

Blue Ruby: A Ruby VM in ABAP



Applies to:

SAP NetWeaver (ABAP) 7.00 and later

SAP ERP 6.0 and later

Summary

The Blue Ruby research project is about creating an enterprise-ready dynamic language environment for the programming language Ruby that runs inside the ABAP Virtual Machine. It combines the “best of both worlds” – lightweight, loosely-coupled, agile programming via Ruby, executed within the robust, proven SAP Web Application Server for ABAP. As a proof of concept, it demonstrates how to simplify and reduce costs of application development for SAP platforms by leveraging the popularity and productivity of Ruby, while retaining scalability and robustness similar to ABAP’s, and providing seamless local integration with existing ABAP applications.

Author: Juergen Schmerder

Company: SAP Labs, LLC

Created on: 15 March 2009

Author Bio



Juergen worked as a developer, architect and solution manager in various groups at SAP after he joined SAP in 1999. In late 2007 he moved to the SAP Research Labs in beautiful Palo Alto, CA and has been working on the Blue Ruby project since that.

Table of Contents

Introduction & Motivation	3
Technical Approach	4
Compiler, BRIL and Virtual Machine	5
Runtime Library	5
Secure Bridges & Blue Sec.....	6
File System	6
Integrated Development Environment	7
What can you do with Blue Ruby?.....	8
Call ABAP Function Modules and Enterprise Services	8
HTTP Server	9
HTTP Client.....	10
Implement BADls	10
Invoke Ruby code from ABAP	10
Implement Ruby Libraries in ABAP	11
Experimental Libraries	12
Limitations.....	13
Outlook.....	14
For more information and comments.....	14
Related Content.....	15
Copyright.....	16

Introduction & Motivation

History has shown that there are major new programming languages every 10 years or so ^[1]. The reasons for this are many, whether it be the needs of different domains, the needs of different types of programmers or changes in programming philosophies. In the case of dynamic programming languages, Gartner predicts that by 2013, there will be a community of over 9.5 million developers worldwide utilizing PHP or Ruby, of which approximately 4.4 million will be Corporate IT Developers or Independent Software Developers (ISVs). The Ruby language ^[2], invented by Japanese Yukihiro Matsumoto in 1993, has especially gained a lot of traction lately, winning Language of the Year 2006 on the TIOBE index ^[3]. Several different implementations of the Ruby language exist ^{[4][5][6]} in addition to the reference implementation MRI (Matz's Ruby Interpreter), including those coming from IT platform technology vendors Sun and Microsoft who are working on the integration of Ruby into their Java / .NET platforms.

The Blue Ruby research project is about creating an enterprise-ready dynamic language environment for the programming language Ruby that runs inside the ABAP Virtual Machine. It combines the “best of both worlds” – lightweight, loosely-coupled, agile programming via Ruby, executed within the robust, proven SAP NetWeaver Application Server ABAP ^[7]. In many ways, Blue Ruby makes the simple things simple and the complex things possible.

After considering various technical approaches, the Blue Ruby team selected the most evolutionary and least SAP-disruptive approach in order to improve the short-term productization potential. We implemented the Blue Ruby runtime as an extension to the ABAP Virtual Machine. The Blue Ruby runtime and Blue Ruby libraries are implemented in the ABAP language and allow interaction with the surrounding ABAP environment.

Blue Ruby...

- is an evolutionary, non-disruptive extension of the ABAP VM
- runs inside the ABAP VM; does not require any additional servers or landscape
- follows a strict sandbox approach which allows consumer-specific adaptation inside the platform, without breaking platform consistency
- offers capabilities that are well suited for composition and platform extension scenarios
- provides an environment to write Domain Specific Languages (DSLs)

Blue Ruby is potentially the first phase of a larger phased “Open ABAP” approach, where SAP could gradually build up an open ABAP runtime that would allow programmers to program in ABAP, Ruby, PHP, Python, and multiple future languages. The current Blue Ruby technical approach is generic and could lead to a Phase 2 Blue PHP, Phase 3 Blue Python, etc.

The intent is to reach programmers that are already choosing to build simple applications or extensions. For the near term, most mission critical enterprise applications and customizations will most likely continue to be implemented in Java or ABAP (in SAP case). Blue Ruby should be viewed as a complementary offering for programmers.

