

Runtime Environments – the Phenomenon

Roy Ben Hayun

Published by the Symbian Developer Network

Version: 1.0 – July 2007

1 INTRODUCTION.....	2
2 ASKING THE RIGHT QUESTIONS.....	2
3 ENUMERATING THE CANDIDATE RUNTIME ENVIRONMENTS.....	3
3.1 FLASH LITE	3
3.2 JAVAME	5
3.3 RUBY.....	8
3.4 .NET COMPACT FRAMEWORK.....	10
3.5 PYTHON.....	11
3.6 OPL	13
3.7 M.....	14
4 SUMMARY.....	16
NEXT TIME.....	16
LINKS TO RESOURCES	16

1 Introduction

Welcome to the third instalment of Mobile Runtime Environments, the column that gives developers, technical leaders and technical decision-makers an insight into the various development platforms available for Symbian OS.

A hosted Runtime Environment can be defined as an execution platform that is hosted by, and receives services from, the underlying native Operating System. Well known examples of mobile Runtime Environments are Java, Flash, Ruby and Python amongst others.

The focus of this month's article is the Runtime Environments phenomenon– what are the Runtime Environments available for Symbian OS and what can they do for us?

To answer this, we first collect a number of key questions for each of the Runtime Environments, and then we investigate each of the Runtime Environments using these questions.

It is recommended that you read the first two articles in the series to learn about the business requirements of Runtime Environments and the theoretical aspects of the Runtime Environment concept.

2 Asking the right questions

The reason that we start with the questions and not with finding available Runtime Environments is that asking the right questions is probably a better way of assessing the suitability of the Runtime Environment than looking at each Runtime Environment on its own; and it gives us a basis for comparison between the Runtime Environments.

Let us use the concepts from the previous articles and find what concrete topics we would like to investigate further with each of the Runtime Environments. Assume that we are about to interview a line of Runtime Environments as candidates for our next project, and these are the questions we would ask them in the interview.

2.1 Introduction - Please tell us about yourself...

We don't want to be too schematic, right? So like most interviews, let us first ask the Runtime Environment to tell us about itself and give some general information; provide a general description of the Runtime Environment, How does it enhance the Symbian OS richness? How does it go with trends? Give some quantitative information e.g. estimated size of developer community, etc.

2.2 Roots - Tell us about your childhood and family..

A little nostalgia before we move to more technical issues - How did the computing language and the Runtime Environment evolve?

And some family ties - How close are the mobile and desktop variants of the Runtime Environment, if that is applicable?

After that, we are ready to start drilling down to specific topics that are of interest to us.

2.3 Libraries support for various technologies

As discussed in the second article of the series – the language delivers the computations capability of the Runtime but beyond that it is the libraries that determine the power of the Runtime

Environment more than anything. If you want to connect over Bluetooth, display UI, use persistent data; this is done through the libraries.

What is the Runtime Environment support for UI, multimedia, connectivity, messaging, personal information, persistence and so on?

2.4 Purposes - What is this Runtime Environment fit for & what it is not?

Which purposes is this Runtime Environment ideal for? What is its main strength? And since we discuss what is it good for, let us also discuss what we should not try and do with it - What is this Runtime Environment not suited for? What is its main weakness? What would we add to it, if we could?

2.5 Integration with the underlying system

Here we would answer several questions such as, is it possible to execute native code (i.e. call native functions from a DLL)? How does the user awareness and experience compare to native applications (i.e. installation, launching etc) and what is the relation to Symbian OS v9 Platform Security and Symbian Signed (i.e. the scripts need to be signed with the right capabilities etc)?

2.6 Developer productivity

This is a simple yet key question - How does the Runtime Environment speed Time To Market (TTM) and developers productivity? Which tools are used for development (i.e. IDE)? How does the language design promote productivity?

3 Enumerating the candidate Runtime Environments

Let us enumerate the various Runtime Environments that have been picked up for this article. In recent years, as mobile programming goes mainstream, we see how hosted Runtime Environments are becoming popular for applications development and in this article we will leave Symbian C++, C standard library and the recently announced P.I.P.S. out of the scope. But we will get back to those native development options for Symbian OS in a later article.

So looking around the Symbian ecosystem, we can spot some leading mass-market Runtime Environments, some others who are mass-market wanna-be's and some smaller and less popular Runtime Environments which are interesting nonetheless. These are Flash, JavaME, Ruby, .NET, Python, OPL and m (yes, there is such Runtime Environment).

This will be our line of candidates and they will answer each of the questions from the previous section (The order in which the Runtime Environments are mentioned, is only random).

Now let the fun begin!

3.1 Flash Lite

3.1.1 Introduction

Flash Lite is the version of the Flash player from Adobe that has been specifically designed to run on mobile devices. It is a compelling Runtime Environment that increases the range and quality of rich media content experiences that have a consistent display across a range of platforms and can be delivered to Symbian OS-based mobile phones.

