Ruby – the Gredia Port & How to Get Involved

John Pagonis, Pragmaticomm Limited

Published by the Symbian Developer Network

Version: 1.0 – March 2009

1 Introduction

This article discusses the background and philosophy of the GREDIA Ruby 1.9.1p0 Symbian OS port and gives details of how to build, use and extend the Ruby 1.9.1 VM for Symbian OS. The article targets Symbian C++ developers at intermediate level. This project also demonstrates how to use P.I.P.S. for a major open source project (about 400K lines of source code), integrate it with Symbian C++ and make use of `Autoconf` builds for Symbian OS/P.I.P.S.

2 About the Port

2.1 Background

Ruby[1] has been of interest to Symbian developers for some time, since it comprises a language, VM and plethora of libraries that make programming enjoyable, simple and extremely effective in terms of developer productivity. Ruby as a language, according to its creator Yukihiro Matsumoto (a.k.a. Matz), borrows from Smalltalk, Lisp, Eiffel, Perl and Ada. Matz's aim was to 'make the language natural, not just simple' and, as he says, 'Ruby is simple in appearance, but is very complex inside, just like our human body.' By way of an example, consider the code below:

```
# Output "I love Ruby"
say = "I love Ruby"
puts say

# Output "I *LOVE* RUBY"
say['love'] = "*love*"
puts say.upcase

# Output "I *love* Ruby"
# five times
5.times { puts say }
```

Ruby was released to the public in 1995 and since then it has received mass acceptance. Ruby is free to use, copy, modify, distribute and, in general, experiment with. Writing native extensions for Ruby is simpler than in many other environments such as Java, Perl or Python, and this makes it an ideal candidate to experiment with Symbian OS-specific APIs.

Ruby for Symbian OS v9.x was initiated by Roy Ben Hayun and John Pagonis in the spring of 2006. Later, as part of the SDN team, Roy ported Ruby 1.8 MRI (Matz's Ruby Interpreter) to Symbian OS v9.1 in two and a half weeks, without the use of any POSIX-like APIs. Although the port was missing several features, in a short time Ruby 1.8.5 was running successfully on mobile phones such as the Nokia N80 and the Sony Ericsson P990. At the same time, others had also started toying with the idea of Ruby for mobile phones, most notably a now defunct effort springing from a 'Google summer of code' that same year.[2]

---

[1] www.Ruby-lang.org

[2] rubyforge.org/projects/ruby-symbian/

The current port of Ruby 1.9.1 on Symbian OS is being undertaken by Symbian Research as part of an EU-funded mobile grid project called GREDIA.[3] Part of Symbian Research's aim for the project is to allow researchers to quickly prototype and develop grid-friendly web services on Symbian OS devices. Pragmaticomm Limited was consulted and then contracted to create the port and maintain it for the duration of the GREDIA project; this effort started in August 2007 before the official developer release of Ruby 1.9.0 was released. Since then, there have been four major releases of the port: the pre-1.9.0, 1.9.0, 1.9.1preview1 and finally the Ruby 1.9.1p0 port. (In the Ruby world, a minor point release such as 1.9.0 denotes a release which is for Ruby language developers only, whereas an odd one, such as 1.9.1, denotes that it is ready for public usage.) The Ruby 1.9.1p0 Symbian OS port for the GREDIA project is available as an open source project and it is hoped that it will be merged back with the core Ruby project soon.

In undertaking the port, the goal was to have a well integrated Ruby environment on Symbian OS, so that users and developers can rapidly experiment with ideas on Symbian OS devices. To achieve this, we needed both a good foundation to start with, which we found in the optimized, multi-threaded YARV Ruby 1.9 VM, and a plethora of extensions that bring the capabilities of a Symbian OS smartphone to Ruby.

Although the Ruby 1.8 MRI was much easier to port, using much less memory and therefore debatably better suited to mobile phones, we wanted to rise to the challenge and bring the future of Ruby and the new capabilities of Ruby 1.9 (and YARV) to Symbian OS as quickly as possible, rather than linger with an old version.

As far as the author is aware, the Ruby 1.9.x VM port is the only VM on Symbian OS 9.x which is actually multi-threaded, using native EKA2 threads;[4] all other VMs on Symbian OS use green threads.[5]

2.2 Current status of the Ruby VM

The philosophy adopted was to port the new Ruby 1.9 VM to Symbian OS with the minimum amount of changes to the Ruby 1.9 VM codebase as possible. This was so that we could merge and track Ruby developments easily and follow the future of Ruby, rather than fork the Ruby 1.9 VM for Symbian OS. To achieve this, the port made use of P.I.P.S. 1.1 through to P.I.P.S. 1.3 in the early days, and later Open C 1.3, so that OpenSSL could be used.