Technical Approach

When we say “we want to run Ruby on the ABAP stack” we more specifically mean "on the ABAP call stack". Ruby programs should be able to share data and data representations with ABAP with minimal overhead in moving between the two. Ruby methods should be callable from ABAP programs and vice versa with a high frequency. To achieve this, we want to make Ruby and ABAP execute together on the same virtual machine by extending the ABAP virtual machine, as outlined in Figure 1.

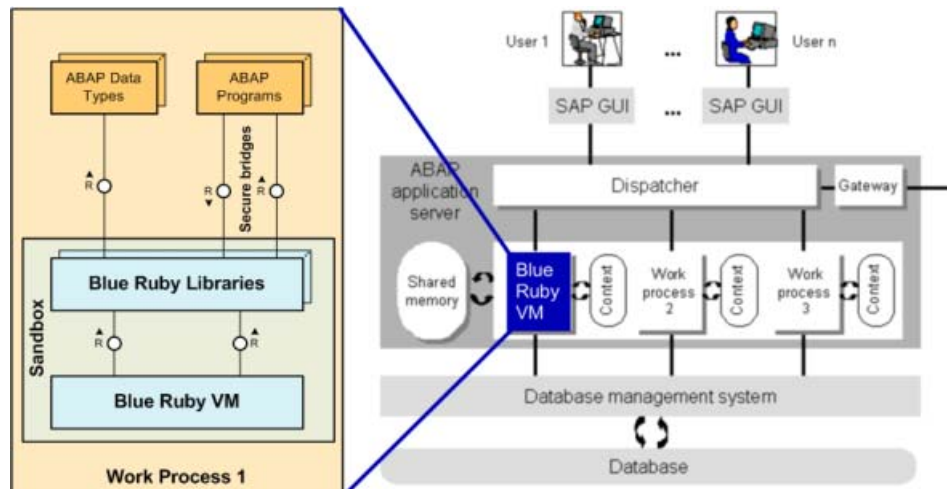


Figure 1 - Blue Ruby inside the ABAP VM

The main pieces of the approach to consider are:

- A compiler, that transforms Ruby source code into a format that can be executed inside an ABAP Work Process
- A virtual machine, which handles program loading, method invocation, control and data flow. Everything is an object in Ruby; therefore the basic operations of the virtual machine are in creating objects and sending messages to them
- The runtime library, which enables the Ruby type system and provides the basic structure and relationships between classes, modules, methods and object instances and their lifetime
- The built-in libraries of predefined operations on the base data types such as strings and numbers
- A facility to add extension libraries that are wholly or partially implemented in ABAP
- Secured bridge packages, which allow access to the functionality of the underlying host platform in a secure way by establishing a well defined sandbox concept
- A database-backed virtual file system that stores the Ruby source code files on the ABAP Application Server

Compiler, BRIL and Virtual Machine

To run a Ruby program in Blue Ruby, the Ruby code is first compiled into an intermediate representation called the Blue Ruby Intermediate Language (BRIL). BRIL codes are to Blue Ruby what Java byte code is to Java (and ABAP loads are to the ABAP VM). Compilation will happen every time a Ruby source code file is saved to the Blue Ruby file system, thus not affecting startup times. Despite turning the source code into something that can be easily picked up by the ABAP VM (a pure interpreted approach, though common in Ruby, would have serious performance issues), the interpreted “scripting-like” flavor of the Ruby language remains unchanged. For example, syntax errors in the source code will not lead to compilation failure but rather raise an exception at runtime. Table 1 shows a simple Ruby program and the corresponding BRIL code as generated by the Blue Ruby compiler.

Table 1 - BRIL Code example

Ruby Source Code	BRIL Code for Blue Ruby VM (simplified)
<pre># define a variable name = 'Blue Ruby' # print a formatted string puts "Hello, #{name}"</pre>	<pre>xrb_init. xrb_var_def `name` 0. xrb_vars_init. xrb_string_new `Blue Ruby`. xrb_var_set 0. xrb_set return_val self. xrb_prepare_method_call `puts`. xrb_string_new `Hello, `. xrb_set current_string return_val. xrb_var_get 0. xrb_call_method `to_s`. xrb_string_append. xrb_set return_val current_string. xrb_push_arg return_val. xrb_perform_method_call.</pre>

The Blue Ruby compiler is implemented in the Ruby language, using Ryan Davis' `ruby_parser` ^[8] and turning the resulting s-expressions into BRIL code. Currently, a separate Ruby 1.8.6 runtime is necessary for running the compiler, but the goal is to make our compiler work in Blue Ruby as soon as possible.