The first (naive) question is probably.. "But is this a Runtime at all?" and the answer is "Yes, it is!" and it is a very good one also. It has language support for SVG-Tiny and ActionScript 2.0, has reasonably good libraries support but it is not a fully featured Runtime Environment and it is tightly sandboxed. Its role is to be a Runtime Environment for a defined set of applications and it fulfils this specific role excellently.

It is undoubtedly a very successful Runtime Environment with a vibrant developer community that is estimated at nearly 2 million worldwide and is becoming more interested in developing mobile applications. Flash is installed on 700 million PCs and mobiles and there is a lot of existing Flash content that could be ported to mobile platforms. All those numbers continue to grow rapidly.

Bringing all this rich content, all those developers and talent to the Symbian ecosystem is a huge achievement and makes Flash a valuable member of the Symbian ecosystem.

Flash is a powerful enabler of web2.0 rich content and has a consistent and positive trend in development of new technology features. Recently Adobe announced the Tamarin open source project for a high-performance, open source implementation of the ECMAScript language specification, which will continue to be used by Adobe as part of the ActionScript VM and opens the way for more innovations to flow into Flash technology.

For more information see www.adobe.com/products/flashlite/

3.1.2 Roots

The Flash Lite player uses ActionScript which is based on the ECMAScript standard. Early versions of the Flash Lite player used the Flash Player 4.0 version of ActionScript but recent releases (2.0 and above), use ActionScript 2.0 which allows for object orientated development and a much richer feature set to allow for better program design.

The desktop version of the Flash Lite player is currently based on ActionScript 2.0 (ActionScript 3.0 will be released in 2007). The Flash Lite player does not include all of the features of the desktop version but it does allow access to native device properties e.g. battery level, signal strength. The current version of Flash Lite does not support FLV files but this looks set to change with the release of Flash Lite version 3.0 later in 2007.

As the Flash Lite player matures and the number of Flash enabled devices increases, it is expected that more and more of this developer community will make the move into mobile devices.

3.1.3 Libraries support

Flash Lite is known for its state of the art support for 2D graphics (but not 3D) which has a consistent display across a range of platforms.

It has limited support for multimedia audio and video (but no camera access, for example).

The Flash Lite player can dynamically load data, images, sound and video using HTTP from a remote server but local connectivity is not possible with any version of the Flash Lite Player.

Flash Lite can pass data to native messaging applications for making phone calls, send SMS & MMS messages.

Due to the strict security limitations of the Flash Lite player, there is no access to personal information and native applications like the calendar or contacts.

Application data can be persisted locally to the file system and be retrieved at a later date.

3.1.4 Purposes

Flash Lite provides a very rich visual experience for end users so it's ideal for creating casual games, screen savers and wallpapers. This fact has also led to some handset manufacturers using

Flash Lite as the UI for the entire phone. Another advantage of using Flash Lite is the vector based and easy to port applications to different screen sizes. The ease of development also makes Flash Lite suitable for rapid prototyping.

Flash Lite can't integrate with device hardware like Bluetooth or the camera. There can be performance issues with the runtime so lower spec phones aren't suitable for running Flash Lite. Access to device events like incoming calls or SMS would allow you to create some interesting applications. Adding support for Flash Remoting would allow you to create more efficient dynamic data applications.

Thousands of pieces of content have been created for mobiles specifically. Additionally, millions of pieces originally created for the web can be repurposed for mobile phones.

3.1.5 Integration with the underlying system

The Flash Lite player runs in a security sandbox like the desktop version so it cannot execute native code.

Flash Lite applications is packaged as a standard Symbian installer file (.SIS) and can be self signed, developer signed or Symbian signed. This gives users the ability to install Flash Lite applications in the same way as a native application. An icon will be created in the menu system and the application can be uninstalled in the same way

3.1.6 Developer productivity

Flash Lite provides an easy way to create mobile content for less experienced developers. Its high level, straight forward language design makes it much simpler than other runtimes like JavaME so TTM can be drastically reduced. The ease of porting applications to a range of devices also increases developer productivity and frees more development time that can be invested in development of the applications features enhancements and more applications.

Adobe has recently announced Device Central which is an application and device community to help Flash Lite developers develop mobile applications. It provides better development workflow and a database of device profiles.

The development tools are the Flash IDE with mobile emulator, which tests device specific settings e.g. battery and signal, and Device Central.

3.2 JavaME

3.2.1 Introduction

Mobile Java is one of the most widely installed runtimes on mobile phones. By enabling rich, advanced data services (for instance, location-based services and wireless commerce) it has gained recognition as a solid revenue stream for operators and content providers.

There are over a billion Java-enabled handsets deployed across all major networks and more than 5 million Java developers world wide, which make Java the de-facto industry standard in software development and as such appropriate for "mass market" services and applications.

