

# Python for Series 60 Platform API Reference

Version 1.2; September 28, 2005

# Python for Series 60 Platform

**NOKIA**

Copyright © 2005 Nokia Corporation. All rights reserved.

Nokia and Nokia Connecting People are registered trademarks of Nokia Corporation. Other product and company names mentioned herein may be trademarks or trade names of their respective owners.

#### **Disclaimer**

The information in this document is provided "as is," with no warranties whatsoever, including any warranty of merchantability, fitness for any particular purpose, or any warranty otherwise arising out of any proposal, specification, or sample. Furthermore, information provided in this document is preliminary, and may be changed substantially prior to final release. This document is provided for informational purposes only.

Nokia Corporation disclaims all liability, including liability for infringement of any proprietary rights, relating to implementation of information presented in this document. Nokia Corporation does not warrant or represent that such use will not infringe such rights.

Nokia Corporation retains the right to make changes to this specification at any time, without notice.

#### **License**

A license is hereby granted to download and print a copy of this specification for personal use only. No other license to any other intellectual property rights is granted herein.

## Contents

<b>1</b>	<b>Introduction</b> .....	<b>7</b>
1.1	Scope.....	7
1.2	Audience.....	7
1.3	New in Release 1.2.....	8
1.4	Typographical Conventions.....	8
1.5	Naming Conventions.....	8
<b>2</b>	<b>API Summary</b> .....	<b>9</b>
2.1	Python Standard Library.....	9
2.2	Python for Series 60 Extensions.....	9
2.2.1	Built-in extensions.....	9
2.2.2	Dynamically loadable extensions.....	9
2.3	Third-Party Extensions.....	9
<b>3</b>	<b>Python for Series 60 UI Programming Model</b> .....	<b>10</b>
3.1	Overview.....	10
3.2	Basics of appuifw Module.....	11
3.3	appuifw UI Controls.....	11
3.4	Softkeys.....	12
<b>4</b>	<b>Selected Issues on Python Programming for Series 60</b> .....	<b>13</b>
4.1	Concurrency Aspects.....	13
4.2	Current Series 60 Python Script Execution Environment.....	13
4.3	Standard I/O Streams.....	14
4.4	Usage of Unicode.....	14
4.5	Date and Time.....	14
4.6	Sharing Native Resources between Threads.....	14
4.7	Scalable User Interface.....	14
4.8	Error Handling.....	15
4.9	Limitations and Areas of Development.....	15
<b>5</b>	<b>appuifw Module</b> .....	<b>16</b>
5.1	Module Level Functions.....	16
5.2	Application Type.....	18
5.3	Form Type.....	19
5.4	Text Type.....	20
5.5	Listbox Type.....	23
5.6	Icon Type.....	24
5.7	Content_handler Type.....	25
5.8	Canvas Type.....	25

<b>6</b>	<b>graphics Module</b> .....	<b>28</b>
6.1	Module Level Functions.....	28
6.2	Image Class Static Methods.....	28
6.3	Image Objects.....	29
6.4	Common Features of Drawable Objects.....	30
6.4.1	Options .....	30
6.4.2	Coordinate representation .....	31
6.4.3	Color representation.....	31
6.4.4	Font specifications.....	32
6.4.5	Common Methods of Drawable Objects .....	32
<b>7</b>	<b>e32 Module</b> .....	<b>34</b>
7.1	Module Level Functions .....	34
7.2	Ao_lock Type.....	35
<b>8</b>	<b>e32db Module</b> .....	<b>36</b>
8.1	Module Level Functions.....	36
8.2	Dbms Type .....	36
8.3	DB_view Type.....	37
8.4	Mapping Between SQL and Python Data Types.....	38
8.5	Date and Time Handling .....	38
<b>9</b>	<b>e32dbm Module</b> .....	<b>40</b>
9.1	Module Level Functions.....	40
9.2	e32dbm Objects .....	40
<b>10</b>	<b>messaging Module</b> .....	<b>42</b>
<b>11</b>	<b>location Module</b> .....	<b>43</b>
<b>12</b>	<b>sysinfo Module</b> .....	<b>44</b>
<b>13</b>	<b>camera Module</b> .....	<b>46</b>
<b>14</b>	<b>audio Module</b> .....	<b>48</b>
14.1	Sound Class Static Methods.....	48
14.2	Sound Objects.....	48
<b>15</b>	<b>telephone Module</b> .....	<b>50</b>
<b>16</b>	<b>calendar Module</b> .....	<b>51</b>
16.1	Module Level Functions.....	52
16.2	CalendarDb Objects.....	52
16.3	Entry Objects .....	53
16.3.1	AppointmentEntry Objects.....	55
16.3.2	EventEntry.....	55
16.3.3	AnniversaryEntry .....	55
16.3.4	TodoEntry .....	55