Runtime Library

Upon execution of a Blue Ruby program, the BRIL codes are loaded into the ABAP system using the normal ABAP Load facility. By building certain extensions on top of the ABAP VM (this is then called the Blue Ruby VM) we are able to execute those BRIL codes in a similar fashion to how ABAP Loads – the equivalent of byte code for ABAP – are normally executed by the ABAP VM.

We reuse the ABAP VM as much as possible for things such as garbage collection, session handling, memory management, etc. It should be kept in mind that unlike a normal Ruby environment, this is all running in the context of an application server – the NetWeaver Application Server (ABAP). This means that all of this is operating in the normal context of ABAP work-processes, roll areas, user authentication & authorizations, etc. While we leverage the ABAP VM as much as possible, it is important to realize that not everything that the Blue Ruby VM needs can be provided by the ABAP VM. This is where the Blue Ruby Runtime Library is needed. While executing the BRIL codes the Blue Ruby VM relies on the existence of the Blue Ruby Runtime Libraries, so that the semantics of Ruby's type system may be supported. For example, it would not be possible for the Blue Ruby VM to use the existing method dispatch mechanism in ABAP, because the Ruby method dispatch mechanism with classes and mixed-in modules significantly differs from the ABAP method dispatch mechanism. The important thing to realize is that it is the Blue Ruby Runtime that provides the basic mechanisms needed for Ruby's dynamic features to work on top of the ABAP VM. Built-in libraries are a core part of the Ruby language: as in Ruby 'everything is an object', even the atomic data types such as String, Integer, etc. are represented as classes and all operations on these data types – e.g. the '+' operator on Integers – are implemented as methods. In Blue Ruby, each of these methods in turn is implemented as an ABAP class.

Secure Bridges & Blue Sec

The built-in libraries make the data types and utility classes of Ruby available to Blue Ruby developers by boxing corresponding ABAP data types. The Blue Ruby VM, Runtime Library and Built-in Libraries make Blue Ruby a fully Ruby compliant environment running inside the ABAP VM, but not yet interacting with other (ABAP) programs running on the SAP NetWeaver Application Server. The wealth of libraries and platform functionality available in the ABAP landscape is not something Blue Ruby can turn its back on. However, access to this powerful functionality from Blue Ruby must be carefully controlled. For this purpose, the Blue Ruby environment provides a secured sandbox model, which guarantees that a Ruby program can access ABAP functionality via well-defined and controlled interfaces, called “bridges”. Using these bridges, Ruby developers are able to both consume data managed by ABAP applications and also extend ABAP applications – Blue Ruby comes with a 2-way integration where Ruby can call certain ABAP APIs (e.g. RFCs or Enterprise Services) and ABAP can also call Ruby code (e.g. via the BAdI bridge explained later in this document).

However, this 2-way integration is not an arbitrary one – Blue Ruby cannot make calls to any piece of ABAP code residing on the NetWeaver Application Server and especially Blue Ruby is not able to modify the database directly. Instead, communication with the ABAP application is only possible via well-defined interfaces – the bridges. The idea behind this restriction is to open up Blue Ruby for communication with ABAP Applications – but only in a very controlled way that protects the Ruby developer from breaking the underlying platform. Without this functionality, there would be little chance of adoption of Blue Ruby in production environments, as developers could write malicious Ruby code which could bypass the ABAP authorization mechanisms and/or cause inconsistencies by directly writing to the database. So there is a need to control Blue Ruby's interaction with the Application Server provided services, APIs and ABAP programs.

Whenever a Ruby program runs inside the sandbox, the normal ABAP authorization concepts remain in place. Users have to authenticate and will have exactly the same authorizations as without Blue Ruby. Furthermore, a Secure Class Loading mechanism provides an even more secure sandboxing for Ruby implementations by controlling which Blue Ruby libraries (e.g. RFC) are allowed to be used in which application types (e.g. Ruby based BAdI implementations). A Blue Ruby system administrator can set policies programmatically from the Ruby workbench using a simple security policy DSL. A policy rule expresses which Blue Ruby libraries (e.g. RFC) are allowed to be used in which application type (e.g. BAdI).