The Symbian OS MIDP Runtime Environment is an integrated component of the OS, incorporating a high-performance CLDC virtual machine and a bespoke MIDP implementation designed to allow Java applications to exploit the advanced multitasking, graphics, and comms capabilities of the underlying OS. The Symbian OS MIDP/CLDC stack includes similarly highly integrated implementations of many JavaME API packages, including Bluetooth, 3D graphics, multimedia and content handling.

Symbian OS JavaME stack exposes the richness of the OS through standard Java APIs and brings to the Symbian platform a richness of content, applications and IP.

3.2.2 Roots

Tracking evolution trees of computing languages is always an interesting archaeological exercise. In the case of Java, its syntax which is similar to C++ is a red herring and it's hard to say that Java was based on C++ (or even Modula-3). For example, Java is dynamically linked; its OOP model is using single inheritance, class objects, and an extensive runtime system. In fact, Java is more of a offshoot of Smalltalk-derived languages like Objective-C which is an object-oriented mutant of C with single inheritance, dynamic loading, class objects and interfaces. Java also borrowed many features directly from SmallTalk and from Simula68 since each of the Java language designers came with his own influence and philosophy of computing languages.

Java still goes through metamorphosis once in a while and keeps evolving. Naturally, the desktop edition, JavaSE, is the Java platform which leads those changes and from the recent 1.5 release there are quite a few changes that take the standard desktop edition further away from the mobile edition (i.e. Generics, enhanced for Loop, Autoboxing/Unboxing, typesafe Enums, static imports).

The mobile platform is different from the other two main Java platforms, JavaSE and JavaEE, to match the different requirements in the mobile environment. JavaME is subdivided by Configurations, Profiles and optional packages, contains a more limited set of the standard APIs and includes its own special APIs and frameworks.

However, a JavaSE developer would find himself in a very familiar environment and be able to become highly productive within a short period of adjusting and learning. The skills and knowledge required to move to JavaME from other Java platforms are easily transferable.

3.2.3 Libraries support for various technologies

JavaME MIDP has a highly powerful, rich set of APIs for UI, graphics, multimedia, connectivity, messaging, persistence, personal information, device managements and much more. But this is where we will not spend much time since there is a wealth of information on this topic and only discuss the Java Community Process and the leading standards for JavaME.

The Java technology evolution process is an interesting type of open, yet managed, collaboration between the Java Community Process (JCP) participants (over 700 corporate and individual participants) and Sun the other Executive Committee (EC) members that guide the expert groups and represent the major stakeholders and other members of the Java Community.

The JSR's are openly developed and revised by the open participative JCP process which since its introduction in 1998 has fostered the evolution of the Java platform in cooperation with the international Java developer community.

The current leading JavaME standard is the MIDP2.0 which includes a rich set of JSR's that are usually implemented in Symbian as interfaces to the native OS services to improve performance and footprint. MIDP application can make use of a large number of optional JSR's which are provided by Symbian MIDP implementation.

The next major step in the JavaME roadmap is the MSA specification which will raise the bar significantly.

3.2.4 Purposes

Currently, JavaME is ideal for applications programming and is not suitable yet for system development.

Java offers highly powerful APIs with a reasonably good fine grained control over details and fine tuning features, which enables a large diversity of JavaME applications for finance, commerce, consumer, enterprise, messaging, location based services and much more (oh yes, games too :-)

However, Any hosted Runtime Environment APIs can not be as powerful as the native Symbian OS APIs and provide the fine grained control or every feature that is provided by the native C++ APIs. Also, system development which extends the OS and requires tight integration is to be accomplished primarily by native languages e.g. C and C++.

The developer needs to be aware of the limitation of the JavaME platform which is fragmented by configurations, profiles and optional APIs. MIDP applications can not execute out of the sandbox and are restricted to MIDP APIs only.

The power of the language and the vast range of libraries show the potential power of a hosted Runtime Environment and explain the success of Java technology which is a model for business and technology development. Because the power and potential of the Java platform is clearly seen, the list of demands from the language and the platform is long and extensive. i.e. developers would surely be happy to use blocks, metaclassing and get the newest JavaSE features to the language; there is always more room for excellent Java APIs from other Java platforms or specifically for mobiles.

The list is long not because the platform is limited; MIDP and MSA have a rich set of API's; but because how Java is perceived as a powerful, dynamic and evolving Runtime Environment that can be stretched further to be a platform for an increasing number of purposes and enables a mass market business penetration.

3.2.5 Integration with the underlying system

The ability to execute native code depends on the JavaME configuration.

CLDC is targeted for limited resources devices therefore applications are sandboxed and can not execute native code outside of the VM. The CDC configuration is targeted for higher end devices and enables us to add native calls through the standard Java Native Interface (JNI) mechanism.

