RSSBus™: A Really Simple Service Bus

A Service Bus?!

If the term *Service Bus* rhymes with confusion, you are not alone. Definitions abound, and almost every vendor provides their own twist on the subject. The core concepts however cut through the very core of every connected system.

Here is a good description from IBM:

> *"An enterprise service bus (ESB) is a pattern of middleware that unifies and connects services, applications and resources within a business. Put another way, it is the framework within which the capabilities of a business' applications are made available for reuse by other applications throughout the organization and beyond. The ESB is not a new software product - it's a new way of looking at how to integrate applications, coordinate resources and manipulate information."*[1]

This paper is about a service bus, a *really simple* one. What we will describe is a service bus based on *RSS,* or *Really Simple Syndication*: a simple protocol for disseminating news on the Internet which has been especially successful in the blogging world. RSS is a basic structure, or protocol for producing information feeds, usually consisting of news headlines. Many people will say that there is nothing *enterprise* about RSS, and indeed it is often assumed that RSS is *too simple* to be useful for anything but news.

We would like to challenge that assumption. No one can deny the tremendous success and popularity of RSS as a tool for information exchange. ESB is also about information exchange and if you read IBM's definition above and are familiar with RSS and World Wide Web concepts in general, you will see connection points between the two concepts: RSS and ESB. These connection points are not coincidental. In fact, we would venture so far as to claim that there is significant intersection between the two concepts.

With RSSBus, we present an open system and architecture that explores and exploits this intersection. RSSBus is an integrated framework of tools and services that allow you to easily produce and consume *service-like information feeds* (for lack of a better term).

[1] http://www-306.ibm.com/software/info1/websphere/index.jsp?tab=landings/esbbenefits

RSSBus makes it easy to create such services/feeds and connect them with existing and new applications, enabling you to easily create a simple but effective layer of middleware that drives your information flow.

A Service Bus for the REST[2] of us

It is a given nowadays that information is a core asset of every business, small or large. Connecting this information to the rest of the world through a number of produce-consume cycles is what drives a business. Yet frameworks that enable and exploit these connections are still quite complicated, expensive and normally reserved for large entities with substantial IT organizations.

With RSSBus, our goal is to offer a simple, easy alternative for the small organization with little to no IT assets, little to no professional development tools, and no professional programmers to use them.

If you spend much of your working day developing web services for one of the major service platforms, and you can no longer write a single sentence without a four, five, or six letter acronym in it, perhaps you should stop reading right here and spare yourself the disappointment.

And if your job title has "Architect" or "Enterprise" in it, then *please* stop reading here. Just in case you didn't notice, we said *Service Bus*, and not *Enterprise Service Bus*. In fact, this is the last time we will mention *Enterprise*[3].

What we are building is something different, a service platform for the rest of us, the non-acronym-speaking crowd. If you have bits of pieces of data that you would like to quickly exchange with and/or connect to other systems, if simplicity and ease of use is your most important consideration, please read on.

We are definitely not the first to try this. In fact, one could say that almost every developer that has ever built a system of any complexity has at one time or another architected a structure that resembles a service bus. However, specific projects are limited by time and resources, and the ability to build software of general utility is likewise limited by a multitude of constraints.

With RSSBus, our goal is to build general purpose software that connects or has the ability to easily connect to every system, data, or information source of any significance. Our core focus is to enable connectivity as simply and as easily as possible, and we believe our experience building networking software components and connectivity toolkits for the past decade, and the software assets we have created in the process, give us a unique advantage.

---

[2] Pun intended: i.e. the capitalization of "REST" is not a grammatical error. If you can't guess why, please read on to find out.

[3] We are not implying that RSSBus is not suitable for Enterprise installations, on the contrary we think RSSBus complements existing Enterprise infrastructure very well. But we want you to note that RSSBus is not designed to replace an Enterprise Service Bus: it foregoes some of the more esoteric features in favor of simplicity and ease of use.

In many ways, RSSBus is a direct continuation of our work of the past 10 years, and a conscious effort to build upon the knowledge we have gained about how developers think and what they expect from makers of reusable software building blocks.

Why RSS?

The success of RSS in the news/blogging world clearly demonstrates the importance of simplicity in information exchange mechanisms. Does this principle apply beyond headline news and weblogs? We believe it does, and we have spent thousands of man hours designing and building a framework of tools and services in an effort to prove that it does. Read the RSS specifications carefully, and you will see through the lines that the designers of the protocol clearly intended RSS to be used for more than just news.