Example – deny execution of BQL queries when implementing any BAdI. Furthermore deny updates to taxes within the tax calculation BAdI

```
require "BlueSec/secure_class_loader"
SecurityPolicy = Secure_class_loader.new
SecurityPolicy.deny( "BAdI", "BQL", :load )
SecurityPolicy.deny( "BAdI.TaxCalculationBAdI", "RFC.RFC_UPDATE_TAX", :call )
```

File System

The Blue Ruby File System is a virtual file system implemented inside the NetWeaver Application Server. It consists of folders and files (technically realized as database tables inside the ABAP Data Dictionary – the relevant database tables are completely buffered for fast access), and offers API's for uploading, retrieving and deleting files. Files are versioned and a roll-back to an earlier version is always possible. Ruby source code files (files ending on the extension '.rb') are automatically compiled into BRIL code when uploaded to the file system.

The Blue Ruby virtual file system can be used for Ruby source code files as well as any other text or binary data that needs to be stored. Mixing concepts from 'real' file systems and the ABAP Data Dictionary, content of folders can be transported via the ABAP Transport Management System and multiple clients in one ABAP system can 'mount' different volumes to folders and decide whether to securely protect their content or share it with others, thus enabling different views on the file system in a multi-tenant environment.

Integrated Development Environment

Blue Ruby offers various ways to get Ruby source code into the Blue Ruby file system. Developers are free to use their IDE or Editor of choice and upload file via HTTP (WebDAV access is planned for the near future). Alternatively, Blue Ruby offers a fully web-enabled development environment to get started easily, as well as a SAPGui based development environment. In future, Blue Ruby might integrate more deeply with IDEs like Eclipse or the ABAP Workbench, based on developers' demands.

What can you do with Blue Ruby?

Note: As this is the first question asked whenever we talk about Blue Ruby: no, we don't run Rails – **not yet...**

First and foremost, Blue Ruby is an implementation of the Ruby language, thus allowing the execution of nearly any code written in the Ruby language. The current implementation is mostly based on Ruby 1.8.6 (which is the most widely used version at the moment), but we already prepare for the switch to Ruby 1.9. At the creation time of this document, the coverage, measured by executing the RubySpec^[9] set of specifications, is around 70% – and increasing.

While Blue Ruby eventually should run arbitrary Ruby programs, the full potential of Blue Ruby is only realized when Ruby code is integrated with the surrounding ABAP-based environment. Therefore, Blue Ruby offers several bridges to the ABAP world. Using these bridges, Ruby developers are able to both consume and modify data managed by ABAP applications and also extend ABAP applications. Blue Ruby comes with a 2-way integration where Ruby can call certain ABAP APIs, such as Function Modules and Enterprise Services, and ABAP can also call Ruby code – e.g. via the BAdI bridge explained later in this document.

In the following, the most important bridges between ABAP and Ruby are explained in more detail.

Call ABAP Function Modules and Enterprise Services

Via the RFC Bridge, Ruby developers can call remote-enabled function modules and use their importing, exporting and changing parameters in the Ruby code. Although only remote-enabled function modules can be used, the call will be a local one – this means the function module runs within the same ABAP work process that runs the Ruby code. The reason for only offering a bridge to remote-enabled functions, rather than letting the Ruby developer call any function module which resides within the system, is not a technical one, but more driven by security aspects: RFCs define a certain contract and are provided to be called from outside the system, whereas arbitrary function modules might not be supposed to be called by any other application than the one they belong to. By expressing a security policy in the BlueSec DSL described earlier in this document, IT administrators can set up further restriction on the functions that can be called by Ruby programs, thus enforcing even stricter contracts – e.g. only allowing the call of BAPIs (Business Application Programming Interfaces).

Example – call a BAPI to read Business Partner data and print the name of partner '1'

```
require 'rfc'
bapi = Rfc.new "BAPI_BUPA_CENTRAL_GETDETAIL"
bapi.businesspartner = '0000000001'
bapi.call!
puts bapi.centraldataperson.firstname
puts bapi.centraldataperson.lastname
```

Dynamic language magic happens in line 3, when the instance of an ABAP function module class suddenly provides a method `businesspartner=`, just because the particular BAPI `BAPI_BUPA_CENTRAL_GETDETAIL` offers an importing parameter `BUSINESSPARTNER`. Same in lines 5/6 for the exporting structure `CENTRALDATAPERSON` that is accessible just like a normal Ruby object.

