

# Python für Series 60

Sebastian König

12. März 2007

---

## Inhaltsverzeichnis

|          |   |          |
|----------|---|----------|
| <b>1</b> | <b>Einleitung</b>                           | <b>3</b> |
| 1.1      | Motivation . . . . .                        | 3        |
| 1.2      | Entwicklung für Symbian OS . . . . .        | 3        |
| <b>2</b> | <b>Überblick über Python für Series 60</b>  | <b>4</b> |
| 2.1      | Python . . . . .                            | 4        |
| 2.2      | Features . . . . .                          | 5        |
| <b>3</b> | <b>Entwicklung mit Python für Series 60</b> | <b>5</b> |
| 3.1      | Hello World . . . . .                       | 6        |
| 3.2      | Threading . . . . .                         | 6        |
| 3.3      | Nutzen S60 spezifischer Features . . . . .  | 7        |
| <b>4</b> | <b>Fazit</b>                                | <b>8</b> |

# 1 Einleitung

Python für Series 60 ist eine Skriptsprache für Symbian. Dieser Artikel beschreibt die Gründe, Vorteile und Eigenarten der Python Portierung.

## 1.1 Motivation

Symbian ist das am weitesten verbreitete Betriebssystem für Smartphones und wird von großen Herstellern wie Nokia, Sony Ericsson und Motorola lizenziert. Laut einer Analyse der Firma Canals<sup>1</sup> im dritten Quartal 2006 besitzt Symbian einen Marktanteil von 59,7%, gefolgt von Linux mit 22,0% und Windows Mobile mit 2,2%. Dabei schätzt Canals dass über 90% heutiger Symbian Geräte Series 60 Modelle sind.

Die meisten Applikationen für Symbian werden entweder in Symbian C++ oder in JavaME entwickelt. Dabei hat Symbian C++ den Vorteil, dass die Anwendung nativ auf der Symbian Plattform läuft und damit alle Features von Symbian ausnutzen kann. JavaME beschränkt den Programmierer auf die angebotene API. Direkte Zugriffe auf das System aus der Virtual Machine hinaus sind nicht möglich. Symbian C++ benötigt jedoch viel Einarbeitungszeit und lange Entwicklungszeiten im Vergleich zu JavaME.

Aus diesem Grund hat Nokia die Skriptsprache Python für ihr Series 60 System portiert. Python ermöglicht als offene High-Level Skriptsprache kürzere Entwicklungszeiten und bietet im Gegensatz zu JavaME die Möglichkeit durch Module neue Funktionalitäten hinzuzufügen, die zuvor nicht über die Standard API gegeben waren.

## 1.2 Entwicklung für Symbian OS

Der Einstieg in die Symbian C++ Programmierung stellt den Entwickler vor einer Reihe neuer Programmier Paradigmen. Diese Paradigmen bezeichnet Symbian als „Essential Idioms“. In diesem Abschnitt werden wir drei solcher Idiome betrachten um ein Gefühl dafür zu bekommen, warum der Einstieg und die Entwicklung in Symbian C++ so mühsam ist.

**Klassen** Symbian C++ unterscheidet zwischen 4 Arten: Value, Heap, Resource und Proxy Klassen. Die Konvention sieht vor, dass Klassennamen je nach Typ mit einem speziellen Zeichen beginnen. Value Klassen beginnen mit einem *T* und haben die Besonderheit, dass sie keine externen Objekte beinhalten. Das bedeutet sie haben keinen Dekonstruktor, weil sie keine Ressourcen besitzen die extra aufgeräumt werden müssen. Dafür gibt es Heap-Allokierte Klassen, diese beginnen mit einem *C* und können nur auf dem Heap erzeugt werden. Diese Klassen müssen nach dem „Two-Phase Construction“ Prinzip erstellt werden, dabei darf im

---

<sup>1</sup>canals.com ltd

Konstruktor nichts geschehen, was zu einer Ausnahme führen könnte. Der dritte Typ stellen die Resource Klassen (*R*) Klassen dar. Dies sind Proxy Klassen für Objekte die in anderen Threads leben und dienen dazu Requests an andere Objekte weiterzuleiten. R Klassen haben keinen expliziten Konstruktor oder Dekonstruktor. Der letzte Klassentyp sind die Interface Klassen (*M*). Diese Klassen stellen Interfaces dar und besitzen meistens nur reine virtuelle Methoden.