In the context of RSSBus, we extend RSS with additional elements (*modules*, in RSS 2.0 terms) that provide structured information about RSS Items. RSS produces news items, but if you stop and think about it for a minute, items can describe anything: a customer in a database, a product in a shopping cart, a file in a directory, a book in a bookshelf, an email in a mailbox, a charge in a credit card statement, anything that can be described with text and attributes. In other words, in object-oriented terms, items are *objects* with *properties*.

We are using the same extension mechanisms described in the RSS 2.0 specification to produce *Rich RSS Content*: RSS information feeds consisting of RSS Items with a structured array of attributes. RSSBus is an extensive framework of tools that enables you to easily and quickly produce rich RSS content from your information/data sources and applications.

RSSBus offers a simple way to produce, access, aggregate, and more generally, compute with RSS Items. It provides tools to produce and consume RSS Feeds, a framework that simplifies management and manipulation of RSS Feeds and RSS Items, and tools for ubiquitous access to and from every platform and development environment. As in IBM's definition above, our goal is to enable you to *integrate applications, coordinate resources, and manipulate information*.

But what about Web Services?

The simple answer is: "*RSS Feeds* are *Web Services*". Whether that answer is acceptable to you or not depends of course on *your* definition of a Web Service.

If you see Web Services as strictly SOAP endpoints, defined by WSDL, discovered via UDDI, secured by WS-Security, etc.., well, we can remind you of what we said about acronyms in the beginning of this paper.

On the other hand, if you see Web Services simply as Web Locations (i.e. URLs) that provide you with information or take actions on your behalf based upon your input, then maybe the strict form or shape does not matter as much, and for most of us: the simpler the better.

Amazon, EBay, Google, Yahoo, and the rest of the *REST* crowd seem to agree, based on some of the reasons stated above, and a few more. REST, or Representational State Transfer, is a different way of looking at Web Services and Information Exchange. If you do a search on Google or Wikipedia for the term, you will find lots of information on the subject.

Casual (Haphazard) Application Integration - *CHAI?*

Normally, the integration space is reserved for large applications, large enterprises, large projects - normally - but that doesn't mean the other end of the market does not have a need for integration. Quite the contrary, but expense and complexity stand in the way. Even if expense is not an issue, complexity rears its ugly head, especially interface complexity. RSS is as simple as it can be. Lists of objects with attributes are a well known construct with enough real world parallels to make them extremely easy to grasp.

Connecting two applications together, creating a simple workflow, should be a simple drag and drop operation. Undoing or modifying that simple connection should be equally simple. Effecting change should be simple and straightforward, or our systems quickly get overwhelmed by the effects of change around us. We would even venture so far as to claim that change can and in certain situations must be *casual*.

Yes, this sounds like the opposite of EAI: Enterprise Application Integration, and if you have Enterprise in your job title, you are probably thinking this is a recipe for disaster. Indeed, we are talking more about *effecting* change than about the *effects* of change. In other words, you may think that this is anything but Safe[4], but maybe that's exactly why we asked you to stop reading above.

OK, so tell me what this really is!

Yes, enough of the general conceptual "fluff". Let's see in a bit more detail what we have built, and most importantly what you can build with what we provide.

We will start with an overview of the main constructs, and then move on to explore what you can do with them.

Items and Attributes

As in RSS, the concept of "*Item*" is central to RSSBus. We said that items are just objects with properties. In RSS, and in RSSBus, items are bags of attributes coming from one or more XML namespaces, which is more or less a way of saying the same thing with the main difference being that XML namespaces are used to categorize and distinguish

[4] As with REST above, the capitalization of "Safe" is not a spelling error: we would urge you to read a somewhat related piece by Dave Winer, creator of RSS titled "Checking in with Mr. Safe", always a source of inspiration and good cheer to the authors of this paper.

attributes. (If you don't know what XML namespaces are, don't worry: for all our intents and purposes they are just unique identifiers expressed as URLs).

Here is an example: if we have an item describing a customer, a set of attributes are going to specify his or her address, another set of attributes could provide their payment information, yet another set could provide information about their purchasing history. Depending on how we are looking at the customer, we consider one or another set of attributes. XML namespaces would categorize and distinguish these sets of attributes from each other.

Namespaces are associated with *prefixes* which provide a simpler way to refer to *namespace:attribute* pairs. Within RSSBus we make heavy use of prefixes as an easy way to refer to the namespaces they represent: item attributes are always referenced as "*prefix:name*" pairs.

The standard RSS 2.0 namespace with attributes such as title and description is always present. Other attributes belong to one or more XML namespaces that define the item. We reserve the special "*rss*" prefix for the RSS 2.0 namespace. This means that we refer to standard RSS attributes as *rss:title*, *rss:description*, *rss:date*, etc.