In the same way that is used to call ABAP function modules, Enterprise Services that reside on the ABAP system that runs Blue Ruby can be invoked. The invocation of the Enterprise Services will not be done by sending a SOAP message, but again be a local call to the service provider class that implements the Enterprise Service. This local call is supposed to be more efficient than sending a SOAP message, especially in a scenario where only a small portion of the data that is returned by the Service is actually used by the application code – in these cases, projections of the data can be implemented in Blue Ruby, running inside the ABAP application server and only returning the data that is actually requested by the client.

Example – call an Enterprise Service to read Supplier data and print the name of supplier ‘1’

```

require "esoa/inbound_service"
service = InboundService.new("SupplierBasicDataByIDQueryResponse_In",
  "http://sap.com/xi/APPL/SE/Global" )
service.input.supplierbasicdatabyidquery.supplier_basic_data_selection.
  supplier_id.value = "0000000001"
service.call!
puts service.output.supplierbasicdatabyidresp.supplier.basic_data.common.name.
  first_line_name

```

HTTP Server

Using the HTTP Bridge, Ruby developers can implement Web Applications that can be accessed via the Internet Communication Manager (ICM) which is part of the SAP NetWeaver Application Server. ICM listens for incoming HTTP request and delegates these requests to the appropriate HTTP handler, which is found based on the URL of the request. Such an HTTP handler is usually an ABAP class implementing the IF_HTTP_EXTENSION interface or a Business Server Page (BSP). The Blue Ruby HTTP Bridge adds another way to implement HTTP handlers for ICM – Ruby. In a way, the Blue Ruby HTTP Bridge is our answer to popular Ruby web servers such as Webrick or Mongrel. As Blue Ruby is a framework that lives within the NetWeaver Application Server, the ability for HTTP requests to trigger the execution of Ruby code is of course an important feature. As with other ABAP extension libraries, Blue Ruby Web Server consists of a number of Ruby libraries that wrap existing ABAP libraries for handling HTTP requests. The Blue Ruby Web server follows the convention-over-configuration paradigm as popularized by the Ruby on Rails framework. As such we want to avoid having to configure new endpoints for each HTTP service via the SICF transaction. To do this we register one generic dispatcher for all Blue Ruby HTTP services. This registers the following URL pattern for the Blue Ruby server:

http://<server>:<port>/sap/sruby/<handler_class> (server and port as configured in ICM)

Based on naming conventions, if for example a *HTTP GET* request is sent to the *http://<server>:<port>/sap/sruby/hello_world* then the dispatcher will try to find a Ruby program called *hello_world.rb* located in the */Handlers* folder on the Blue Ruby file system. Then an object of the class called *HelloWorld* is created and the *do_get* method is invoked on that object.

Example – implement a ‘servlet’ that returns “Hello World” to an HTTP GET request

```

require 'net/http_server'
class HelloWorld
  def do_get request, response
    response.body = "Hello World"
  end
end

```

(save this file as *hello_world.rb* in the folder */Handlers*)

It is worth mentioning that due to the architecture of the SAP Web Application Server (an HTTP server is always running), the way to handle HTTP requests in Blue Ruby might look a bit unfamiliar to developers who have experience with the usual Ruby web servers Webrick and/or Mongrel. Whilst we think the Blue Ruby way to implement servlets is even simpler than Webrick, for compatibility reasons we are also considering a library with Webrick-like behavior.

HTTP Client

Blue Ruby can also play the role of an HTTP Client. Again, the current implementation is a bit different from the usual Ruby `net/http` library – gaps will be closed over time... A Blue Ruby program can send an HTTP request to a web server (directly or via proxy) and receive the response status, header and body.

Example – google for “SAP Blue Ruby” via proxy

```
require 'net/http_client'
http = HTTPClient.new('google.com', false, 80, 'proxy', 8080)
req = http.request
req.request_method= 'GET'
req.request_uri = '/search?q=SAP+Blue+Ruby'
http.send_request
res = http.receive_response
puts res.body if res.status == 200
```

Implement BAdIs

Business AddIns (BAdIs) are SAPs main enhancement concept^[10] – a BAdI is basically an exit foreseen by an SAP developer to let others enhance the functionality of his application. The Blue Ruby BAdI Bridge is described in more detail later in this document. In a nutshell, the BAdI bridge enables developers outside SAP (e.g. at customers or partners) to implement these enhancement points in Ruby rather than ABAP. The registration of a custom BAdI implementation will be done by executing a Ruby script in Blue Ruby – the script registers the BAdI implementation by creating the necessary metadata and generates a corresponding ABAP wrapper class that calls the relevant Ruby method at runtime.

