

Radical Blue Gaming

Incredibly Innovative Gaming Solutions

RadBlue's S2S Quick Start Package (RQS) Developer's Guide

Version 0.1

www.radblue.com

April 17, 2007

Trademarks and Copyright

Copyright © 2007 Radical Blue Gaming, Inc. (RadBlue). All rights reserved. All trademarks used within this document are the property of their respective owners.

Gaming Standards Association (GSA) is a trademark of the Gaming Standards Association of Fremont, California.

Contact Information

Radical Blue Gaming, Inc.
3000 West Plumb Lane
Reno, Nevada 89509

Phone: +1 (775) 329-0990

E-mail: sales@radblue.com

WWW: <http://www.radblue.com>

Forum: <http://radblue.mywowbb.com>

Contacting GSA

If you need a copy of GSA's latest S2S protocol documents, or to find out more about the Gaming Standards Association and the work being done by over 70 companies in the areas of protocol standardization for the gaming industry, we encourage you to discover more about the organization at www.gamingstandards.com, or to contact them directly via e-mail (sec@gamingstandards.com) or by phone at +1 (510) 609-4007.

The most current version of S2S can be found here:

<http://www.gamingstandards.com/standards.html>

Document Modification History

Doc #	Date	Description
0.1	17 Apr 2007	Updated early docs to make first stand-alone version

Table of Contents

Overview.....	3
Referenced Resources.....	3
JAVA PLATFORM	3
ECLIPSE IDE	3
XMLBEANS	3
AXIS SOAP STACK	3
JETTY WEB SERVER	3
APACHE'S TOMCAT WEB SERVER	3
Reference Diagrams.....	4
DEVELOPMENT CONFIGURATION	4
PRODUCTION CONFIGURATION	5
Developer Machine Requirements.....	6
Getting started with RQS.....	7
(1) INSTALL JAVA	7
(2) INSTALL ECLIPSE	7
(3) START ECLIPSE	7
(4) IMPORT THE RQS INTO ECLIPSE	8
Working with RQS.....	9
DIRECTORIES OF THE RQS SOURCE CODE	9
UNIT TESTS	9
THE RQS IS A STACK	10
HOW DO I RUN THE EDGE SYSTEM DEMO PROGRAM?	10
MAIN COMPONENTS OF RQS	11
How an inbound message is processed by the Edge System.....	11
WEB CONTAINER	11
XML LAYER	12
DISPATCHING LAYER	12
EDGESYSTEM CLASS	12
How an outbound message is processed by the Edge System.....	13
API LAYER	13
TRANSPORT LAYER	13
SOAP LAYER	13
Adding new classes and commands to the RQS.....	14
JAXB 2.0	15
Distributing your code.....	16
CREATING A WAR FILE FOR THE HOST SYSTEM	16
CREATE A ZIP FILE FOR THE HOST SYSTEM	16

Overview

The RadBlue S2S Quick Start (RQS) SDK is intended for programmers who wish to implement GSA's System to System (S2S) protocol in their applications.

The RQS SDK will allow you to quickly create and extend either side of an S2S conversion. By making use of open standards and open source modules, a programmer does not have to learn all the low level details in order to send and receive S2S compliant messages.

Referenced Resources

The following are sources for the resources referenced in this document:

JAVA Platform

The SDK makes use of the JAVA 6 platform. You can download the Java Development Kit (JDK) directly from Sun Microsystems at the following site: <http://java.sun.com/>

Eclipse IDE

The RQS SDK is supplied as an Eclipse project that can be imported into the Eclipse Integrated Development Environment (IDE). It was developed using Eclipse 3.2, whci you can get here: <http://www.eclipse.org/>

XMLBeans

XMLBeans is used to take the S2S XSD files furnished by GSA and compile them into Plain Old Java Objects (POJO). Developers can then use the resulting POJOs to get and set values in the S2S data model. XML Beans is available here: <http://xmlbeans.apache.org/>

AXIS SOAP Stack

Apache Axis is an implementation of a SOAP ("Simple Object Access Protocol") stack that we find useful. It's available here: <http://ws.apache.org/axis/>

JETTY Web Server

Jetty is an open-source, standards-based, full-featured web server implemented entirely in JAVA. It's great for developing your implementation, and is also used as the web container when RQS is running inside of Eclipse. We use JETTY very successfully in our Scriptable Tester (RST) and G2S Scope (RGS) products. JETTY is available here: <http://www.mortbay.org/> .