16.3.5	ToDoListDict .....	55
16.3.6	ToDoList.....	56
16.4	Repeat Rules .....	56
<b>17</b>	<b>contacts Module .....</b>	<b>58</b>
17.1	Module Level Functions .....	58
17.2	ContactDb Object .....	58
17.3	Contact Object.....	59
17.4	ContactField Object.....	61
<b>18</b>	<b>Extensions to Standard Library Modules .....</b>	<b>62</b>
18.1	thread Module .....	62
18.2	socket Module.....	62
<b>19</b>	<b>Terms and Abbreviations .....</b>	<b>64</b>
<b>20</b>	<b>References .....</b>	<b>66</b>
<b>Appendix A</b>	<b>Python Library Module Support.....</b>	<b>67</b>
<b>Appendix B</b>	<b>Extensions to C API.....</b>	<b>70</b>
B.1	class CPyObject.....	70
B.2	Extensions to C API.....	70
<b>Appendix C</b>	<b>Extending Series 60 Python .....</b>	<b>72</b>
C.1	Overview .....	72
C.2	Services for Extensions .....	73
C.3	Example .....	73

## Change History

December 10, 2004	Version 1.0	Initial document release.
June 29, 2005	Version 1.1.5	Sections 1.3, 3.3, 4.5, 4.6, 4.7, 5.6, 5.8, 6, 12, 13, 14, 15, 16, and 17 added. Sections 1, 3.1, 3.2, 5.1, 5.2, 5.4, 5.5, 7.1, and 18.2 updated.
September 28, 2005	Version 1.2	Section 4.7 added. Sections 1.3, 5.1, 5.2, 5.4, 5.5, 5.6, 13, and 19 updated.

# 1 Introduction

The Python for Series 60 Platform (Python for Series 60) simplifies application development and provides a scripting solution for the Symbian C++ APIs. This document is for Python for Series 60 release 1.2 that is based on Python 2.2.2.

The documentation for Python for Series 60 includes three documents:

- *Getting Started with Python for Series 60 Platform [5]* contains information on how to install Python for Series 60 and how to write your first program.
- This document contains API and other reference material.
- *Programming with Python for Series 60 Platform [6]* contains code examples and programming patterns for Series 60 devices that can be used as a basis for programs.

Python for Series 60 as installed on a Series 60 device consists of:

- **Python** execution environment, which is visible in the application menu of the device and has been written in Python on top of Python for Series 60 Platform (see *Series 60 SDK documentation [4]*)
- Python interpreter DLL
- Standard and proprietary Python library modules
- Series 60 UI application framework adaptation component (a DLL) that connects the scripting domain components to the Series 60 UI
- Python Installer program for installing Python files on the device, which consists of:
  - Recognizer plug-in
  - Symbian application written in Python

**Tip:** The Python for Series 60 developer discussion board [9] on the Forum Nokia Web site is a useful resource for finding out information on specific topics concerning Python for Series 60. You are welcome to give feedback or ask questions about Python for Series 60 through this discussion board.

## 1.1 Scope

This document includes the information required by developers to create applications that use Python for Series 60, 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 Series 60 phones. The reader should be familiar with the Python programming language (<http://www.python.org>) and the basics of using Python for Series 60 (see *Getting Started with Python for Series 60 Platform [5]*).

### 1.3 New in Release 1.2

This section lists the updates in this document since release 1.1.6.

- Section 4.7, *Scalable User Interface* has been added.
- There are some general updates in 5.1, *Module Level Functions* and 5.2, *Application Type*.
- Section 5.4, *Text Type* has been updated to include a `delete` method.
- Sections 5.5, *Listbox Type* and 5.6, *Icon Type* have been updated with SVG-T support information.
- Chapter 13, *camera Module* has been updated with new functions.

### 1.4 Typographical Conventions

The following typographical conventions are used in this document:

<b>Bold</b>	Bold is used to indicate windows, views, pages and their elements, menu items, and button names.
<i>Italic</i>	Italics are used when referring to another chapter or section in the document and when referring to a manual. Italics are also used for key terms and emphasis.
<code>Courier</code>	Courier is used to indicate parameters, file names, processes, commands, directories, and source code text.
>	Arrows are used to separate menu items within a path.

### 1.5 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.

## 2 API Summary

All built-in object types of the Python language are supported in the Series 60 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 Series 60 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 Series 60 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 *Appendix A, Python Library Module Support*.