Example – a real life example would go beyond the scope of this short document, so this is only a pseudo-BAdI

```
require 'badi'
# "BADI_NAME"           = name of a BAdI definition in the ABAP system
# badi_method           = name of a method in BAdI "BADI_NAME"
# in_number             = importing parameter of badi_method
# out_text              = exporting parameter of badi_method
implement :badi => "BADI_NAME" do
  at :badi_method do |parameters|
    if parameters[:in_number] < 100 then
      parameters[:out_text] = 'low'
    else
      parameters[:out_text] = 'high'
    end
  end
end
end
```

Invoke Ruby code from ABAP

ABAP developers are able to call Blue Ruby methods from their ABAP code – a Ruby factory class wraps the ABAP to Ruby communication and provides methods to find Ruby files and classes, create objects and invoke methods. Before and after invocation of a Ruby method from ABAP, method parameters need to be converted from ABAP types to Ruby objects and vice versa. For very simple cases, a convenience method `CALL_METHOD_ABAP_TYPES` can be used with native ABAP types and handles all the conversions around the method invocation. This technique is used i.e. in our ABAP-to-Ruby bridges such as the BAdI bridge.

Example – use ABAP to calculate 7 + 3 via Blue Ruby

```

Report ZABAP_RUBY.

data: BLUERUBY type ref to /SRUBY/IF_BLUERUBY,
      FIXNUM1 type ref to /SRUBY/CL_OBJECT,
      FIXNUM2 type ref to /SRUBY/CL_OBJECT,
      RESULT type ref to /SRUBY/CL_OBJECT,
      ABAP_RESULT type I.

create object BLUERUBY type /SRUBY/CL_BLUERUBY_FACADE.
FIXNUM1 = BLUERUBY->CONVERT_ABAP2RUBY( 7 ).
FIXNUM2 = BLUERUBY->CONVERT_ABAP2RUBY( 3 ).
RESULT = BLUERUBY->CALL_METHOD(
          METHOD_NAME = '+'
          RUBY_RECEIVER_OBJECT = FIXNUM1
          ARGUMENT1 = FIXNUM2 ).
call method BLUERUBY->CONVERT_RUBY2ABAP
  exporting
    RUBY_OBJECT = RESULT
  importing
    ABAP_DATA = ABAP_RESULT.
write: 'Blue Ruby thinks that 7 + 3 = ', ABAP_RESULT.

```

Implement Ruby Libraries in ABAP

When writing a Ruby class, developers can decide to implement the complete class or selected methods in ABAP rather than Ruby. In a way, this mechanism resembles the C Extensions in standard Ruby. We don't expect this feature to be used by the typical Ruby developer, however for some performance-critical libraries the option to implement the functionality in ABAP and use it from Ruby is available – and most of the Blue Ruby core libraries are written that way. To implement a Ruby method in ABAP, one would simply link the relevant Ruby class or module to a corresponding ABAP class, define the method with the keyword **abap_def** instead of the standard **def** and follow some naming conventions that help the Blue Ruby runtime to find the ABAP implementation.

Example – define a Ruby class and implement a method in ABAP

```

class MyClass
  abap_class 'ZCL_MY_CLASS'
  abap_def my_method # implementation in ZCL_MY_CLASS->BLUE_MY_METHOD
end

```

Experimental Libraries

Without providing too many details, it is worth listing some of the experimental libraries Blue Ruby includes for special purpose integration with ABAP-based applications and frameworks:

1. SAP Business Suite specific

- Atom ^[11] feed for Coghead: together with the SaaS vendor Coghead (Recently acquired by SAP), Blue Ruby was used to turn BAPI calls into data feeds according to the Atom Publishing Protocol specification that could be consumed as “linked applications” inside the Coghead platform – or in any standard Atom reader, e.g. the Mozilla Browser. Coghead then used these feeds to implement a collaborative sourcing application on top of SAP ERP.
- Floor Plan Manager (FPM) ^[12] scripting: A library that enables UI scripting inside the Floor Plan Manager UI framework based on Web Dynpro for ABAP. Using the FPM / Blue Ruby integration, developers can add dynamic behavior to FPM screens, i.e. dynamically change visibility or color of fields as well as calculate field values.
- POWL-to-Atom library: Prototype that shows how to consume SAP Business Suite Power Lists (aka POWL) ^[12] as Atom feeds provided by Blue Ruby.