Apache's Tomcat Web Server

When you need a production oriented, high performance web server, Tomcat is a popular choice. We include build files in RQS to create WAR files for Tomcat for your new project. Tomcat is available here: <http://tomcat.apache.org/> .

Reference Diagrams

The following diagrams should prove valuable as you read through the balance of this document:

Development Configuration

The following is a view of your JAVA application running in a development (JETTY) environment:

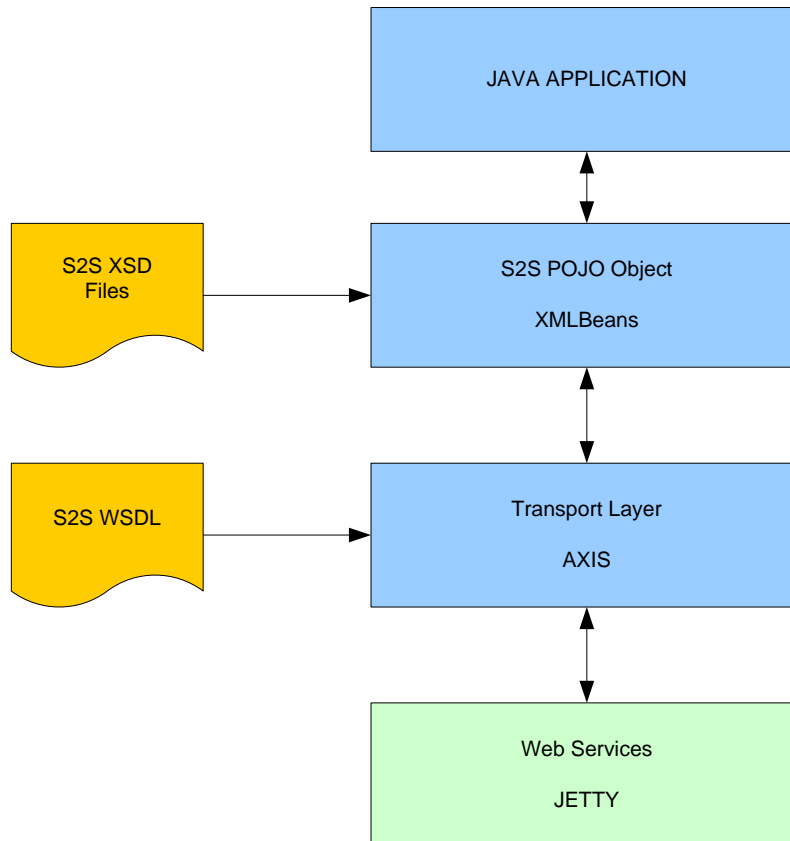


Diagram 1 – Development Configuration

Production Configuration

The following a view of your JAVA application running in a production environment with Apache's Tomcat web server:

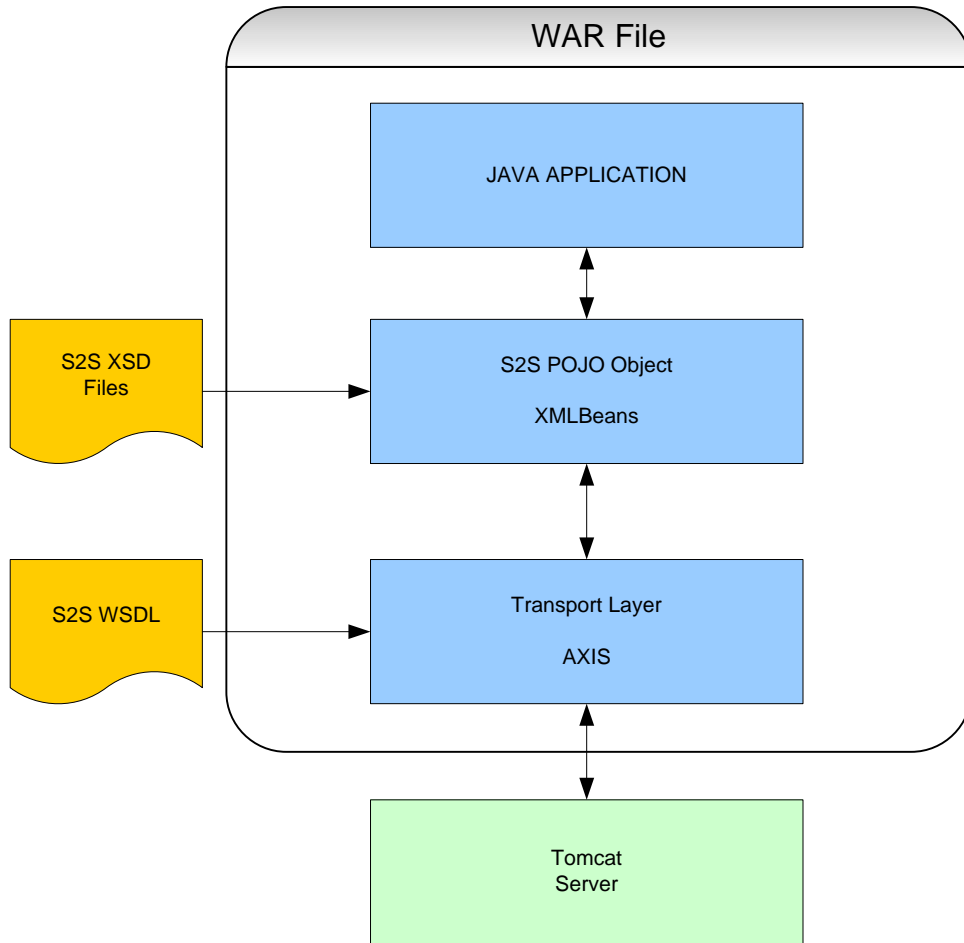


Diagram 2 – Production Setup

Developer Machine Requirements

The following are the minimal requirements for any development machines that will be running the S2S Quick Start SDK.

Intel 2.8 Ghz machine or comparable
1.5 GB Memory
80 GB HD
Windows XP

Java JDK 6 Platform <http://java.sun.com/>

Eclipse 3.2 <http://www.eclipse.org/>

Tomcat <http://tomcat.apache.org/>

Getting started with RQS

(1) Install Java

The first step is to install Java. You need to get Java 6 from the Sun web site. Use the installer to install Java 6 anywhere on your computer. Just follow the instructions in the installer.

To ensure you have the correct version of Java 6 you should, from the command line, run the command:

```
java -version
```

Your JVM should report:

```
java version "1.6.0"  
Java(TM) SE Runtime Environment (build 1.6.0-b105)  
Java HotSpot(TM) Client VM (build 1.6.0-b105, mixed mode)
```

If you get an error like "Command not found" it means that Java 6 is not on your path. Simply navigate to the bin directory of the Java 6 installation and run the same command.

Using the appropriate version of Java 6 is key. The RQS uses JAXB 2.0 for the marshalling and unmarshalling of S2S commands. JAXB 2.0 was unstable until the final public release of Java 6. So if you are using a pre-release of Java 6 you will not be able to run the RQS.

(2) Install Eclipse

Once you have Java 6 installed you will need to install Eclipse. Get the Eclipse 3.2.x (Callisto) distribution. It comes as a single zip file which you unzip on your computer. That's all there is to it.

(3) Start Eclipse

You can start the Eclipse IDE by navigating to the directory where you installed Eclipse and running the `eclipse.exe` executable. The first time you start Eclipse it will prompt you for a working directory. We usually pick `C:\workspace` but any directory will be acceptable.