When Python, available at <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 *G. van Rossum, and F.L. Drake, Jr., editor. [Python] Library Reference. [1]*

Python for Series 60 extends some standard modules. These extensions are described in this document, see Chapter 18, *Extensions to Standard Library Modules*.

### 2.2 Python for Series 60 Extensions

There are two kinds of native C++ extensions in the Python for Series 60 Platform: built-in extensions and dynamically loadable extensions.

#### 2.2.1 Built-in extensions

There are two built-in extensions in the Python for Series 60 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 Series 60 Platform offers UI application framework related Python interfaces.

#### 2.2.2 Dynamically loadable extensions

These dynamically loadable extension modules provide proprietary APIs to Series 60 Platform's services: `graphics` (see Chapter 6, *graphics Module*), `e32db` (see Chapter 8, *e32db Module*), `messaging` (see Chapter 10, *messaging Module*), `location` (see Chapter 11, *location Module*), `sysinfo` (see Chapter 12, *sysinfo Module*), `camera` (see Chapter 13, *camera Module*), `audio` (see Chapter 14, *audio Module*), `telephone` (see Chapter 15, *telephone Module*), `calendar` (see Chapter 16, *calendar Module*), and `contacts` (see Chapter 17, *contacts Module*).

### 2.3 Third-Party Extensions

It is also possible to write your own Python extensions. Series 60 related extensions to Python/C API are described in *Appendix B, Extensions to C API*. For some further guidelines on writing extensions in C/C++, see *Appendix C, Extending Series 60 Python*. In addition, for an example on porting a simple extension to Series 60, see *Programming with Python for Series 60 Platform [6]*.

### 3 Python for Series 60 UI Programming Model

This chapter gives an outline of the conceptual model of UI application programming that underlies the APIs described in the following chapters.

#### 3.1 Overview

Figure 1 shows the Python for Series 60 environment for UI application programming. The built-in `appuifw` Python module provides an interface to the Series 60 framework.

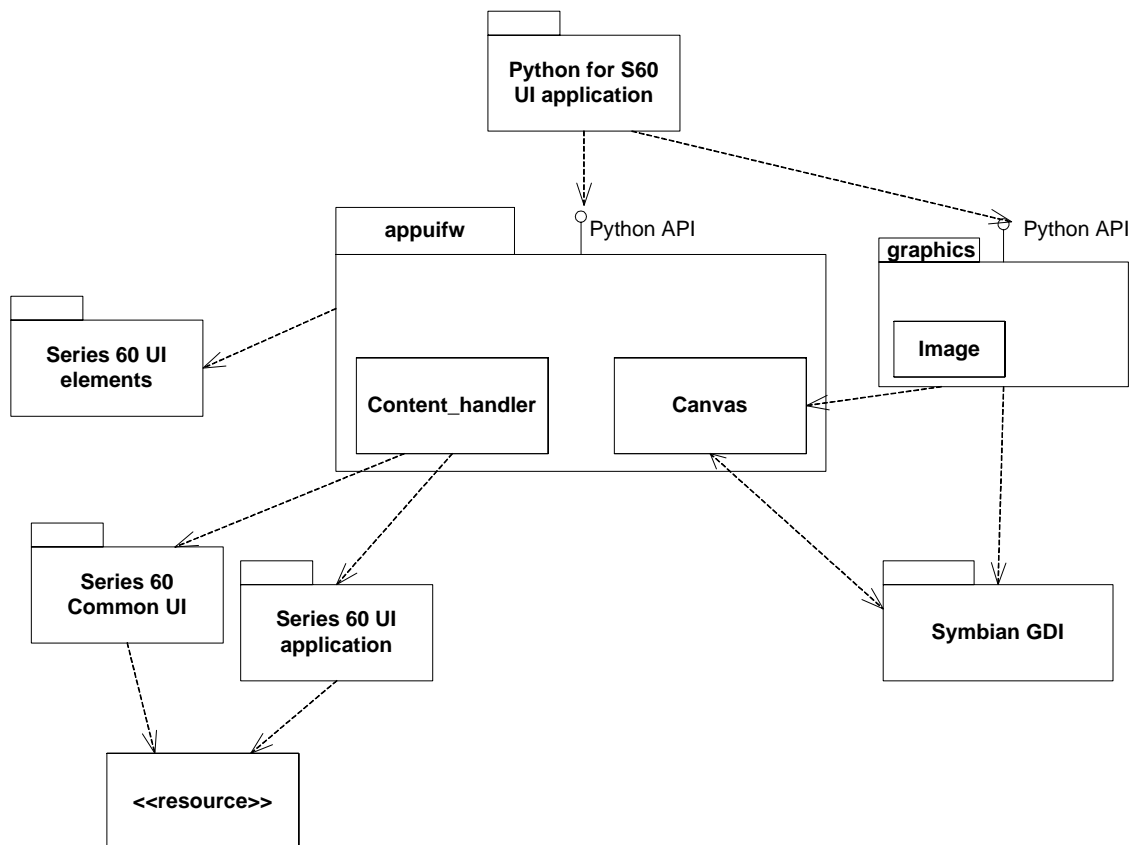


Figure 1: UI application programming

The `appuifw.Content_handler` object 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.