**Deskriptoren** Speicherverwaltung findet in Symbian über Deskriptoren statt. Die wichtigsten Deskriptoren sind Pointer, Buffer und Heap Deskriptoren. Pointer Deskriptoren (*TPtrC*) verweisen auf eine Speicheradresse auf dem Stack und kennen im Gegensatz zu einem `char*` Pointer in C die Größe des dort vorhandenen Speicherbereichs. Buffer Deskriptoren (*TBufC*) vergleicht Symbian mit `char[]` Arrays in C. Der Deskriptor verweist nicht auf eine andere Adresse. Die Daten liegen direkt an der Adresse des Buffer Deskriptors. Heap Deskriptoren (*HBufC*) werden benötigt wenn die Größe des Buffers erst zur Laufzeit bekannt ist, sie kapseln also einen dynamisch reservierten Speicherbereich. Nicht nur der reservierte Speicherbereich, sondern auch der Deskriptor selbst liegt auf dem Heap und wird mit einem einfachen Pointer (*HBufC\**) referenziert.

**Threading** Im Gegensatz zum gewohnten Multitasking auf Desktop Systemen geschieht Threading in Symbian nicht preemptiv, sondern kooperativ über so genannte *Active Objects*. Ein *Active Object* stellt einen kooperativen Task dar, der von einem *Active Scheduler* verwaltet wird. Dieses System kommt zum Beispiel beim UI Event Handling zum Einsatz. Ein blockierender Thread verhindert dadurch das Auslösen von Event Callbacks.

Dieser Einblick in die Symbian Idiome soll verdeutlichen welche Anforderungen Symbian an die Python Portierung stellt und warum das daraus resultierende PyS60 nicht identisch mit Python für Desktop Systeme ist.

## 2 Überblick über Python für Series 60

Python für Series 60 ist ein Projekt von Nokia mit dem Ziel die Entwicklung von Software für Series 60 einfacher zu gestalten und damit mehr Programmierer für eigene Smartphones zu gewinnen.

### 2.1 Python

Python ist eine objektorientierte, funktionale Skriptsprache und wird seit 1990 als Open Source Projekt entwickelt. Die Popularität von Python ist besonders in den letzten Jahren gestiegen. So wird Python zum Beispiel von Google, NASA oder Industrial Light and Magic verwendet. Zu

den Stärken der Skriptsprache zählt die Einfachheit der Syntax. Python ist plattformunabhängig und lässt sich sehr einfach durch Module in C/C++ erweitern. Dies ist insbesondere im direkten Vergleich mit JavaME ein großer Vorteil und gibt dem Entwickler mehr Freiraum.

## 2.2 Features

PyS60 bietet zwei Arten nativer C++ Module für die S60 Plattform: eingebaute und dynamisch ladbare Module. Es gibt zwei eingebaute Module, diese werden automatisch mit dem Start des Interpreters geladen.

| Modul   | Beschreibung   |
|---------|--|
| e32     | implementiert Interfaces für spezielle Symbian OS Dienste, die über die Standard Python Bibliothek nicht verfügbar sind. |
| appuifw | bietet ein UI Framework um native GUI Elemente in Python verwenden zu können.  |

Die dynamisch ladbaren Module bieten APIs zu den weniger integralen Diensten der S60 Plattform.