2. SAP Business ByDesign specific

- Business Query Language (BQL): Blue Ruby can be used to build BQL statements and execute these BQL statements against the underlying Business Objects.
- Business Objects extensibility: Using Blue Ruby, custom behavior can be added to Business Object defined in the Enterprise Service Repository (ESR), without breaking platform consistency. As with the BAdI library, custom code and registration of the extension are contained in one simple Ruby code file.

Limitations

As Blue Ruby is an extension of the ABAP VM, a couple of compromises need to be considered. The compromises affect the compatibility to standard Ruby as well as performance at runtime. The execution environment for Blue Ruby programs is the ABAP VM, which completely shields developers from the underlying Operating System layer – therefore, access to OS functions is currently not supported in Blue Ruby. This imposes limitations w.r.t. Processes, IO, C-extensions to the Ruby language, etc.

Furthermore the current implementation of Blue Ruby does not offer Threads – although Ruby 1.8 uses green threads that are managed by the Ruby VM rather than native OS-threads, the current implementation of Blue Ruby does not consider multithreading – our experiments did not suggest we could implement Threads with reasonable performance in our current ‘ABAP-only’ approach.

Table 2 - Limitations of Blue Ruby

<u>Category</u>	<u>Class / Method</u>	<u>Reason / Comment</u>
<i>How to read this table</i>	<i>NotSupported NotStandard</i>	<i>Ruby features that are not available in Blue Ruby Ruby features that are significantly different in Blue Ruby</i>
Processes	Process Kernel#exec Kernel#system Kernel#syscall Kernel#` (backquote) ...	As the “OS” for Blue Ruby is the ABAP VM, real OS access is not possible. This imposes limitations w.r.t. Process handling, as the container for a Blue Ruby program is the ABAP work process
Threads	Thread ThreadGroup Continuation ...	Green threads are currently not implemented in Blue Ruby. Might be added at a later time
IO	Kernel#gets IO File Dir ...	Basic IO implementation available but significant differences as compared to standard Ruby due to limited IO availability of ABAP server (e.g. no STDIN)
Networking	Sockets TCPServer UnixServer Net::Telnet Net::FTP CGI Net::POP Net::SMTP Net::HTTP Mongrel / Webrick	No low-level networking libraries, as the ABAP VM shields developer from these layers. HTTP Server and Client available via ICM (Internet Communication Manager) – we try to keep these libraries close to Webrick / Net::HTTP, but differences exist.

Note: This list of limitation is not a comprehensive one – Blue Ruby is still under development and (as mentioned earlier) the current completion rate is ~70%. The presented list is shows conceptual limitations, not implementation status.

Outlook

As with most research, the goal of the Blue Ruby project is to introduce new technologies and concepts to SAP and its community. Blue Ruby is not an SAP product, but with your support, we hope to build a strong case for productization.

Please provide your comments and feedback on this article and the project via the SDN Wiki ^[13]. We will also make a test-drive environment available for interested parties – if you want to play with Blue Ruby, please contact us. Your input is very welcome.

We are convinced that Blue Ruby provides substantial value to you, the developer. It provides an easy way to build simple applications or extend existing SAP applications utilizing the Ruby programming language. It provides the “best of both worlds” – lightweight, loosely-coupled, agile programming via Ruby, executed within the robust, proven SAP Web Application Server for ABAP.

For more information and comments

For more information, to provide comments, and for video tutorials of Blue Ruby, please visit our wiki page on SDN: <https://wiki.sdn.sap.com/wiki/display/Research/BlueRuby>