Currently there are two ports, one for the WINSCW emulator and one for the GCCE Symbian OS build environment. Having the VM ported to the emulator means that developers can write Ruby programs on their desktops with ease and execute them on the emulator, thus allowing really rapid development. Deploying Ruby programs on the device is also very easy using the PC Suite, which allows them to be copied to the phone's `c:\data\ruby` or `e:\data\ruby` default locations.

We chose the path of making the VM available to Symbian OS developers in such a way so that they can not only create programs on Ruby, but also have the ability to easily embed Ruby inside their Symbian C++ applications. This is to encourage multi-language programming and enable this on Symbian OS devices. Ruby has been gaining ground in developing domain specific languages and this is something that can now be realized on Symbian OS.

---

[3] www.Gredia.eu

[4] Harrison, Richard and Shackman, Mark (2007) *Symbian OS C++ for Mobile Phones, Vol. 3*

[5] Green threads are scheduled by the Virtual Machine (VM) rather than natively by the operating system.

The Ruby project comprises three sub-projects: the VM, the VM launcher and a third which is an application that embeds the VM inside it (this will be discussed later). All three, `Ruby_VM.mmp`, `Ruby_App.mmp` and `Ruby_S60.mmp` respectively, are built via the `symbian\group\bld.inf` file.

The VM launcher's responsibilities are to receive user input about which Ruby program to execute, to launch the VM with that program information and to communicate user input and output to and from the VM.

When the VM launcher gets the pathname of the Ruby program to execute from the user, it creates a new process, inside which it starts the VM and passes to it the pathname of the program to execute. From that point onwards, the VM and the VM launcher application communicate through EKA2 message queues[4]. This communication takes place by sending characters between the two processes on the message queues, therefore emulating what on UNIX would have been 'stdio streams'. In order to achieve this, the part of the Ruby VM that emits characters to 'stdout' (`rb_write_internal`) and the part that receives characters from 'stdin' (`io_fillbuf`) in the `io.c` file had to be tweaked (for Symbian OS builds) so that they would communicate with the message queues. Doing so allows any Symbian C++ control to potentially communicate with the VM directly (or through the control stack). In the example `Ruby_App`, this is done very simply (and primitively) in the `CEikAppUi` that sends the received characters to a `CEikEdwin` and the user key presses to the VM.

The parts of the Ruby VM which are built and included in the final binary for each platform (apart from the Win32 platform) are controlled by a GNU Autoconf process that outputs a `config.h` file. This directs which conditional parts are compiled in the VM through the numerous `#define` directives that are in all VM files. Consequently, the `config.h` identifies which CPU/OS attributes, functionality and 'UNIX-like' APIs (using P.I.P.S.) are present on the platform. Where some functionality is not present or not working, the VM either ignores it or replaces it with code found under its `\missing` directory. For the Symbian OS port, all the files, including `config.h`, that are generated from the Autoconf stage are stored under `\symbian\generated`, so that developers don't have to run Autoconf scripts (using cygwin or some other POSIX host) before they can build the port (see Section 3 below).

3 Getting and Building the Project

To build and deploy the GREDIA Ruby 1.9.1p0 Symbian OS port, developers should install either the Symbian OS v9.1/S60 3rd Edition MR or Symbian OS v9.2/S60 3rd Edition FP1 SDKs with P.I.P.S. v1.3. To make use of the OpenSSL extension, Open C v1.3 is also needed. The port has been developed and tested on the Nokia E90, E61/I, E51 and N95 and it should run on all S60 3rd Edition MR and FP1 devices.

To get the code, you need to either use a Subversion[6] client or download it from the www.pragmaticomm.com/MobileRuby web site. To use a Subversion client, point it to ella.pragmaticomm.com/svn/symbian-ruby/ and just download the head revision of the project.

There are two main stages to building Ruby 1.9.1p0. The first uses Autoconf which invokes all of the Ruby core developer's `make` process, and needs to be applied only once per major release; the auto-generated files for this port are pre-supplied in the '`symbian\generated`' directory. The second stage performs the builds for the Symbian OS GCCE and WINSCW platforms.

If you want to build from scratch and regenerate the auto-generated files sourcing from the Autoconf stage, read the `SYMBIAN.README` file first. This stage should be seen as 'reserved' for those who really change the language or the structure of the VM and for the maintainer of the port

[6] subversion.tigris.org/

who has to do it every time there is a major release. It's not needed if you are only building the port or adding extensions to it.