As far as possible Symbian OS treats Java applications as first class citizens (i.e. identical to native applications). For instance the installation process is similar so the user installs, launches, and removes MIDlets in the same way as native applications.

MIDlets also use native UI components (rather than UI components written in Java), improving performance, reducing footprint, and giving more of a native look and feel. The prehistoric byte-code interpreter has long time ago been enhanced with an aggressive compiler that produces platform dependant code for speed optimisation.

The Symbian OS v9 platform security model does not apply for CLDC/MIDP applications, which need to be signed in the standard process (For more information please refer to the Sun JavaVerified program). CDC applications are packaged as a standard .SIS file, which can be assigned with Platform Security capabilities and needs to go through the usual Symbian Signed process.

3.2.6 Developer productivity

Java development is characterised as being rapid and robust. The simple and robust language design protects the code integrity and the wealth of advanced SWE tools available for Java ensures developers can use state of the art tools for almost any development task.

Currently the leading IDE is probably Eclipse, followed by NetBeans, JBuilder and others. Java tools excel not only in IDEs and debugger but provide a huge number of software engineering tools for Java development for all Java and native platforms.

3.3 Ruby

3.3.1 Introduction

Ruby is an open source, interpreted, dynamic, reflective, and pure object-oriented programming language with syntax similar to Python and Perl and has Smalltalk-like OO support.

One of the wide usages of Ruby is Ruby on Rails which is a successful web development framework that allows rapid building of web pages.

JRuby is an open source Java implementation of the Ruby interpreter, tightly integrated with the Java platform to allow embedding in other Java applications. A primary powerful feature of JRuby is two way access between Java and Ruby which gives the ability to use the services of the Java platform from Ruby and use the power Ruby from Java.

At the time of writing this article, Ruby is in a different status than all the other mobile Runtime Environments that are discussed. Ruby for Symbian OS was released on September 2006 and is still in an embryonic state so it is difficult to give an evaluation of Ruby as a viable mobile Runtime Environment yet. The native Ruby port has been done on WINSCW and ARMv5 for S60 3rd edition N73, N80 and UIQ3.1 P990i and M600.

It's too early to estimate if Ruby will become a viable mobile Runtime Environment at all, but there is a very strong positive trend in the popularity of Ruby - the developer community is growing rapidly and more awareness is created. The potential is there and the choice will be made by the Ruby developer community itself.

If Ruby does become mainstream then Ruby for Symbian OS will provide an easy entry for developers wishing to develop for mobiles or reuse existing code for mobile applications.

Symbian OS port for Ruby MRI C interpreter and a newer port of JRuby for Symbian OS CDC mobile devices will be demonstrated in JavaOne 2007.

For more information on Symbian's presentation in JavaOne please refer to www.developer.symbian.com.

3.3.2 Roots

Ruby was first released in 1995 by Matz (Yukihiro Matzimoto). It became more popular than Python in Japan by 2000 and almost as popular as Perl by 2004. Today it has the strongest trend upwards in the TIOBE index. There is something about Ruby that makes it the most hyped programming language these days.

Ruby is an interpreted, dynamically typed, pure OO language which supports other programming paradigms such as functional programming and can masquerade as a procedural language (by silently adding global methods to the base Object class). Ruby's OO purity provides a number of features that Python or Java lack like meta-classing and lets developers change, add or remove a method for any class at a later point after it was declared (The changes will be present in any new objects that you create and even in existing objects of that class).

Rather than being a limited scripting language, Ruby is a full-fledged programming language which makes it suitable for heavy duty tasks. It is much more complex than Python and its features, hang together well. The design philosophy behind Ruby give developers lots of ways to implement the same task, to be imaginative, creative and enjoy the coding.

Ruby's syntax and design philosophy are heavily influenced by Perl, it includes built-in regular expressions but the elegant and lightweight syntax makes the code easily readable, succinct and concise.

3.3.3 Libraries support for various technologies

To make it clear, Ruby is not a viable mobile Runtime Environment yet. A full fledged Runtime Environment would contain, at a minimum, a reasonable number of mobile platform APIs and have a mobile deployment model. Currently the Ruby for Symbian OS includes a port of the Ruby MRI C interpreter and a Proof Of Concept module, which contains reference code, demonstrates how to do basic rendering and play audio files and purposed to show how to extend the runtime for developers who wish to use Ruby on Symbian OS.

A viable mobile Runtime Environment would require Ruby interfacing to Symbian OS services using hybrid C and C++ but also a very long wish-list that contains instrumentation and tool support for on-target development and debugging, well specified Ruby mobile frameworks, APIs, GUI and optimisations for mobile platforms and much more.

The first step in the current state would have to be a limited UI support to enable interaction with users.

It is reasonable to assume that with some progress on the project, Ruby libraries support would have similar scope and features to Python but not as powerful as Java.