### 3.2 Basics of appuifw Module

Figure 2 shows the layout of a Series 60 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 4.

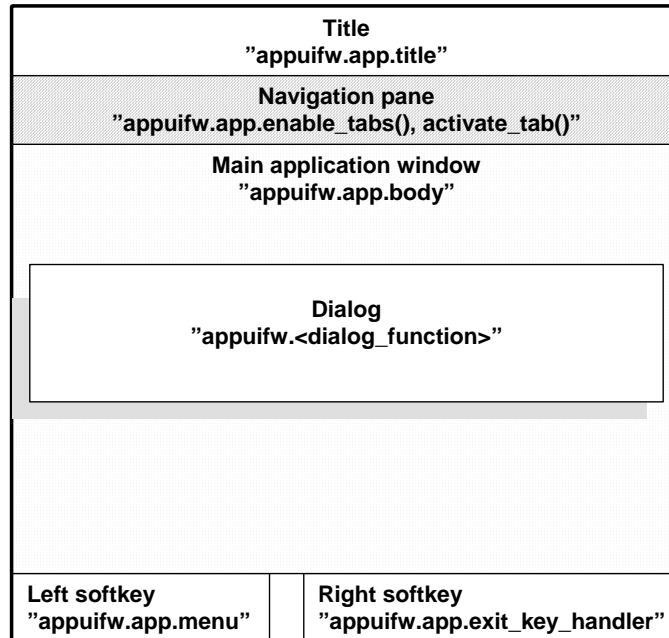


Figure 2: UI layout when `appuifw.app.screen` is set to 'normal'

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.

### 3.3 appuifw UI Controls

The 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 created and set to the `body` field (`app.body`) of the current application UI.

`Form` is a versatile dialog implemented as a type.

The following dialogs are implemented as functions:

- `note`
- `query`
- `multi_query`
- `selection_list`

- `multi_selection_list`
- `popup_menu`

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.

### 3.4 Softkeys

The softkeys are managed by the underlying Series 60 Platform. When no dialog is visible, the right softkey is bound to application exit and the left one represents an **Options** menu. Python for Series 60 offers an interface for manipulating the menu (see Section 5.2, *Application Type, menu*) and for binding the Exit key to a Python-callable object.

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`.

## 4 Selected Issues on Python Programming for Series 60

The following issues must be considered when using Python on Series 60.

### 4.1 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 Series 60 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.

### 4.2 Current Series 60 Python Script Execution Environment

The current options for installing Python scripts to a Series 60 device are: a stand-alone installation to the device's main application menu, and an installation to a folder hierarchy maintained by the Python execution environment. For more details on this topic, see *Programming with Python for Series 60 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 execution environment delivered with Python for Series 60 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 execution environment. 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

shared execution environment 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 from the current Python execution environment, see *Programming with Python for Series 60 Platform [6]*. These include the arrangements for standard output and the maintenance of the **Options** menu contents.

### 4.3 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.

### 4.4 Usage of Unicode

No changes have been made to the standard library modules with regard to string argument and return value types. Series 60 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.

### 4.5 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 Series 60 APIs described in this document. The float type is used for storing time values.

### 4.6 Sharing Native Resources between Threads

**Warning:** Python for Series 60 objects that wrap native resources cannot be shared between threads. Trying this can lead to a crash.

In general, Python for Series 60 objects that wrap native resources cannot be shared between threads. Trying this can lead to a crash. This is because native resources cannot be shared between native threads. Two examples of this follow:

- Symbian OS STDLIB implementation has some limitations that are reflected at OS module support (see *Series 60 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 Series 60 version of the `socket` module have the same restriction.

### 4.7 Scalable User Interface

**Note:** Series 60 2nd Edition FP3 and further releases.

Series 60 2nd Edition FP3 enables a new feature called scalable user interface. For Python developers scalable user interface 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 5.6, *Icon Type* of this document, as well as *Programming with Python for Series 60 Platform* [6].

#### 4.8 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.

#### 4.9 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.





















































































