| Modul     | Beschreibung   |
|-----------|--|
| graphics  | Zugriff auf Grafik Primitive und Bildbearbeitung               |
| camera    | Kamera Kontrolle: Bildgröße, Farbtiefe, Zoom, Weißabgleich ... |
| messaging | Versenden von SMS und MMS                                      |
| inbox     | Auslesen des Posteingang                                       |
| location  | Auslesen von Location Informationen                            |
| sysinfo   | Systeminformationen: Batteriestand, Arbeitsspeicher ...        |
| audio     | Aufnehmen und Abspielen von Audio.                             |
| telephone | Wählen von Rufnummern  |
| calendar  | Zugriff auf Symbian Agenda Datenbank                           |
| contacts  | Zugriff auf Adressbuch Dienste von Symbian                     |
| e32db     | Interface für native Symbian Datenbank                         |

Dies sind nur die Series 60 spezifischen Module. Zusätzlich bietet PyS60 die meisten Module der Standard Python Bibliothek.

## 3 Entwicklung mit Python für Series 60

Dieses Kapitel dient als Einstieg in die Programmierung mit PyS60 und weist dabei auf Unterschiede zum gewohnten Python für Desktop Systeme hin.

### 3.1 Hello World

In diesem Abschnitt wollen wir ein einfaches Hello World Skript mit Hilfe von UI Elementen schreiben. Dafür verwenden wir das Modul *appuifw*.

```
1 import appuifw
2 appuifw.note(u"Hello World", "info")
```

Die Methode *note* dient zur Ausgabe von Informationen in Form eines Pop-Ups. Dabei nutzen wir den Typ „info“, wodurch ein *i* als Icon im Pop-Up erscheint. Den auszugebenden String müssen wir als Unicode übergeben. Generell erlauben alle API Methoden Unicode und ASCII. Strings, die auf dem Display dargestellt werden sollen, müssen jedoch zur korrekten Darstellung in Unicode formatiert sein. Als Rückgabeformat für Strings verwenden alle API Funktionen ebenfalls Unicode.

Eine Benutzereingabe lässt sich ähnlich leicht erstellen:

```
1 import appuifw
2 time = appuifw.query(u"Type a time:", "time")
```

Analog zur *notify* Methode erwartet *query* einen Typ. In unserem Beispiel ist dies „time“. Python stellt automatisch eine passende native Eingabemaske dar. Mögliche Typen für ein Query sind: *text*, *code*, *number*, *date*, *time*, *query* und *float*. Der Rückgabewert des Query „time“ sind die Sekunden seit Mitternacht.

### 3.2 Threading

Wie schon in Sektion 1.2 erwähnt wurde, verwendet Symbian OS *Active Objects* um kooperatives Multitasking zu verwalten. Um in Python diese Active Objects verwenden zu können gibt es das *e32* Modul. Als Beispiel für kooperatives Multitasking in PyS60 wollen wir ein kleines Skript schreiben, welches Active Objects verwendet um auf Event Callbacks des UI reagieren zu können.

```
1 import camera
2 import appuifw
3 import e32
4
5 def my_exit_handler():
6     lock.signal()
7
8 def say_hi():
9     appuifw.note(u"Hi!") #Active Scheduler arbeitet
10
11 appuifw.app.menu=[(u"Say Hi!", say_hi)]
12 appuifw.app.exit_key_handler=my_exit_handler
```

```
13 lock = e32.Ao_lock()
14 img = camera.take_photo() #AS arbeitet bis das Foto fertig ist
15 lock.wait() #AS arbeitet bis lock.signal() aufgerufen wurde
```

Dieses Skript erzeugt ein Foto und legt sich anschließend schlafen. Über ein Menü können wir ein Pop-Up mit dem Text „Hi“ öffnen. Das Programm gibt an drei Stellen die Kontrolle an den *Scheduler* ab. Die API Funktionen in Zeile 9 und 14 machen dies implizit. Der Scheduler arbeitet andere Active Objects ab während das Pop-Up zu sehen ist und das Foto von der Kamera geschossen wird. In der letzten Zeile legen wir unser Active Object schlafen, wir geben also explizit die Kontrolle zurück an den Scheduler. Das Skript reagiert in diesem Zustand auf Event-Callbacks, die Funktion `my_exit_handler` kann also jederzeit aufgerufen und das Programm beendet werden. Würden wir anstatt eines `e32.Ao_lock` ein `thread.lock` aus dem Python thread Modul verwenden, so wäre der Callback nicht möglich gewesen, weil `thread.lock` den gesamten Systemthread blockiert.

