Quick Start for G2S Implementation Guide

Table of Contents

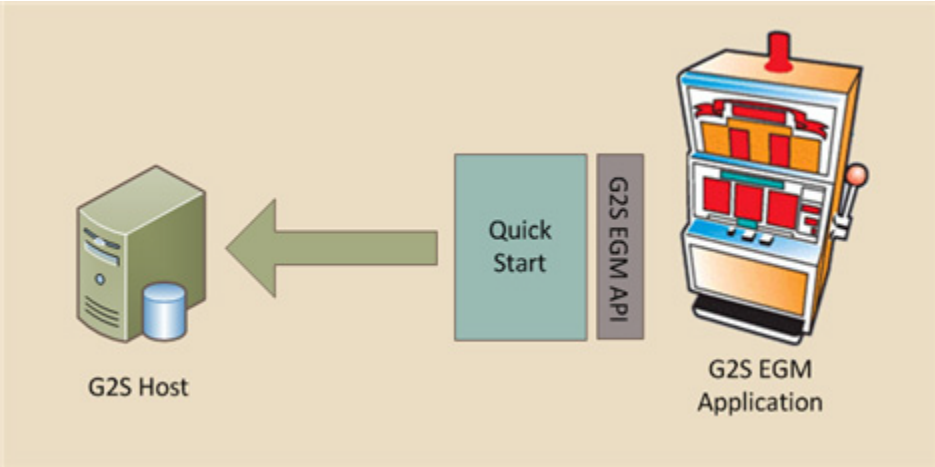
Chapter 1 ..... **2**
About Quick Start for G2S 2
Supported Commands 3
Quick Start Implementation 3
Quick Start Deployment 4
Resources 4
Development Configuration 5
Quick Start for G2S Stack 6
Chapter 2 ..... **7**
Message Processing by the EGM and G2S Host 7
Outbound Message Processing 7
Outbound Command Flow 8
Inbound Message Processing 9
Inbound Command Flow 11
Class Sequence Flows 12
Chapter 3 ..... **16**
Getting Started 16
Import Quick Start into Eclipse 16
Quick Start Directories 18
Chapter 4 ..... **20**
Run the G2S Host Demonstration Program 20
Connecting throught the RadBlue System Tester 20
Run the G2S EGM Demonstration Program 21
Connecting to RadBlue G2S Scope 22
Running Unit Tests 23

Chapter 1

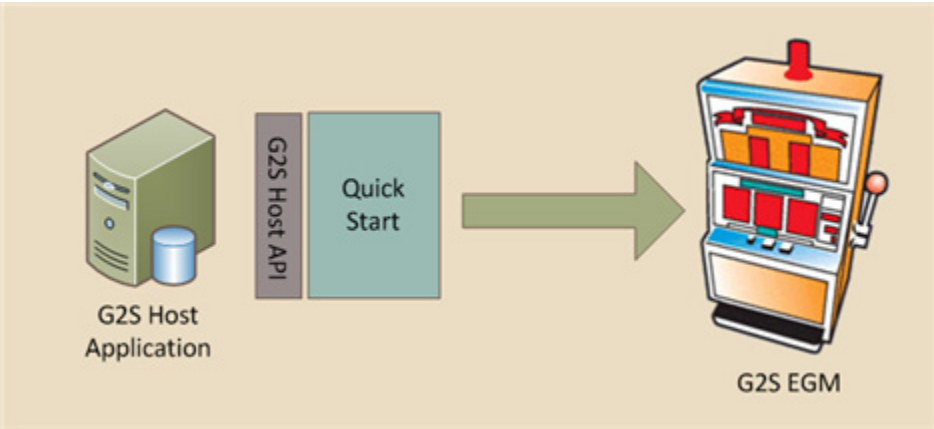
About Quick Start for G2S

The Quick Start Software Development Kit for G2S is intended for programmers who want to implement the GSA Game-to-System (G2S) protocol in their applications.

The Quick Start SDK allows you to quickly implement either side of a G2S conversation - EGM or Host. Through open standards and open-source modules, a programmer does not need to learn all of the low-level G2S and web service details in order to send and receive G2S-compliant messages - Quick Start for G2S gets you started and helps you learn to implement G2S on your own.



By making use of open standards and open-source modules, a programmer does not need to learn all of the low-level details in order to send and receive G2S-compliant messages.