Related Content

- [1] SAPs CTO Vishal Sikka on “Timeless Software”
<https://www.sdn.sap.com/irj/scn/go/portal/prtroot/docs/library/uuid/d03dcd88-078c-2b10-f398-b1639d6a6e65>
- [2] The Ruby programming language
<http://www.ruby-lang.org/>
- [3] TIOBE Programming Community Index
<http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html>
- [4] The JRuby project
<http://www.jruby.org/>
- [5] The IronRuby project
<http://www.ironruby.net/>
- [6] The Rubinius project
<http://www.rubini.us/>
- [7] Thomas Weiss on the architecture of SAP NetWeaver Application Server ABAP
<https://www.sdn.sap.com/irj/scn/weblogs?blog=/pub/wlg/6164>
- [8] RubyParser – a parser for the Ruby language, written in Ruby
<http://parsetree.rubyforge.org/>
- [9] The RubySpec project – executable specifications for the Ruby language
<http://www.rubyspec.org/>
- [10] ABAP Enhancements and Modification on SDN
<https://www.sdn.sap.com/irj/sdn/abap?rid=/webcontent/uuid/109f5161-ee76-2910-cb99-db10b559ef4b>
- [11] The Atom Syndication Format and Publishing Protocol
<http://www.atomenabled.org/>
- [12] Monika Ahrens, Marko Degenkolb, Miro Vins – SAP TechEd 2008 session on Simplified User Interfaces in SAP Business Suite
<https://www.sdn.sap.com/irj/scn/subscriptions/content?rid=/media/uuid/60ebb427-ac5f-2b10-4d80-b8c9e01f0a1c>
- [13] Blue Ruby Wiki on SDN
<https://wiki.sdn.sap.com/wiki/display/Research/BlueRuby>
- [14] Ruby & Ruby on Rails forum on SDN
<https://forums.sdn.sap.com/forum.jspa?forumID=221>

Copyright

© Copyright 2009 SAP AG. All rights reserved.

No part of this publication may be reproduced or transmitted in any form or for any purpose without the express permission of SAP AG. The information contained herein may be changed without prior notice.

Some software products marketed by SAP AG and its distributors contain proprietary software components of other software vendors.

Microsoft, Windows, Excel, Outlook, and PowerPoint are registered trademarks of Microsoft Corporation.

IBM, DB2, DB2 Universal Database, System i, System i5, System p, System p5, System x, System z, System z10, System z9, z10, z9, iSeries, pSeries, xSeries, zSeries, eServer, z/VM, z/OS, i5/OS, S/390, OS/390, OS/400, AS/400, S/390 Parallel Enterprise Server, PowerVM, Power Architecture, POWER6+, POWER6, POWER5+, POWER5, POWER, OpenPower, PowerPC, BatchPipes, BladeCenter, System Storage, GPFS, HACMP, RETAIN, DB2 Connect, RACF, Redbooks, OS/2, Parallel Sysplex, MVS/ESA, AIX, Intelligent Miner, WebSphere, Netfinity, Tivoli and Informix are trademarks or registered trademarks of IBM Corporation.

Linux is the registered trademark of Linus Torvalds in the U.S. and other countries.

Adobe, the Adobe logo, Acrobat, PostScript, and Reader are either trademarks or registered trademarks of Adobe Systems Incorporated in the United States and/or other countries.

Oracle is a registered trademark of Oracle Corporation.

UNIX, X/Open, OSF/1, and Motif are registered trademarks of the Open Group.

Citrix, ICA, Program Neighborhood, MetaFrame, WinFrame, VideoFrame, and MultiWin are trademarks or registered trademarks of Citrix Systems, Inc.

HTML, XML, XHTML and W3C are trademarks or registered trademarks of W3C®, World Wide Web Consortium, Massachusetts Institute of Technology.

Java is a registered trademark of Sun Microsystems, Inc.

JavaScript is a registered trademark of Sun Microsystems, Inc., used under license for technology invented and implemented by Netscape.

SAP, R/3, SAP NetWeaver, Duet, PartnerEdge, ByDesign, SAP Business ByDesign, and other SAP products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of SAP AG in Germany and other countries.

Business Objects and the Business Objects logo, BusinessObjects, Crystal Reports, Crystal Decisions, Web Intelligence, Xcelsius, and other Business Objects products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of Business Objects S.A. in the United States and in other countries. Business Objects is an SAP company.

All other product and service names mentioned are the trademarks of their respective companies. Data contained in this document serves informational purposes only. National product specifications may vary.

These materials are subject to change without notice. These materials are provided by SAP AG and its affiliated companies ("SAP Group") for informational purposes only, without representation or warranty of any kind, and SAP Group shall not be liable for errors or omissions with respect to the materials. The only warranties for SAP Group products and services are those that are set forth in the express warranty statements accompanying such products and services, if any. Nothing herein should be construed as constituting an additional warranty.