The following process is for the second stage only – for those who just want to build the source tree and either fix bugs or add extensions – and shows how to build on a Win32 host from the source:

1. Set up a Nokia S60 3rd Edition FP1 SDK (and optionally Carbide.c++ v2.0)
2. Download and install the Open C (this includes P.I.P.S. v1.3) plugin from: www.forum.nokia.com/Resources_and_Information/Explore/Runtime_Platforms/Open_C_and_C++/
3. There is a known issue with a core Symbian OS header, `e32def.h`. To make it work with Open C, go to line 2804 of `%EPOCROOT%\include\e32def.h` and place a conditional compilation directive around the declaration, like this:

```
#if defined(__cplusplus)
static const char* const KSuppressPlatSecDiagnostic =
                                              KSuppressPlatSecDiagnosticMagicValue;
#endif
```

4. Then you need to build as usual from the `symbian\group` directory by doing:

```
bldmake clean
bldmake bldfiles
abld build <platform> <variant_build> Ruby_VM
abld build <platform> <variant_build> Ruby_App
```

At the end of the build process, two binaries are built: the VM launcher application and the VM with the TCP/IP, OpenSSL, digest, `zlib` and `fnctl` extensions. The MMP file reflects the different components should you wish to separate them (for example, to add or remove extensions).

The VM is built by default with the localization files `ascii.c`, `euc_jp.c`, `shift_jis.c`, `unicode.c`, `us_ascii.c` and `utf_8.c` for sizing reasons. Should you wish to add more, look for them in the `\enc` directory of the main Ruby tree and add them in the `Ruby_VM.mmp` file.

The build scripts will also copy the files from the Ruby root directory's '`test`' and '`lib`' directories in to your `%EPOCROOT%epoc32\winscw\c\data\ruby\` '`lib`' and '`test`' directories respectively.

4 Deploying and Using the Port

This section describes how to deploy and use the Ruby 1.9.1p0 port for the first time.

Following the build process outlined above, the `symbian\sis` folder will contain four PKG files; these are:

- `ruby-lib.pkg`, for deploying all the pure Ruby libraries, as found in the `<main Ruby project path>\lib` and `\bin` directories. This package is optional for someone that just wants to program using the standard Ruby 1.9 libraries (which are built in the VM).
- `ruby-test.pkg`, the official Ruby regression tests.
- `ruby.pkg`, the VM and launcher.
- `ruby_S60.pkg`, an application that embeds the launcher and VM in one.

With this Ruby port (including all the current extensions) you can also build self-signed binaries by calling `createsis create` for the PKG files. The resulting SIS files can then be installed on the device. Remember to also install the Open C SIS files, apart from the `stdioserver` which is not needed.

To try rapid Ruby development for Symbian OS on the desktop, use the emulator (WINSCW) build and execute Ruby programs by running the Ruby VM launcher from the emulator. Ruby programs can then be edited and executed in place from the `%EPOCROOT%epoc32\winscw\c\data\ruby\` directory. To run a Ruby program from the Ruby VM launcher, simply select ‘Load and execute’ from the ‘Options’ menu, then give the full path of the program (such as `c:\data\ruby\getgoogle.rb`).

5 Embedding Ruby 1.9.1 in Symbian C++ applications

There are two main ways to embed the Ruby VM in a Symbian C++ application. One is to simply enter its `main` function by supplying it with the correct arguments inside the application’s main thread (which may sound strange to non-Symbian C++ developers and in fact is not strictly legal ISO C++) and the other is to spawn a separate thread and supply it with the correct thread function to start the appropriate Ruby program.

The latter method is demonstrated in the `CRuby_S60AppUi::VMThreadFunction()` of the application inside the `symbian\src\Ruby_S60AppUi.cpp` file. To communicate with the VM you can set up local message queues. Embedding of Ruby is demonstrated in the `Ruby_S60` application, which can be built by issuing:

```
    abld build <platform> <variant_build> Ruby_S60
```

The `Ruby_S60.mmp` project file demonstrates how to build an application that embeds the Ruby VM and its extension in one executable. Doing so makes launching of Ruby programs faster and allows more control over the lifecycle of the Ruby VM thread, but currently has a caveat that prevents the VM from being re-launched with a new program, due to the fact that the VM is executed from its `main` function (which naturally assumes that it is launched in a new process every time, and so all global writable static data are initialized to `NULL`).

Integration with any Symbian C++ application is fairly simple: user input and output (`print`, `puts`, `getc`, `gets`, etc.) are available and not bound to any particular console implementation, and the P.I.P.S./stdlib console is not required. File system access is available, as well as almost all Ruby 1.9 VM built-in classes and pure Ruby libraries (apart from some Process and signal related ones).

6 Extending the Ruby 1.9.1 VM for Symbian OS

Writing native extensions for Ruby is well documented by the core Ruby team in the `README.EXT` file of the Ruby project.

By default, the official Ruby 1.9 VM project statically embeds certain extensions within the VM (such as TCP/IP sockets and OpenSSL), whereas other extensions can be dynamically loaded. The list of extensions that are statically linked with the VM is found in the ’`ext\Setup`’ file and for some operating systems there are separate setup files too.

