes general good practices.
A
appuifw, 13
audio, 47

C
calendar, 60, 65
camera, 33
contacts, 55

E
e32, 9
e32db, 70
e32dbm, 73

G
glcanvas, 44
gles, 38
graphics, 27

I
inbox, 50

K
keycapture, 35

L
location, 51

M
messaging, 49

P
position, 52

S
socket, 77
sysinfo, 11

T
telephone, 49
thread, 77
topwindow, 36
INDEX

Symbols
__delitem__() (CalendarDb method), 61, 67
__delitem__() (ContactDb method), 56
__delitem__() (Contact method), 58
__getitem__() (CalendarDb method), 61, 67
__getitem__() (array method), 38
__len__() (array method), 38
__setitem__() (array method), 38

A
access_point() (in module socket), 78
access_points() (in module socket), 78
activate_tab() (Application method), 18
active_profile() (in module sysinfo), 11
add() (Text method), 22
add_anniversary() (CalendarDb method), 61, 66
add_appointment() (CalendarDb method), 61, 66
add_contact() (ContactDb method), 55
add_event() (CalendarDb method), 61, 66
add_field() (Contact method), 57
add_group() (Groups method), 59
add_image() (TopWindow method), 36
add_reminder() (CalendarDb method), 66
add_todo() (CalendarDb method), 61, 66
add_todo_list() (CalendarDb method), 61
address() (Inbox method), 51
AF_BT (data in socket), 77
after() (Ao_timer method), 11
alarm (Entry attribute), 63, 68
all_keys (data in keycapture), 35
AnniversaryEntry (class in calendar), 63, 68
ao_callgate() (in module e32), 9
Ao_lock (class in e32), 10
ao_sleep() (in module e32), 9
Ao_timer (class in e32), 11
ao_waittid() (in module thread), 77
ao_yield() (in module e32), 9
Application (class in appuifw), 16
AppointmentEntry (class in calendar), 63, 68
Canvas (class in appuifw), 25
appuifw (standard module), 13
arc() (method), 32
array (class in gles), 38
as_vcalendar() (Entry method), 63, 68
as_vcard() (Contact method), 57
audio (extension module), 47

AUTH (data in socket), 77
AUTHOR (data in socket), 77
available_fonts() (in module appuifw), 15

B
background_color (TopWindow attribute), 37
battery() (in module sysinfo), 11
begin()
  Contact method, 56
  Dbms method, 71
bind()
  GLCanvas method, 45
  Inbox method, 51
  Listbox method, 24
  Text method, 22
blit() (method), 32
body (Application attribute), 16
bt_advertise_service() (in module socket), 77
bt_discover() (in module socket), 77
bt_obex_discover() (in module socket), 77
bt_obex_receive() (in module socket), 77
bt_obex_send_file() (in module socket), 77
bt_rfcomm_get_available_server_channel() (in module socket), 77

BTPROTORFCOMM (data in socket), 77

C
calendar (extension module), 60, 65
CalendarDb (class in calendar), 61, 66
camera (extension module), 33
cameras_available() (in module camera), 33
cancel() (Ao_timer method), 11
Canvas (class in appuifw), 25
clear()
  method, 32
  Text method, 22
close()
  Dbms method, 71
e32dbm method, 74
  Sound method, 48
col() (Db_view method), 71
col_count() (Db_view method), 71
col_length() (Db_view method), 71
col_raw() (Db_view method), 71
col_rawtime() (Db_view method), 71
col_type() (Db_view method), 71
color (Text attribute), 21
commit()
    Contact method, 56
    Dbms method, 71
    Entry method, 62, 67
compact()
    CalendarDb method, 62
    ContactDb method, 56
    Dbms method, 71
compact_required() (ContactDb method), 56
Contact (class in contacts), 56
ContactDb (class in contacts), 55
ContactField (class in contacts), 59
contacts (extension module), 55
content() (Inbox method), 51
content (Entry attribute), 62, 67
Content (class in appuifw), 24
corner_type (TopWindow attribute), 37
count_line() (Db_view method), 72
create() (Dbms method), 71
cross_out_time (TodoEntry attribute), 63, 69
crossed_out (Entry attribute), 63, 68
current() (Listbox method), 24
current_position() (Sound method), 49
current_volume() (Sound method), 48

daily_instances() (CalendarDb method), 61, 67
Db_view (class in e32db), 71
Dbms (class in e32db), 71
default_list (TodoListDict attribute), 64
default_module() (in module position), 52
delete()
    Inbox method, 51
    Text method, 22
dial() (in module telephone), 49
display_pixels() (in module sysinfo), 11
display_twips() (in module sysinfo), 11
drawNow() (GLCanvas method), 45
drive_list() (in module e32), 9
duration() (Sound method), 48