Das UI wird automatisch beim Start des Skripts erzeugt. Weil die UI Methoden nicht Thread sicher sind darf das `appuifw` Modul nur im Haupt Thread verwendet werden. Dies gilt für alle nativen Ressourcen wie beispielsweise Sockets oder Dateideskriptoren. Solche Ressourcen dürfen nur in dem Thread verwendet werden, in dem sie erzeugt wurden.

### 3.3 Nutzen S60 spezifischer Features

Der Vorteil von PyS60 gegenüber JavaME ist die Verfügbarkeit von S60 spezifischen Features. Als Beispiel wollen wir ein Skript schreiben, welches diese S60 spezifischen Features benutzt. Das Skript soll sich in den Posteingang von Symbian einhängen und eingehende SMS direkt durch Sprachsynthese vorlesen.

Für das Einhängen in den Posteingang und für die Sprachausgabe benötigen wir entsprechend zwei Module

```
1 import inbox
2 import audio
```

Wir lassen uns den Posteingang von Symbian als Objekt geben

```
1 i = inbox.Inbox()
```

Um auf eingehende Nachrichten zu reagieren müssen wir eine Funktion erstellen, die als Event Handler dient.

```
1 def read_sms(id):
2     text = i.content(id)
3     audio.say(text)
```

Die Funktion holt sich anhand der ID der eingegangenen Nachricht aus dem Posteingang den Inhalt und gibt diesen über das Audio Modul aus.

Als letzten Schritt müssen wir unsere neue Funktion als Event Handler beim Posteingang registrieren.

```
1 i.bind(read_sms)
```

Damit ist das Skript fertig. Das gleiche Programm wäre in Java nicht möglich, weil wir innerhalb der Virtual Machine keine Möglichkeit haben an eine Sprachsynthese zu kommen, obwohl diese vom darunterliegenden System wie Symbian angeboten wird.

## 4 Fazit

Die erste Version von Python für Series 60 wurde vor über 2 Jahren von Nokia veröffentlicht. In der Zeit seit dem ersten Release wurde die PyS60 API stark erweitert.

Bislang gibt es jedoch nur wenige Python Applikationen für Series 60. Die meisten Python Applikationen im Internet sind Prototypen oder Proof of Concepts um die Mächtigkeit und Flexibilität von Python darzustellen. Der Mangel an Applikationen kann in erster Linie an dem Mangel potentieller Benutzer liegen. Python für Series 60 ist nicht sehr bekannt und kaum ein Series 60 Besitzer hat einen Python Interpreter installiert. Ein weiteres Problem ist der Unterschied an Funktionalität zwischen einzelnen Python Versionen. So werden mit jedem neuen Minor Release Funktionen hinzugefügt oder geändert. Hinzu kommt, dass die Verfügbarkeit einzelner API Funktionen stark von der Series 60 Edition und dessen Feature Pack abhängig ist. In der PyS60 API Dokumentation sind solche Feature Pack Abhängigkeiten nur mangelhaft dokumentiert.

## Literatur

- [1] Nokia, *PyS60 Library Reference*. 1.3.15 final, 01.12.2006  
<http://sourceforge.net/projects/pys60>
- [2] Jukka Laurila, *Python On A Phone*. 27.06.2005.  
<http://research.nokia.com/research/projects/python-for-s60/PythonOnAPhone.ppt>
- [3] Symbian, *Essential Idioms*. 2002.  
[http://www.symbian.com/developer/techlib/v70sdocs/doc\\_source/DevGuides/EssentialIdioms/index.html](http://www.symbian.com/developer/techlib/v70sdocs/doc_source/DevGuides/EssentialIdioms/index.html)
- [4] *Python for S60 Discussion Boards*.  
<http://discussion.forum.nokia.com/forum/forumdisplay.php?forumid=102>