Developers who are new to G2S can use Quick Start to quickly build a G2S application. The low-level G2S work is already complete (through Quick Start), allowing the developer to focus on connecting in the business logic. If an application already has existing business logic and databases completed, using Quick Start gets you approximately 70 percent of the way toward completing an G2S implementation of these classes in your product.

Supported Commands

Quick Start for G2S provides a limited number of commands in the Communications and Cabinet classes as examples to help you understand how Quick Start works. These commands have been selected specifically to allow programmers to begin communicating in G2S. The supported commands include:

Class	Request	(Initiatator)	Response	(Responder)
Communications	commsOnLine	(EGM)	commsOnlineAck	(Host)
Communications	commsDisabled	(EGM)	commsDisabledAck	(Host)
Communications	commsClosing	(EGM)	commsClosingAck	(Host)
Communications	setCommsState	(HOST)	commsStatus	(EGM)
Communications	getCommsState	(HOST)	commsStatus	(EGM)
Communications	getDescriptor	(HOST)	descriptorList	(EGM)
Communications	setKeepAlive	(HOST)	setKeepAliveAck	(EGM)
Communications	keepAlive	(Either)	keepAliveAck	(Either)
Cabinet	getCabinetStatus	(HOST)	cabinetStatus	(EGM)
Cabinet	setCabinetState	(HOST)	cabinetStatus	(EGM)

Quick Start Implementation

There are two implementations of Quick Start for G2S: the G2S EGM and the G2S host. Plain Old Java Objects (POJOs) for all of the supported G2S commands have been implemented for both the G2S EGM and G2S Host.

The G2S EGM implementation is an example of a G2S-based EGM. It manages all of the basic G2S tasks required of a G2S EGM. The G2S EGM implementation is the main integration point between your business logic and G2S. If everything is done properly, your EGM application need only interacts with the G2S EGM implementation. Everything else in G2S should be hidden from you.

The G2S host implementation is an example of a G2S-based host. It manages all of the basic G2S tasks required of a G2S host. The G2S host implementation is the main integration point between your business logic and G2S. If everything is done properly, your host application need only work with the G2S host implementation. Everything else in G2S should be hidden from you.

Quick Start for G2S includes:

- an HTTP 1.1-compliant stack based on Jetty 9.x.
- a SOAP 1.1 stack based on Apache CXF.
- a G2S transport mechanism based on Apache CXF.
- an object marshalling/unmarshalling mechanism based on JAXB 2.0.
- a command dispatching mechanism that uses reflection to dispatch the received G2S commands.
- the infrastructure to manage the allocation of unique identification numbers, which are required by the G2S protocol.
- default implementations for the communications and cabinet classes.

The Quick Start .zip file contains the entire source code for the Quick Start SDK, including unit tests, Java source code, Ant scripts and the G2S schema and WSDL 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 needed.

Quick Start for G2S does **not** include:

- Apache CXF implements all of the SOAP security standards through the JAX-WS specification

from Sun. None of these are enabled inQuick Start but they can be through various CXF configuration options. See the CXF documentation for more information.
- There is no retry logic in the transport layer.
- There is no persistent storage in the transport layer. If you shut down Quick Start, pending messages will be lost.

Quick Start Deployment

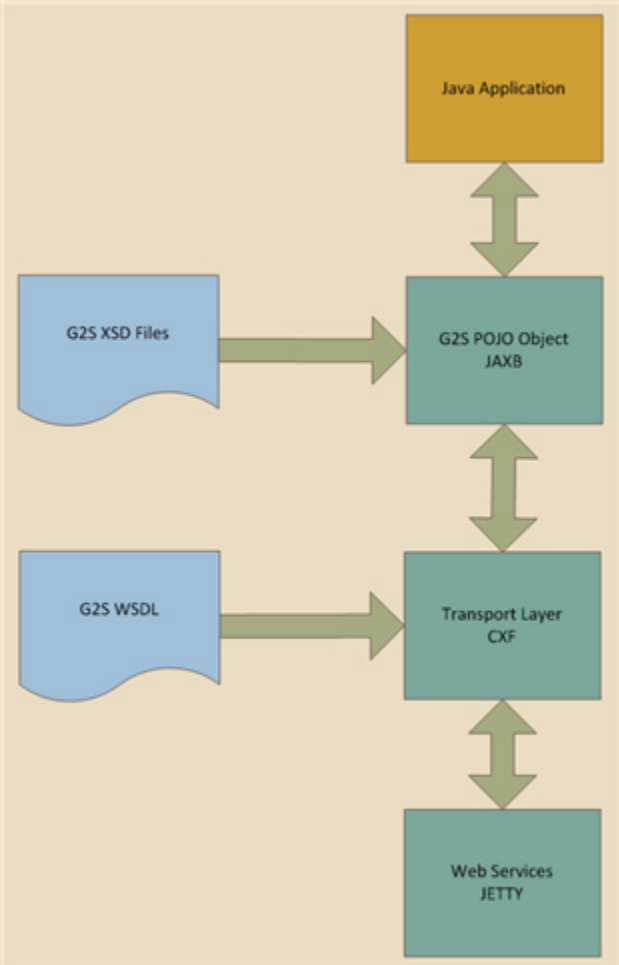
By default, Quick Start is a stand-alone application, using Apache CXF as the SOAP stack. This includes Jetty 9.x as the web container. Quick Start provides a Java main method that initializes the application, starts the SOAP stack, and then waits for the application to take action. This deployment option is aimed at stand-alone applications that can be successful with the CXF SOAP stack.