3.3.4 Purposes

With Ruby it is easy to sketch a mesh of scripts that do the trick, but instead, it is more likely that developers would produce a well thought solution that is easy to understand, maintain, extend and reuse.

It is a fun general purpose language for non system level programming, which is mainly programmer efficient as opposed to program efficient.

The general case for Ruby is shown by Ruby on Rails which is becoming a popular framework for agile web development and demonstrates that Ruby can be used for heavy duty jobs and for serious applications.

On mobiles, Ruby would be applicable to end-user programming, applications and development tools. Since speed is not a major concern in Ruby, it is not expected to be used for time efficient usages or system development and certainly not for mass market in the near future.

3.3.5 Integration with the underlying system

The VM is open source and developers can add whatever extensions they wish.

Developers who add native extensions are required to add Platform Security capabilities to the Ruby launcher wrapper-application and sign it with their developer certificate. Currently, the Ruby launcher has no capabilities by default.

User awareness and experience is not applicable yet but it is expected that when the Runtime Environment matures, it will have a deployment model similar to Python in which a script is either installed as a .SIS file or interpreted interactively using short link connection e.g. infra red.

3.3.6 Developer productivity

Ruby is a high level language which simplifies tasks and enhances developer productivity. Computations are done easily and the code is easy to change and amend.

As a scripting language that does not have a compile-link-deploy build process, the development process is speeded up considerably and frees more of the developer time to focus on other tasks, and receive immediate feedback during programming.

This feature could also allow on-device programming using the device keyboard, IrDA (infrared) and mobile VNC like implementation.

3.4 .NET Compact Framework

3.4.1 Introduction

The .NET Compact Framework is a stripped-down version of Microsoft's full .NET Framework. It enables C# and Visual Basic .NET developers to create applications for mobile and embedded computing devices. Microsoft provides the .NET Compact Framework for devices running their Windows Mobile operating system. Red Five Labs (<http://www.redfivelabs.com>) has created an implementation of the Compact Framework for Symbian OS smartphones and the Red Five team is presently finalizing the v1 release of their product.

Enabling C# and Visual Basic .NET developers to easily create mobile applications introduces a very large number of developers and their applications into the mobile ecosystem. More applications mean a richer experience for phone users and more network traffic for operators/carriers.

There are at least 2.5 million C# and Visual Basic .NET developers worldwide. Although it is not clear what fraction of those is actively developing mobile applications, such a large pool of developers could help to snowball mobile application innovation around Symbian devices as well.

The Red Five implementation does not make use of Microsoft's implementation. It was designed and implemented from the .NET ISO/ECMA standard and from open and publicly available documentation about the Compact Framework APIs.

3.4.2 Roots

The Compact Framework is an order of magnitude smaller than the full .NET framework. APIs that have no place on a mobile or embedded device were removed and those that were kept were trimmed. A number of APIs are unique to the Compact Framework, like IrDA (infrared) support.

Despite the differences, if care is taken to use only APIs included in the Compact Framework, then the same .NET binary will run on both a PC and a mobile device. Although it's very unlikely one would want to run the same graphical application on a PC and phone, this may be useful for non-graphical modules (DLLs) that are common to both PC and mobile applications.

3.4.3 Purposes

The Compact Framework enables the millions of C# and Visual Basic .NET developers to use exactly the same programming skills and development tools to create mobile applications. Often they can even reuse their existing code to target Symbian OS smartphones. The same application binaries that run on Windows Mobile smartphones can run unchanged on Symbian powered devices, unless the developer has sidestepped the .NET APIs and called directly into a native DLL – possible via the Platform Invoke facility.

The initial release of Red Five Labs' product will only support applications written for version 1 of the .NET Compact Framework, although Microsoft has already shipped version 2. This means applications that make use of v2-specific features cannot run on Symbian OS smartphones yet. Additionally, the Red Five Labs' release currently only supports S60; UIQ support is in active development.

3.4.4 Libraries support for various technologies

All the APIs available in the Microsoft .NET Compact Framework 1.0 are available on Symbian with the Red Five Labs product.

There is 2D drawing and a rich set of UI controls, so that applications built in the graphical Visual Studio designer run unchanged on Symbian phones, but adopt the native Symbian UI look and feel. 3D support is missing from Compact Framework 1.0, but present in 2.0.

Multi-media is not available through managed APIs yet, but can be accessed by calling native libraries.

There is an excellent networking support for remote connectivity and developers can use IrDA for local connectivity.

There is complete support for persistence and file system access, excellent XML, data and web services support. Other libraries whose support is not available through managed APIs, such as PIM, Multimedia, Bluetooth etc, can be accessed by calling native libraries.

3.4.5 Integration with the underlying system

The Red Five Labs' Compact Framework implementation is designed from the ground up for Symbian OS. The runtime itself is written in Symbian C++, with small pockets of native assembler, while the class libraries (which provide the .NET API) are written in C#.