The Symbian OS port doesn’t use this `Setup` file mechanism at the moment but nevertheless statically embeds all the extensions inside the VM binary (like the JVMs do for CLDC profiles). This decision is only temporary and is due to (i) ease of debugging of extensions during porting and (ii) the nature of the Symbian OS building process. All the extensions that are necessary for the VM to be useful on a mobile device were ported and statically linked, and in fact were guided by the list in the `Setup` file. Currently, a close inspection of Ruby's `r_call_inits` in `inits.c` shows that for Symbian OS builds we have statically added the initialization functions of all the non-Symbian OS specific extensions that we have ported.

At the time of writing, Symbian Research are releasing two very interesting Symbian OS-specific Ruby 1.9 VM extensions that demonstrate how to bring native APIs to Ruby. One is for accessing the battery and signal strength, which bridges to `ETel`, and the other gets a simple location fix from the on-device GPS. These two extensions, `ELoc` and `ETel`, are not currently built by default as part of the project, but can be found in the `ext\etel` and `ext\eloc` directories respectively. Note that including the `ELoc` extension with the VM will prevent self-signing for pre-FP2 SDKs. Nevertheless, these extensions demonstrate how easy is to use Symbian OS features from Ruby and what the potential for developer productivity is.

For example, consider the following code snippet that obtains the battery and signal strength levels:

```
battery = ETelephony::battery_level
signal = ETelephony::signal_strength
printf "Battery: %d(%d)\nSignal: %d(%d)\n",
       battery[0], battery[1], signal[0], signal[1]
```

In the above snippet, the `[0]` values are the raw values from the hardware, expressed as a percentage, while the `[1]` values indicate the number of bars to display, in the case of signal strength and the charging status in the battery case (`0` = unknown, `1` = battery being used, `2` = on external power, `3` = no battery present, `4` = fault).

A more advanced but still very short and powerful Ruby program for Symbian OS may look like the following, which is taken from the LBS API extensions example developed by Symbian Research.

```
require "ELocation"
position = "%d latitude, %d longitude, and %d altitude" %
                               ELocation::last_known_position.map! { |x| x*1000 }
puts "Last known position is: %s" % position

# Current position with default 10s timeout
loc = ELocation::current_position
if loc
position = "%d latitude, %d longitude, and %d altitude" % loc.map! { |x| x*1000 }
    puts "Current position is: %s" % position
else
    puts "Timeout"
end

# Current position with 0.5s timeout
loc = ELocation::current_position(500)
if loc
position = "%d latitude, %d longitude, and %d altitude" % loc.map! { |x| x*1000 }
    puts "Current position is: %s" % position
else
    puts "Timeout"
end
```

7 How to Get Involved

There are at least five main ways to get involved and help with the Ruby 1.9 port for Symbian OS:

- Run as many of the core Ruby regression tests as possible on your Symbian OS device and report your findings. To run the tests, install the `ruby-test.sis` application on the device together with all other relevant SIS files, go to the `c:data\ruby\test` directory and run tests or create your own. You can submit reports on the Symbian Ruby mailing list (ruby@Symbian.com), or ask Pragmaticomm (symbian-ruby@pragmaticomm.com) for a read-write software configuration management account and the issue tracking system that Pragmaticomm have set up.
- Submit fixes to problems you find for the port.
- Develop or extend the existing Symbian OS extensions. Start by looking in the Ruby `\ext` directory for extensions specific to Symbian OS, such as `ELoc`.
- Port more of the existing Ruby 1.9 VM extensions to Symbian OS.
- Translate any of the port documentation, bug reports, etc., to Japanese!

The effort for the port was supported by the GREDIA project so far but it needs to continue beyond that, which can only happen if Symbian OS developers embrace the project and contribute back. The potential for extending the Ruby VM capabilities on Symbian OS is endless and with this Ruby port, developers don't have to ask for permission from anyone to do so for their own projects. The Ruby 1.9 VM is also a very capable virtual machine implementation that gives developers, possibly for the first time, the chance to experiment with VMs on mobile phones without a huge cost or learning curve. Therefore, even as an educational (if not commercial) tool, the Ruby VM port for Symbian OS has tremendous value.

8 Author Profile

John Pagonis has been working for about 11 years in the mobile telecoms industry at Ericsson, Symbian Ltd and Pragmaticomm Limited, engineering the software that powers modern, advanced mobile phones.

In the domain of mobile phones, John's experience ranges from communication protocols and infrastructure, security, location-based services, operating systems and middleware design, to software engineering methodologies, developer consulting, team coaching and organizational improvements. John has authored several articles for the SDN and has been a contributing author for both volume 2 and volume 3 of *Symbian OS C++ for Mobile Phones*.[4] He has been a visiting lecturer at City University in London and holds an MSc(Hons.) in Computer and Information Networks and a BEng(Hons.) in Computer and Networks, both from the University of Essex Electronics Systems Engineering department.