Resources

The following are sources for the resources referenced in this document:
- **Java Platform** - The SDK makes use of the JAVA 9 platform. You can download the Java Development Kit (JDK) directly from Sun Microsystems. https://openjdk.java.net/
- **JAXB** - Java Architecture for XML Binding (JAXB) generates Java objects from an XML schema. JAXB 2.0 is bundled with Java 9.
- **Eclipse IDE** - The Quick Start SDK is supplied as an Eclipse project that can be imported into the Eclipse Integrated Development Environment (IDE). It was developed using Eclipse 3.x (Europa). https://www.eclipse.org/
- Eclipse in Action by David Gallardo. Eclipse is an excellent IDE and a Radical Blue Gaming standard. If you are new to Eclipse, we recommend this book.
- **CXF** - Apache CXF is an implementation of a Simple Object Access Protocol (SOAP) stack that generates Java objects and method calls from WSDL files. http://cxf.apache.org
- **Jetty Web Server** - Jetty is an open-source, standards-based, full-featured web server implemented entirely in Java. Jetty is bundled with CXF as the default web container. https://www.eclipse.org/jetty/
- **G2S Message Protocol v3.1.0 Package** - Available from the Gaming Standards Association. https://www.gamingstandards.com/en/standards/g2s-game-system
- **GSA Point-to-Point SOAP/HTTPS Transport Specification Package** - Available from the Gaming Standards Association. https://www.gamingstandards.com/en/standards/xtp-transport

Development Configuration

The following is a view of your Java application running in a development (Jetty) environment:



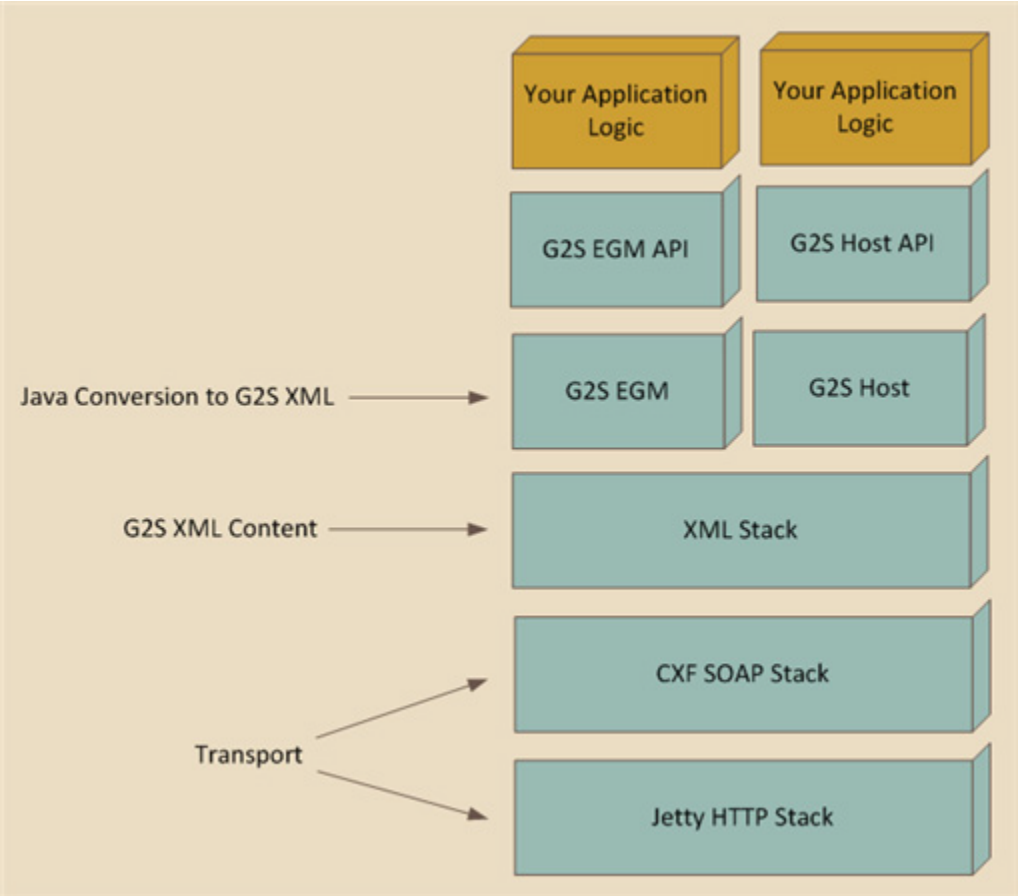
In G2S there are two marshalling and unmarshalling steps. An inbound message is unmarshalled from a SOAP string into a Java string. This transformation is defined in the G2S WSDL file.