Once you've fired up Eclipse you want to check that you are using the newly installed Java 6 JDK with Eclipse. Within Eclipse, navigate to

```
Windows -> Preferences -> Java -> Installed JREs
```

and double check that the Java 6 you installed is in the list. If it is not hit the **Add...** button and follow the instructions to register you new Java 6 installation. If you forget to do this step and try to compile the RQS with a different Java version you will get thousands of compiler.

Eclipse is an excellent IDE and one that we have standardized on. If you are new to Eclipse we would suggest the excellent book "Eclipse in Action" by David Gallardo.

(4) Import the RQS into Eclipse

You will receive the RQS zip package from Radical Blue Gaming. The zip file contains the entire source code for the RQS SDK, including unit tests, Java source code, ant scripts and the S2S schema files. The directory structure in the zip is laid out in a manner that will be acceptable to the Eclipse IDE. You can also work with the same source code tree outside of Eclipse if you are so inclined.

To import the RQS source into Eclipse, straight from the Zip, do the following:

File -> Import

From the menu double click on:

General -> Archive File

Hit the **Browse** button to locate the RQS zip file. Once you've selected the Zip file Eclipse reads through the file and finds the RQS project info. You should see the **Info folder** field update to hold the value **RadBlueS2SQuickStart-HEAD** which is the Eclipse project name that we use.

Hit the **Finish** eclipse will import the RQS into your workspace directory and make it a full fledged project. Eclipse will then compile all of the Java source code and if everything is configured correctly, after a few minutes, you will have successfully imported the RQS.

Working with RQS

Directories of the RQS Source Code

The root of the RQS project has the following directories:

- **bin** – this is where Eclipse puts the class files. If you are using a source code control system you would exclude this directory.
- **schemas** – this is where the S2S schemas are kept. The RQS currently ships with S2S 1.2.0 (the base version of S2S released by GSA). The RQS reads the schema file at run time so it is always required.
- **src** – this is where the RQS source code resides
- **tests** – this is where the RQS unit tests reside
- **vendor** – this is where all of the third party JAR files reside. This is also where the JAXB generated JAR files reside.
- **vendorsrc** – this is where the source code of some of the third party packages reside. Also the source code generated by JAXB is found in here.
- **webapp** – this is where the web container specific files reside. When running Jetty inside of Eclipse the RQS uses this directory to load the proper servlets and filters for Jetty.

Unit Tests

Radical Blue Gaming is a proponent of Extreme Programming and the practice of writing unit tests. The RQS is no different. At the present time there are over 250 unit tests which exercise quite a bit of the RQS. We encourage you to continue this practice as you add code to the RQS.

You can run the full suite of RQS unit tests by executing the following JUnit test:

```
com.radblue.s2s.testsuites.TestMaster
```

The easiest way to invoke the master test suite from within Eclipse is to open the TestMaster Java source file and then hit:

```
Run -> Run As -> JUnit Test
```

This will bring up the JUnit view and start the progress bar. The unit test generates a lot of output which is useful if a test fails. Once the test is finished check out the JUnit View and make sure you got a green bar.

The RQS is a Stack

The idea behind the RQS is to provide an API to S2S such that anyone can write an S2S Edge System or an S2S Host System. To this end your application should interact with less than 10% of the RQS (see below).

To meet this goal the RQS is written as a vertical stack. It looks something like this:

Your Application Logic
Edge System API
Edge System
XML Stack
SOAP Stack
HTTP Stack

The layers in gray are provided by the RQS. Your layer, at the very top, uses the API layer to invoke the services of the RQS. If you find yourself interacting with any layer beside the API layer you should contact RadBlue before wading into the mud...

How do I run the Edge System Demo Program?

The RQS comes with a stand alone demo program for the S2S Edge System. It uses Jetty as the web container and is suitable for running directly from within Eclipse. You can find it at:

```
com.radblue.s2s.quickstart.edgesystem.engine.EdgeSystemEngineMain
```

If you run this you will start up the RQS in Edge System mode. It will try to contact an S2S Central Host. The program takes a few arguments:

- **--from-system** – this is the from URL of the S2S Edge System
- **--host-url** – this is the URL of the S2S Central Host you are hoping to connect to (or it could be the URL of an instance of RST)
- **--timeout** – defines how long, in milliseconds, the S2S Edge System should wait while initially connecting to the S2S Central Host system.
- **--bind-address** – the IP address for Jetty to bind against. If you don't provide this argument then the RQS will bind against `localhost:28752`.