The Platform Invoke (P/Invoke) feature of .NET allows C# or Visual Basic code to call functions in native libraries. In the case of the Red Five Labs product, and S60 3rd Edition security capabilities permitting, the entire native Symbian C++ API is available to .NET applications. This is very useful where the .NET APIs don't provide functionality required by the application but such functionality is available on the device. The downside is that applications that use P/Invoke can require some porting between Windows Mobile and Symbian OS. (With care, and by using wrappers and platform detection, it is still possible to create an application that uses P/Invoke yet the same binary runs on Windows Mobile and Symbian OS.)

In Red Five Labs' Compact Framework, applications running on Symbian OS should be indistinguishable from native apps.

3.5 Python

3.5.1 Introduction

Python provides an alternative choice for developers to write applications for Symbian OS by opening up the platform to a large development community that is not familiar with the intricacies of Symbian C++ and the Symbian OS APIs. It has gained significant traction, in particular in universities, many of which have adopted it as a part of the curriculum.

PyS60 (<http://sourceforge.net/projects/pys60>) is a Python system for the S60 platform that runs on Symbian OS v8.1 and 9.1. It consists of a port of the Python open source scripting engine (version 2.2.2) to the S60 environment, as well as a set of high level custom API bindings that enable applications to access Symbian OS APIs such as telephony, phonebook, calendar, camera, audio, SMS messaging and so on.

There is a simple port of PyS60 to UIQ that has minimal GUI functionality. This was done independently by TietoEnator based on the open sourced PyS60 code. See <http://sourceforge.net/projects/pyuiq>. A fuller system for UIQ would require the GUI bindings to be rewritten.

The size of the developer community is hard to estimate, but figures for the total number of Python programmers in the world of around one million have been suggested. However, we should realize that Python as a language is exceptionally easy to learn, so it is very possible that people would be willing to learn Python just to program the phone.

3.5.2 Roots

Python was created by Guido van Rossum in 1991, designed to be a successor to the ABC programming language and capable of exception handling and of interfacing with the open source Amoeba operating system.

In terms of the core language functionality, PyS60 is very close to the mainstream Python. All core language features are supported. The differences are mainly that the mobile version is missing some native extension modules and not all Python library modules are shipped by default. Note that since the core language is highly compatible, many pure-Python library modules work unchanged on the mobile.

3.5.3 Purposes

Python allows a faster development cycle compared to C++ and even Java, and it allows a relatively inexperienced developer to create either simple or complex applications very quickly for various purposes. As such, it is suitable for rapid application development for prototyping, research or teaching use. PyS60's main strengths are very low starting threshold for programming, high development productivity and native extensibility.

The runtime is not suited for developing applications that would need portability to mobile platforms other than S60 or for applications that stretch the system to the limits of its memory and computational capacity (i.e., start-up of a large application can take a few seconds).

3.5.4 Libraries support for various technologies

When it comes to high level APIs, PyS60 has a broad set of libraries support. Although the support is intended to be very high level and is not on the scale of Java APIs' granularity, PyS60 can take pride in being a powerful runtime.

PyS60 has API bindings for the S60 Avkon UI widget set, called "appui fw", a simple 2D graphics API (wrapping of the Symbian GDI graphics), and access to OpenGL ES on platforms that support it.

It is possible to play any sound file supported by the platform, and to record sound. Real-time viewfinder support and photo-taking support is available. (e.g. taking a photo is one line of code: `photo=camera.take_photo()`)

Bluetooth RFCOMM serial connections and OBEX connections are supported, both as a client and a server. Socket communication is supported via TCP and UDP. SSL is supported as well. The Python standard library provides an HTTP implementation that runs on top of the socket API, and that runs on PyS60 as well.

SMS and MMS sending are supported, as is SMS inbox reading. A Python application can get a callback when an SMS arrives and run code in response. Contacts and calendar database access is also supported.

PyS60 applications have as full access to the file system as native applications.

3.5.5 Integration with the underlying system

In order to run Python applications, the Python runtime package must be installed in the device. The runtime package can be bundled in the application SIS file or installed separately.

On S60, the installation and launching of Python applications is transparent to the user and the process is the same as that for native or Java applications. Python applications are packaged into a SIS file. When installed, the SIS file creates its own icon in the phone's application menu and can be launched by clicking that icon just like any native application.

PyS60 relies on the Symbian Platform Security framework for security, and does not try to invent its own security framework. SIS-packaged Python applications contain a stub EXE that loads the Python DLL, which has been signed with certain capabilities. The capabilities of the stub EXE then define what capabilities the Python code in that application has.

PyS60 supports dynamically-loadable native extensions (.PYDs) that can extend the interpreter with new object types, functions, classes and so on. The native extensions are first class citizens: access to them is as convenient as the access to the Python standard library modules.