Then the Java string is unmarshalled into a message POJO. This transformation is defined in the G2S XSD file.

The reverse transformations happen when a G2S command POJO is sent out through the SOAP stack.

Quick Start for G2S Stack

Quick Start for G2S provides an API to G2S, allowing anyone to write a G2S EGM or a G2S host application. To this end, your application interacts with less than 10 percent of Quick Start . For this reason, Quick Start is written as a vertical stack.



The layers in blue are provided by the RQS. Your layer, at the very top, uses the API layer to invoke the Quick Start services.

Note If you find yourself interacting with any layer other than the API layer, contact RadBlue.

Chapter 2

Message Processing by the EGM and G2S Host

Outbound Message Processing

The following section shows the path an outbound G2S message follows as it works its way through the G2S EGM or the G2S host.

API Layer

The EGM and host each have an API that maps methods directly to G2S commands (for example, the sending of the commsOnLine message). Each API method takes all of the data necessary to complete the command. The application logic invokes the API method with the correct 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 a G2S command POJO. The POJO is then passed to the transport layer.

Transport Layer

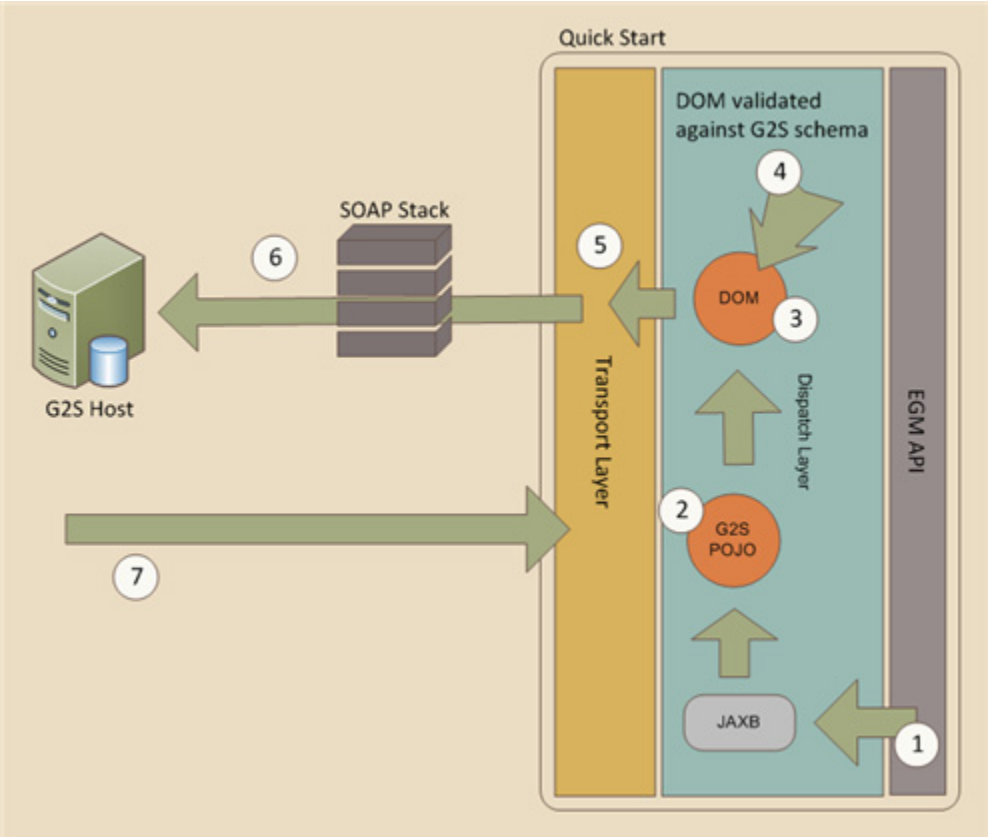
The transport layer is responsible for handling all outbound SOAP communication. The transport layer determines where the message should be delivered and passes it to an instance of the SOAP layer which is configured to talk to the remote destination.