For example, here are the command line arguments you would use start the demo to talk to the RST running the S2S central host script:

```
--from-system http://localhost:28752/RQS/api-services/S2SAPI
--host-url http://localhost:28749/RST/api-services/S2SAPI
```

If you run this command from within Eclipse and have the RST running you will see the demo program go through a series of S2S voucher commands. This demo program uses the asynchronous high level API provided by the Edge System Engine.

Main Components of RQS

The main components of the RQS are the EdgeSystem and the HostSystem.

The EdgeSystem is the core implementation of an S2S Edge System. It manages all of the basic S2S tasks required of an S2S Edge System. The EdgeSystem is the main integration point where your business logic and S2S meet. If everything is done properly your Edge System application need only work with the EdgeSystem. Everything else in the RQS should be hidden from you.

The HostSystem is the core implementation of an S2S Central Host. It managers all of the basic S2S takes required of an S2S Central Host. The HostSystem is the main integration point where your business logic and S2S meet. If everything is done properly your Central Host application need only work with the HostSystem. Everything else in the RQS should be hidden from you.

How an inbound message is processed by the Edge System

The following section shows the path an inbound S2S message follows as it works its way through the Edge System.

Web Container

The web container, either explicit via Jetty or implicit via Tomcat, houses the HTTP and SOAP stacks. SOAP Messages that arrive at the Edge System first encounter the HTTP stack and are handed upwards to the SOAP stack. The Axis SOAP stack removes the SOAP wrapper and via the web server configuration file maps the URL of the SOAP request to the Edge System service implementation:

```
com.radblue.s2s.quickstart.soap.services.s2s.S2SGSAServiceImpl
```

This is the class that is responsible for implementing the logic behind the S2S WSDL file. The transcript is notified, S2S ACK is assembled, the S2S message is parsed and validated and sent upward towards the XML layer and then the S2S ACK is sent back as the result of the SOAP call.

XML Layer

The SOAP message contains an S2S message in string format. The S2S document factory class is responsible for parsing and validating the S2S message and turning it into a first class Java object. You can find the factory here:

```
com.radblue.s2s.core.types.document.S2SDocumentFactory
```

The factory returns a wrapper around the DOM representation of the validated S2S message. This wrapper is used by layers higher in the application stack to access command information as needed.

Once a message has been converted by the factory it is passed upwards to the dispatching layer.

Dispatching Layer

The dispatching layer is responsible for dispatching newly arriving S2S commands to their appropriate layer above. This is often called the Controller pattern. The dispatcher is found via the Dispatcher Factory:

```
com.radblue.s2s.quickstart.disatpch.DispatchManagerFactory
```

The EdgeSystem is a client of the Dispatching Layer. You may find other uses for being a client of the Dispatching layer. In particular, performance tracking and debugging often benefits from tapping into the stream of inbound messages.

EdgeSystem Class

The EdgeSystem class is the core of the Edge System implementation. You can find it here:

```
com.radblue.s2s.quickstart.edgesystem.EdgeSystem
```

EdgeSystem is the home of the core application logic for the Edge System implementation. It handles all of the mundane tasks for routing inbound commands and sending outbound commands.

The EdgeSystem also has a listener mechanism so that additional business logic can be added. The Edge System Demo program is a client of the EdgeSystem. As commands come into the EdgeSystem the Edge System Demo is notified and takes additional actions. This is probably the best way to implement your business logic on top of the ROS Edge System. By using the EdgeSystem listener infrastructure you isolate yourself from most of the EdgeSystem itself.

How an outbound message is processed by the Edge System

API Layer

The EdgeSystem has an API that maps methods directly to S2S commands. So each API method takes all of the data necessary to complete the command. The application logic invokes the API method with the right arguments.

The API implementation takes the arguments and creates a command instance for the appropriate command passing in all of the given arguments. The result of this is an AbstractS2SCommand. The command is then passed to the transport layer.