3.5.6 Developer productivity

Python reduces the total coding effort considerably. The writing and maintaining of an application in Python can be accomplished much faster than using C++ or even Java.

Two main components to this are the language and the higher abstraction level APIs.

Python is an example of a high level language that has been designed from the start to be as friendly to the programmer as possible.

In the PyS60 development work, a central idea has been to expose the platform functionality to the programmer at the highest level that makes sense. For example, sending an SMS is one line of code in PyS60.

Other Python features such as its inherent simplicity, good design, dynamic typing, automatic garbage collection, eliminating the need for compilation and more, contribute to productivity further.

There is currently no IDE solution that would directly integrate with Python for S60 specifically, but the programmer can use any of the generic Python IDEs that are available. For development and testing purposes, there is a special application called the "script shell" that can run arbitrary plaintext Python scripts from the memory card or phone memory.

3.6 OPL

3.6.1 Introduction

OPL probably will not be the next big thing in development for Symbian OS phones. However, it played a big role in our history and won its place in this article because of its place in the hearts of many Psion and Symbian users.

Twenty years ago, OPL was originally the acronym for 'Organiser Programming Language', as it ran on the Psion Organiser hand-held devices. To reflect the new open status of the language, it was subsequently renamed as 'Open Programming Language'.

OPL is now an open source project, available for major Symbian OS platforms.

There are also lots of freeware applications, games and utilities written in OPL, which are available to help developers get more out of their devices. The open source project is still maintained by some enthusiastic users; currently about 20 programmers for the Communicator/UIQ versions and about 10 hardcore programmers for the Psion version. There is greater potential if OPL can be ported to Series 60 3rd edition and UIQ 3 (especially for the new devices with a full keyboard).

3.6.2 Roots

There is no direct ancestor to OPL but the language was tailored for database access and based on the BASIC programming language, which was then available for many home computers (e.g. ZX Spectrum).

OPL was originally part of the firmware of Psion devices like the Sienna, 5MX, Revo etc. But as of Symbian OS v6, the OPL runtime was no longer included in the ROM and now comes as a downloadable component.

3.6.3 Purposes

OPL is suitable for rapid development and simple real world applications. The language is basic-like, easy to learn and applications can be written and tested directly on the device.

The native look and feel makes it compelling to the user and disguises the simplicity of the Runtime Environment.

3.6.4 Libraries support for various technologies

The spirit of OPL development is that of knowing the power of Symbian OS C++ APIs, but using OPX extensions to drive the application in a simple BASIC commands-like syntax.

Therefore, the de-facto OPL programming is laced with adding OPX extensions, which means that the notion of available OPL APIs changes according to available extensions. If an API is missing, find an OPX or write it yourself and progress further using the extension.

With OPL, there is support for: all default UI widgets except for tabbed dialogs (standard dialogs, menu's, command bars etc); 2D graphics; basic sound and video via a standard OPX; basic imaging in the language, extended with the Imaging OPX; IR connectivity via a standard OPX, but no remote connectivity; SendAs functionality via a SendAs OPX; full access to Contacts and Calendar via standard OPXs – newer OPXs for more extensive Calendar and Contacts management are in development; full access to the file system via IO functions etc.

3.6.5 Integration with the underlying system

OPL has not been ported to Symbian OS v9 yet and therefore discussing Platform Security is not applicable.

Although calling native code is not supported directly from the language, the standard way to do this in OPL is to write a wrapper OPX that calls the native functions. OPL supported native extensions from the beginning and currently the OPX extensions are commonly used for OPL programming.

The user experience for OPL application is identical to native applications. OPL applications look and behave identically to native applications. Although OPL is interpreted, a translated application is installed in \System\Apps with an accompanying .aif file and can be launched as if it were a native application. To the end user, there is no difference.

3.6.6 Developer productivity

OPL allows you to produce professional looking applications in a very short time, and to program directly on the device. The language is comprised of simple BASIC-like commands and the development process is quick and simple.

3.7 m

3.7.1 Introduction

m is a good example of a runtime that serves a niche of developers and has the likeability factor. To be realistic, it is unlikely that m would compete with mature, popular and leading Runtime Environments. However, it is an indication of the need for various development options and that

Symbian OS enables this need to be fulfilled. In this sense, m is a nice example of a variety of niche Runtime Environments that exist in the Symbian ecosystem.

Runtime Environments such as m have the potential to enhance Symbian OS mostly by adding an on-device development environment using a built-in shell.

The m Runtime Environment, mShell, is a Symbian application with a virtual machine that takes care of pre-emptive multitasking, garbage collection etc; all m scripts run from within that application.

mShell was written by infowing (<http://www.infowing.ch/> - in German). Since mShell has to be activated via SMS, the number of installations is known quite precisely – it was 27,000 as of June 2007. From this number there is an estimated base of around 3,000 active developers. Feedback suggests there are a group of users who are gaining their first programming experiences using m.