Feeds and Operations

Once you start looking at your data as RSS Items, you are only a short step away from making them available to the network as RSS Feeds.

We produce RSS Feeds through o*perations*, procedures with a predefined signature that can be configured to produce items of various types. The *RSSBus Engine* executes these procedures and serves the resulting feeds.

Operations are bundled in *connectors* which are libraries that implement one or more related operations. RSSBus comes prepackaged with an extensible set of connectors which provide operations for file system access, database operations, standard communication features such as email and file transfer, etc. The interface is open, and you can easily add your own custom connectors and operations.

The input to an operation (or feed) is a set of parameters, or attributes, normally provided through standard HTTP mechanisms such as the URL query string and/or form (www-url-encoded) POST data. The set of input attributes constitutes an *input item*. The operation creates a feed based on values provided as input.

Each operation is configured to receive input at one or more web locations (HTTP endpoints). The RSSBus Engine is responsible for reading the HTTP input, converting it into a form suitable for operation input, passing the input to the operation, invoking the operation logic, and then formatting the result as a standard RSS Feed.

Integrated Services and Workflows: Item In, Items Out

All RSSBus operations follow the same input-output pattern: *Item In, Items Out*. The HTTP input constitutes an *input item*. The resulting feed is nothing else but a collection of *output items* produced in sequence.

Since we are both producing and consuming items, we can think of sequential calls to operations as simple pipelining of the output of one operation to the input of another. Multiple feeds, from one or more location on the network can be pipelined to each other to create *workflows*.

RSSBus provides standard operations for manipulating, redirecting, and recombining the resulting feeds, coupled with a simple XML-based workflow scripting tool which we call *RSBScript* (described further below).

The RSSBus Engine

The current version of the RSSBus Engine is a Microsoft ASP.NET application that runs under Microsoft IIS or any other ASP.NET (2.0) compatible web server such as Microsoft IIS. The RSSBus Engine is used by the RSSBus Feed Server to process feed requests.

The core tasks of the engine include managing the internal collection of connectors that implement operations, providing serialization and deserialization services for the input and output items, returning usage (meta) information about installed operations, performing error checking, and managing the lifecycle of calls.

The RSSBus Engine is also responsible for executing workflows and pipelining operations to each other based on scripted instructions, as well as generating HTML content based on scripted templates (described in the next section).

RSSBus Desktop

RSSBus Desktop is a small web server that runs on your desktop and responds to feed requests. It routes these requests to a single user version of the RSSBus Engine, which is responsible for serving RSS feeds that correspond to these requests. Based on security settings you specify, it has the ability to interact with applications and data on the system.

RSSBus Desktop is available as a free download and it is our intention to distribute it freely as far and wide as we can. It's purpose is to provide access to the tips of the network removing the barriers towards publishing, participating, and integrating feeds.

RSBScript

RSBScript is an XML-based scripting language that allows you to easily call operations, manipulate their results, and produce feeds.

We have kept the number of language constructs to a minimum in order to make understanding and creating scripts as easy as possible. RSBScript is based on simple XML instructions such as *<rsb:call op=[url]>, <rsb:set attr=[name] value=[value]>, <rsb:check>,* which are easy to parse and manage though automated tools.

Most of the time RSBScript is used as a glue to produce feeds from operations by defining the right default values and calling an internal operation. In other occasions, results from remote operations (feeds) can be combined to produce other feeds, or change something in a local or remote system. The primary purpose of RSBScript is to tie operations and feeds together.

Moreover, XML workflow instructions in RSBScript can be interspersed among text which is generated as a result of the script. In this capacity, RSBScript serves as a *templating tool* converting RSS items to HTML, plain text, or other presentation formats. We call these scripts *RSBTemplates*. We make extensive use of RSBTemplates for creating web pages that display results of RSS feeds.

Other Languages

The RSSBus Engine enables integration and programmability with other languages in two basic levels:

The first is the obvious support for other languages provided by Microsoft .NET. You can write a connector in any .NET-supported language. RSSBus won't know the difference.

The second option is the language provider framework in RSBScript. A language provider is a special operation that interprets script content and then executes it, providing the results as a feed. The script is include within a special <rsb:script> tag in RSBScript.

Writing a language provider is a relatively simple task. We illustrate this with the Python provider included in the RSSBus package.

Bundled Connectors

RSSBus comes with an extensible set of connectors that implement an array of reusable operations which can be called as is, or combined with your own logic to produce feeds.