E
e32 (extension module), 9
e32db (extension module), 70
e32dbm (module), 73
EAColumn (data in appuifw), 19
EApplicationWindow (data in appuifw), 18
EBatteryPane (data in appuifw), 19
EBColumn (data in appuifw), 19
EColumn (data in appuifw), 19
EControlPane (data in appuifw), 18
EControlPaneBottom (data in appuifw), 19
EControlPaneTop (data in appuifw), 19
ECreated (data in messaging), 49
EDColumn (data in appuifw), 19
EDeleted (data in messaging), 49
EDraft (data in inbox), 50
EFatalServerError (data in messaging), 50
EFindPane (data in appuifw), 19
EInbox (data in inbox), 50
IndicatorPane (data in appuifw), 19
ellipse() (method), 32
EMainPane (data in appuifw), 18
EMovedToOutBox (data in messaging), 49
ENaviPane (data in appuifw), 19
ENDECRYPT (data in socket), 77
end_time (Entry attribute), 62, 68
ENoServiceCentre (data in messaging), 50
ENotReady (data in audio), 47
Entry (class in calendar), 62, 67
EOpen (data in audio), 47
EOutbox (data in inbox), 50
EPlaying (data in audio), 47
ERecording (data in audio), 47
EScheduledForSend (data in messaging), 49
EScheduleFailed (data in messaging), 50
EScreen (data in messaging), 18
ESendFailed (data in appuifw), 50
ESent
    data in inbox, 50
    data in messaging, 49
ESignalPane (data in appuifw), 18
ESTaconBottom (data in appuifw), 19
ESTaconTop (data in appuifw), 19
ESTatusPane (data in appuifw), 18
ESTatusPaneBottom (data in appuifw), 19
ESTatusPaneTop (data in appuifw), 19
ETitlePane (data in appuifw), 18
EUnclassifiedIndicatorPane (data in appuifw), 19
EventEntry (class in calendar), 63, 68
EWallpaperPane (data in appuifw), 19
execute()
    Dbms method, 71
    Form method, 20
exit_key_handler (Application attribute), 17
export_vcalendars() (CalendarDb method), 61, 67
export_vcards() (ContactDb method), 56
exposure_modes() (in module camera), 33

F
FFormAutoFormEdit (data in appuifw), 20
FFormAutoLabelEdit (data in appuifw), 20
FFormDoubleSpaced (data in appuifw), 20
FFormEditModeOnly (data in appuifw), 20
FFormViewModeOnly (data in appuifw), 20
field_types() (ContactDb method), 56
file_copy() (ContactDb method), 56
find()
    Contact method, 58
    ContactDb method, 56
find_instances() (CalendarDb method), 61, 66
first_line() (Db_view method), 72
flags (Form attribute), 20
Q
query() (in module appuifw), 15

R
record() (Sound method), 48
rectangle() (method), 32
release() (in module camera), 35
ReminderEntry (class in calendar), 69
remove_image() (TopWindow method), 77
ring_type() (in module sysinfo), 12
rollback()
  Contact method, 57
  Dbms method, 71
  Entry method, 62, 67

S
s60_version_info (data in e32), 10
save() (Image method), 28
save_hook (Form attribute), 20
say() (in module audio), 47
schema (ContactField attribute), 59
screen (Application attribute), 17
screenshot() (in module graphics), 27
select_access_point() (in module socket), 78
select_module() (in module position), 52
selection_list() (in module appuifw), 16
set() (Text method), 23
set_default_access_point() (in module socket), 78
set_exit() (Application method), 18
set_home_time() (in module e32), 9
set_list() (Listbox method), 24
set_pos() (Text method), 23
set_position() (Sound method), 48
set_repeat() (Entry method), 62, 67
set_requestors() (in module position), 52
set_security() (in module socket), 77
set_tabs() (Application method), 18
set_time() (Entry method), 62, 67
set_volume() (Sound method), 48
shadow (TopWindow attribute), 37
show() (TopWindow method), 36
signal() (Ao_lock method), 11
signal_bars() (in module sysinfo), 12
signal_dbm() (in module sysinfo), 12
size
  Canvas attribute, 27
  Image attribute, 29
  TopWindow attribute, 37
sms_messages() (Inbox method), 51
sms_send() (in module messaging), 49
socket (extension module), 77
Sound (class in audio), 47
Sound.open() (in module audio), 47
SPyAddGlobal(), 80
SPyAddGlobalString(), 80
SPyErr_SetFromSymbianOSErr(), 79
SPyGlobal(), 80
SPyGetGlobalString(), 80
SPyRemoveGlobal(), 80
SPyRemoveGlobalString(), 80
start() (KeyCapturer method), 36
start_exe() (in module e32), 10
start_finder() (in module camera), 35
start_server() (in module e32), 10
start_time (Entry attribute), 62, 68
state() (Sound method), 48
stop()
  Image method, 29
  KeyCapturer method, 36
  Sound method, 48
stop_finder() (in module camera), 35
style (Text attribute), 21
STYLE_BOLD (data in appuifw), 22
STYLE_ITALIC (data in appuifw), 22
STYLE_STRIKETHROUGH (data in appuifw), 22
STYLE_UNDERLINE (data in appuifw), 22
sw_version() (in module sysinfo), 12
sync() (e32dbm method), 74
sysinfo (extension module), 36

take_photo() (in module camera), 33
telephone (extension module), 49
text() (method), 32
thread (extension module), 77
time() (Inbox method), 51
title
  Application attribute, 17
  Contact attribute, 56
todo_list (TodoEntry attribute), 64
todo_lists (CalendarDb attribute), 61
TodoEntry (class in calendar), 63, 69
TodoList (class in calendar), 64
TodoListDict (class in calendar), 64
TopWindow (class in topwindow), 36
topwindow (extension module), 36
total_ram() (in module topwindow), 36
total_rom() (in module sysinfo), 12
transpose() (Image method), 28
type (ContactField attribute), 59

U
uid() (Application method), 18
unread() (Inbox method), 51
value (ContactField attribute), 59
visible (TopWindow attribute), 37

Index
W

wait() (Ao_lock method), 10
white_balance_modes() (in module camera), 33