Transport Layer

The transport layer is handled by the Transport Factory:

```
com.radblue.s2s.quickstart.transport.TransportManagerFactory
```

The transport layer is responsible for handling all outbound communication and for updating the transcript. The transport layer determines out where the message should be sent and then converts the AbstractSSCommand instance to a string form which can then be passed to the SOAP layer.

SOAP Layer

The SOAP layer uses Axis to invoke the remote method in the far S2S end. Axis uses the S2S WSDL to properly marshall the S2S command and the S2S ACK. Because Axis uses HTTP Commands Client the SOAP Layer bypasses the web container and makes a direct connection to the far end.

Adding new classes and commands to the RQS

It is a relatively straight forward operation to add new class and command support to the RQS.

First, you must encode all of the commands for the class. These belong in the package:

```
com.radblue.s2s.common.commands
```

There is one package per class. The RQS currently supports communications and voucher. All commands ultimately extend the command base class:

```
com.radblue.s2s.common.commands.AbstractS2SCommand
```

It is important to write unit tests for all new commands. Follow the pattern used for the Communications and Voucher commands.

Once you have the commands completed you need to update either EdgeSystem or HostSystem. Both use reflection to dispatch a newly arrived command. So in either class you need to add a new method that takes a single argument, InterfaceS2SDocument. The name of the document is the canonical name of the command without any punctuation. If you look in EdgeSystem you will see the following method:

```
public void voucherVoucherConfig(InterfaceS2SDocument document);
```

This is the method that processes all voucher.voucherConfig commands. If you follow this pattern by adding methods for each new inbound command you will find that the dispatching method works without any modification.

When you add the method you should copy what the current voucher methods:

- 1) convert the document to an actual first class command instance and then
- 2) invoke the listener

This allows the business logic listening above to receive the command and take the appropriate action.

To update the EdgeSystem listener you need to add the new command to the listener interface:

```
com.radblue.s2s.quickstart.edgesystem.InterfaceEdgeSystem$InterfaceEdgeSystemListener
```

To update the HostSystem listener you need to add the new command to the listener interface:

```
com.radblue.s2s.quickstart.hostsystem.InterfaceHostSystem$InterfaceHostSystemListener
```

Just follow the already established pattern in the listener interface. Any classes that implement `InterfaceEdgeSystemListener` or `InterfaceHostSystemListener` can now hear about the new command.

It really is that simple. Once you get the hang of it you can add a new command in 15 minutes or less.

Adding an outbound command is basically the same. Both the `EdgeSystem` and the `HostSystem` have an API that they expose via an interface. The interface methods map directly to S2S commands. To add a new outbound command just update the interface and update the `EdgeSystem` or `HostSystem` API instances to create the command then hand the command off to the transport layer.

JAXB 2.0

The RQS uses JAXB 2.0 to convert the S2S Schema file (XSD) to Plain Old Java Objects (POJO). JAXB is very powerful and saves a great deal of typing on our part.

To use JAXB in the RQS we first had to convert the schema to source code which would generate the POJOs. This is done via the `xjc` compiler that comes with Java 6. If you look in:

```
src/build.xml
```

you will find an ant build script which will rebuild the POJOs. You would only need to do this if you made changes to the S2S schema or you were trying to upgrade the schema to a new namespace.

Distributing your code

Creating a WAR File for the Host System

The RQS Host System can be deployed as a WAR file. To create the WAR file simply execute the **host-war** task in the ant build script:

```
src/build.xml
```

This generates a WAR file which be dropped into any JSEE 2.3 compliant container. When the container loads the WAR file the RQS Host System starts automatically.

Create a ZIP File for the Host System

The RQS Host System can be deployed as a standalone application using Jetty as the web container. To create a Zip file of the entire RQS Host System simply execute the **host-zip** task in the ant build script:

```
src/build.xml
```

This generates a Zip file which can be unzipped anywhere you have Java 6 already installed. The Zip includes the Windows BAT script:

```
bin/rgs-host-system.bat
```

which invokes Jetty which invokes the RQS Host System.