Here is an alphabetical list of some of the connectors included in the product:

- *CsvOps*: operations for managing delimited record files.
- *EncOps:* operations for encoding/decoding data using various formats.
- *ExcelOps*: operations that read and write data to Excel spreadsheets.
- *FileOps*: operations for managing files and directories.
- *FtpOps*: operations for transferring files to and from FTP servers.
- *GcalOps*: operations that provide access to Google calendar.
- *GsheetOps*: operations that provide access to Google spreadsheets.
- *GtalkOps*: operations that provide access to Google talk services.
- *ImapOps*: operations for receiving email messages from IMAP mail servers.
- *ImOps*: operations for instant messaging (Jabber, SMS, etc.).

- *LdapOps*: operations for connecting to LDAP directories.
- *MediaOps:* operations that provide information about digital media files.
- *NntpOps:* operations for reading and posting newsgroup articles.
- *OfxOps*: operations for accessing bank accounts and financial services.
- *OracleOps*: operations for accessing Oracle databases.
- *PaypalOps*: operations that provide access to PayPal payment services.
- *PopOps*: operations for receiving email messages from POP servers.
- *SmtpOps*: operations for sending SMTP email.
- *SqlOps*: operations for connecting to SQL Server databases.
- *SysOps*: operations for system management (processes, memory, etc..).
- *TsvOps*: operations for managing simple tab-separated file databases.
- *XmlOps*: operations that process XML files.

The list above is just a snapshot, and we expect to continuously update it with new connectors and operations. We certainly hope and expect the community (*i.e. You*) will contribute new connectors, new capabilities, new feeds and services, and publish them on our online directory of connectors, feeds, and operations at www.rssbus.com.

Custom Connectors / Operations

The RSSBus Engine provides an extensible architecture for integrating additional services. Built on top of the .NET Framework, the RSSBus engine can be customized and extended with additional connectors written in any .NET compatible language.

As we mentioned earlier, a connector consists of a set of operations. An operation is a class that implements a simple interface: *RSBOperation*. The interface has two methods: *Info* and *Exec*.

The *Info* method provides (XML) information about the operation: a textual description of what the operation does, a list of its arguments and their descriptions, default values for arguments, output attributes and their descriptions, the associated namespace(s), etc.

The *Exec* method is what is called to execute the operation. It takes a single argument of type *RSBContext* which provides information about the context in which the operation is executing: input arguments, output locations, etc.

The operation provides its output by *pushing* items to its execution context. The items are then either used internally by another operation or script inside the local engine, or serialized to the outbound HTTP stream and sent to a remote client. The important thing to note here is that the operation itself is completely agnostic of its execution context: i.e. it does not need to know, and in fact it has no way of knowing what happens with the information it serves, how it's consumed, how it is processed, how it is aggregated or integrated.

RSSTools: Client Components

RSSTools is a set of development components that accompanies the RSSBus Engine. It includes a full array of client access components for all major platforms and development environments. These components are useful for use within applications other than the RSSBus framework.

Supported platforms include: Microsoft .NET and .NET Mobile, Java, Microsoft COM (ActiveX and ASP), C Library and C++ Classes for Windows and Unix/Linux, including Mac OS, Borland Delphi VCLs, PHP, etc..

RSSTools: Server Components

In addition to client components, the RSSTools package provides a set of embedded servers that allow you to serve RSS content from any application.

The server components are embedded web servers that run within the context of your application or device and translate RSSBus operation calls to events within your application. You intercept these calls and provide output to the component as a set of RSS items. The component then serializes these items into an RSS feed which gets served to the client.

The list of platforms and development environments (editions) for the embedded server components is identical to that of the client components described above.

So what exactly do I do with this?

The benefits derived from a service bus architecture are directly related to the number of services available and the ability to integrate with them. The relationship is exponential: benefits grow exponentially with every new service added to the bus. This is a direct application of what has come to be known as Metcalfe's Law:

"*The power of the network increases exponentially*[5] *by the number of computers connected to it. Therefore, every computer added to the network both uses it as a resource while adding resources in a spiral of increasing value and choice.*"

Bob Metcalfe, Inventor of the Ethernet

---

[5] Some people say Metcalfe described the relationship as "quadratic", some say "exponential". Does it really matter? What matters is that any relationship above linear in substantial numbers packs an awesome amount of power.

With RSSBus we provide tools that allow you to quickly add and access services through a simple, standard, widely accepted protocol. The power of the resulting system depends on the number and type of services you and others expose, and the extent to which they are employed.

What follows is a five step process towards building a simple service bus around your data and information. You don't necessarily have to follow these steps in order, in fact you may enter the process in any of the steps, depending on what has already been built by you or others.