3.7.2 Roots

As m is specifically designed for small portable devices, there is no desktop variant.

This drawback for such Runtime Environments is the lack of large developer community and existing content. The potential is that the Runtime Environment can tailor mobile specific support without the burden of legacy frameworks, APIs, idioms etc.

Although there is no direct ancestor of m, developers will find language constructs familiar from Modula-2, Java and awk.

3.7.3 Purposes

m was originally designed as a glue language, exposing the APIs for a phone's standard components (messaging, Bluetooth, TCP/IP, camera, contacts & calendar, UI, files etc.). This allowed rapid prototyping of tailored applications as well as exposing the full capabilities of the device to the power user. But it has also proven to be a suitable platform for games, scientific calculations (astronomy), or intelligent content distribution.

m's main strength is, in conjunction with its built-in IDE, the ability to develop software directly on the device, without the need for a PC or supporting software ("code anytime and anywhere").

The language and environment are not ideal for very large, complex pieces of software. The Runtime Environment itself has internal constraints on the number of variables, functions, modules etc. However, these limitations seem adequate considering the purpose of m and could be easily increased at the expense of a larger memory footprint.

3.7.4 Libraries support for various technologies

Now here comes a nice surprise!

m, although not very well recognized and being a "wall flower" is equipped with a good number of modules to support various mobile capabilities. These include UI support for 2D rendering and widgets, a mathematical library, full access to contacts and calendar, BT local connectivity, TCP/IP remote connectivity, SMS and MMS sending and receiving, playing and recording of audio files and sounds, basic camera functions, and telephony support.

That's not bad at all for a niche Runtime Environment!

3.7.5 Integration with the underlying system

The Native Module Interface supports implementation of native modules as DLLs (most of m's standard modules use this) and enables developer to break out of the sand box.

m scripts are normally started from within the m application, like a submenu of the normal Symbian application menu. There are auto start features, so scripts can be started simply by starting m. Alternatively, m and its scripts can also be controlled via SMS commands or when turning on the device.

When sent via BT or MMS or downloaded from a WAP site, script file types are recognized and can be added to the script list with one or two keystrokes.

Users will be restricted to the capabilities of self signed m, or have to sign the m package with their own Developer Certificate.

3.7.6 Developer productivity

m enhances productivity by its simplicity. It attempts to hide the internals of Symbian OS as far as possible via APIs which are abstract enough to generalize the concept. A design goal of m was to be independent of the actual phone platform.

Developers can use the built-in editor and help system with code completion support for on-device development. For PC-based development, mVNC is recommended (another infowing product). mVNC maps the phone screen and keyboard and automatically transferring code between PC and phone.

A large effort was invested in creating enhanced documentation and examples, which encourages many users to take their first steps in m, computer programming and mobile phone technologies.

4 Summary

In this article we first defined key questions for Runtime Environments using the concepts from the previous two articles. Second, we enumerated various available Runtime Environments on Symbian OS in a mix of leading (Flash, Java), market segment oriented (.NET), niche (Ruby, Python), old timer (OPL) and esoteric ("m") Runtime Environments. Last, we gave a chance for each of those Runtime Environments to answer the questions, as applicable.

Other Runtime Environments for Symbian OS (such as Perl, JavaScript, Lua etc.) do exist, but our selection was meant to be mixed and to reflect the variety of requirements, usages and opportunities that exist for application developers.

Next time

In the next instalment we will attempt to guess what lurks in the future for some of the Runtime Environments discussed, and how things might be affected by the formation of developer communities. Due to the lack of a crystal ball, this will be a mere guess on where the current trends will lead us.

Links to resources

The following links provide further details regarding the topics discussed in this article.

Term	Reference
Series index	http://developer.symbian.com/main/tools/runtime/index.jsp

Term	Reference
Flash	http://www.adobe.com/products http://www.adobe.com/products/flashlite/
JavaME	http://java.sun.com/javame/
Ruby	http://developer.symbian.com/main/tools/opensrc/ruby/index.jsp
.NET	http://www.redfivelabs.com/
Python	http://developer.symbian.com/main/tools/opensrc/languages http://wiki.forum.nokia.com/index.php/Category:Python
OPL	http://developer.symbian.com/main/tools/opensrc/languages https://sourceforge.net/projects/opl-dev
“m”	http://www.m-shell.net
General	http://developer.symbian.com/main/tools/opensrc/languages/

[Back to Developer Library](#)

Want to be kept informed of new articles being made available on the Symbian Developer Network?

[Subscribe to the Symbian Community Newsletter.](#)

The Symbian Community Newsletter brings you, every month, the latest news and resources for Symbian OS.