Start by creating Feeds: Publish

Think about your information as collections of items with attributes. This should come naturally: we normally think about the world around us as a universe of items, classified by their attributes. Items of similar structure can be assembled in feeds and served at predefined web locations.

Think about the most important pieces of information in your business and how you can access them. Look at what information your business partners provide that you care about, and put an RSS interface in front of them. Check the online directory at www.rssbus.com to see whether someone provides a connector for accessing the data or system you care about. In most cases, you can simply plug the connector into the bus and serve away. If not, you may have to build your own connector. If you do, consider sharing it with others.

Most of the data today exist in relational databases. We provide standard connectors that allow you to directly expose relational data as feeds. The mapping is more or less direct: RSS items and relational database records are very similar. One or more feeds can be directly associated with every SQL table, view, or query. RSSBus makes this process quick and painless.

If you follow this process for any amount of time, you will soon end up with a number of feeds providing various types of information. Look at these feeds with a critical eye and try to classify and separate the important ones from the rest: identify the ones that are at the core of your business, the ones that expose information likely to used and reused over and over again by you or others. We like to call these *"services of record"*: feeds you should keep track of and carefully maintain over time.

Use Feeds, yours or others: Subscribe

Now that you have your feeds, start consuming them. Use the client components to get to them, consider making applications go to the feed instead of directly to the database, or whatever was providing access to the data before. Look at your business partners for services they already expose, or need to expose. Recognize opportunities to streamline existing processes, find new efficient ways to deal with your partners, customers and other stakeholders, and you will soon see the power of standard formats.

In some cases this may seem as redundant and useless work for applications that have been already written, and that very well may be the case, nonetheless, consider architecting your applications around *services of record*, services you know you will maintain and take care of over time. Once you do that, it doesn't matter if the data or processes change, be that for technical reasons or business reasons. Change is inevitable, but as long as services of record keep running, your applications, the software that makes your business tick, will keep on running.

Tell others about your Feeds: Advertise

There are obvious benefits in communicating with your customers and partners in standard ways. The easier you make it for them to access the information they care about, the more empowered will they be to work with you and focus on the fundamental value you, and your common relationship provides.

Open the floodgates of information, and they will figure out how to best work with you. You can't anticipate everything they will build, nor should you try to. What's important is that they are able to work with you. Market forces will sort out the rest, and very likely bring one or more positive surprises in the process.

Amazon is a great example of opening up to others and allowing them to innovate around their services. You may not be Amazon, but if you have customers and partners, you definitely have things they care about. Think about empowering them and simplifying communication.

If you already have the feeds they care about, give them access, let them come in. Remember, feeds are just URLs: scribble them on the back of envelopes, email them, publish them on your site, or publish them in our online directory.

Build and they will come.

Use Feeds in your applications: Integrate

Once you have created feeds that provide access to your data, or identified third party services offered as feeds, you can immediately use them in any of your existing applications as a way to access information or services.

The RSSBus client components allow you to effortlessly access feeds from any major platform and/or development environment. Use these components to integrate your existing applications with service feeds. This will create a layer of indirection between your applications and data, a layer that will most certainly prove invaluable if your data or your systems change.

At the same time, look at your existing applications and their needs to expose data and services. The RSSBus server components allow you to generate feeds from a variety of systems and platforms. You can use these components to provide access to your applications from inside or outside of your network.

Make the Bus work for you: Automate

Once a service infrastructure has been created, the drive to automate is a natural progression. We described ways to both provide and access services from any major platform or development environment. This means that you can use any tools you are familiar with, or any environment you are proficient in for automation purposes.

In addition, we provide RSBScript as a dedicated automation tool for RSSBus. It is based in a very simple set of XML instructions, and it is targeted specifically towards automating RSSBus services. The learning curve is small and the concepts it is based on are simple and straightforward. Access is also simple: save and run, no compilation process, no build cycle.

RSSBus services take items or feeds as input, and produce feeds as output. Connecting the output of one service to the input of another is easy and straightforward. Writing operations that operate on feeds, or using the existing feed operations, is also easy and straightforward.

Conclusion

We hope this short overview has sparked your interest in simple RSS-based web services in general, and RSSBus in particular. We invite you to download the free RSSBus software from www.rssbus.com and experience it first hand.

Our goal is to make RSSBus the best tool available for creating simple, lightweight service architectures. This is the first release of the product, and we have a long roadmap ahead of us. We realize that we can't walk that road alone. Let us know what you think, tell us about your experiences, and most importantly let us know how you would like to see the product evolve.

If you made it to this point, chances are we are already within your thoughts. We are thankful and delighted at that possibility, and once more we wholeheartedly invite you to explore, open doors, and connect!