SOAP Layer

The SOAP layer is responsible for physically delivering the G2S message. The SOAP layer first converts the G2S command POJO to an XML String by using JAXB. The result is an XML string that is valid against the G2S schema.

The SOAP layer then uses Apache CXF to invoke the remote SOAP method in the remote G2S host. Apache CXF uses the G2S WSDL to properly marshall the G2S command and the G2S ACK. Using the SOAP standard, the SOAP layer connects to the G2S host, exchanges the G2S message, and waits for the G2S acknowledgement (ACK).

Outbound Command Flow



1. Using the EGM or host API your application invokes the appropriate API method, passing in all required data.

2. The API builds the appropriate G2S command POJO.

3. The API passes the G2S command POJO to the transport layer.

4. The transport layer converts the G2S command POJO to a DOM using JAXB.

5. The transport layer validates the DOM against the G2S schema.

6. The transport layer converts the DOM to an XML string and then uses Apache CXF to deliver the G2S message to the remote G2S host.

7. The G2S ACK is returned from the G2S host back to transport layer. Note that a response must be sent within 30 seconds, or the transport layer returns a time-out exception.

Inbound Message Processing

The following section shows the path an inbound G2S message follows as it works its way through the G2S EGM or the G2S host.

Web Container

The web container implements the HTTP 1.1 standard. It is responsible for handling all of the socket issues, the handshaking and any SSL. Once the web container receives the SOAP request, it passes the request to the SOAP stack.

SOAP Stack

The SOAP stack, working with the web container, enforces the parsing and the validation of the G2S WSDL. The SOAP stack parses out the G2S payload from the incoming SOAP request and matches the received parameters against the WSDL method name. The SOAP stack then invokes the appropriate method in the SOAP service implementation.

The SOAP service implementation is the boundary between the generic SOAP stack and the custom business logic needed to execute G2S. The service accepts the parameters from the SOAP stack, validates the data and then processes the message.

To process the G2S message, the SOAP service uses the Java API for XML to convert the G2S message string to a (Document Object Model) DOM object. It then uses the Java API for XML Validation to validate the DOM object against the G2S schema. Any errors in this process are reported as errors in the G2S acknowledgement message (ACK).

Once the DOM is parsed and validated, it is wrapped in a G2S message POJO. This POJO encapsulates the DOM so that it can be processed by the dispatch layer. The service queues up the G2S message POJO into the dispatch layer for further processing. Now that the responsibility for the G2S message has been accepted, the SOAP service generates the appropriate G2S ACK and returns that to the SOAP stack, which in turn returns it to the caller through the web container.

Dispatch Layer

The dispatch layer is responsible for converting the G2S message POJO to a G2S command POJO. This is done through the Command Factory. The Command Factory examines the G2S message POJO (which usually contains just one command of the possible 500+ commands in G2S) and converts the data to a single G2S command POJO.

The command POJO is a key concept in that it saves the upper layers of the application from having to understand the actual construction of a G2S message. Without the Command Factory, the business logic layer would have to contain a huge amount of code that searched for the command to process.

Once the G2S command POJO is created, it is double-dispatched to update the EGM data model. The EGM data model is a data structure that maintains the current state of the EGM, either on the EGM or the host. G2S is designed to make this model agree on both sides of the conversation. RQS does this by using the same data structure in both implementations.

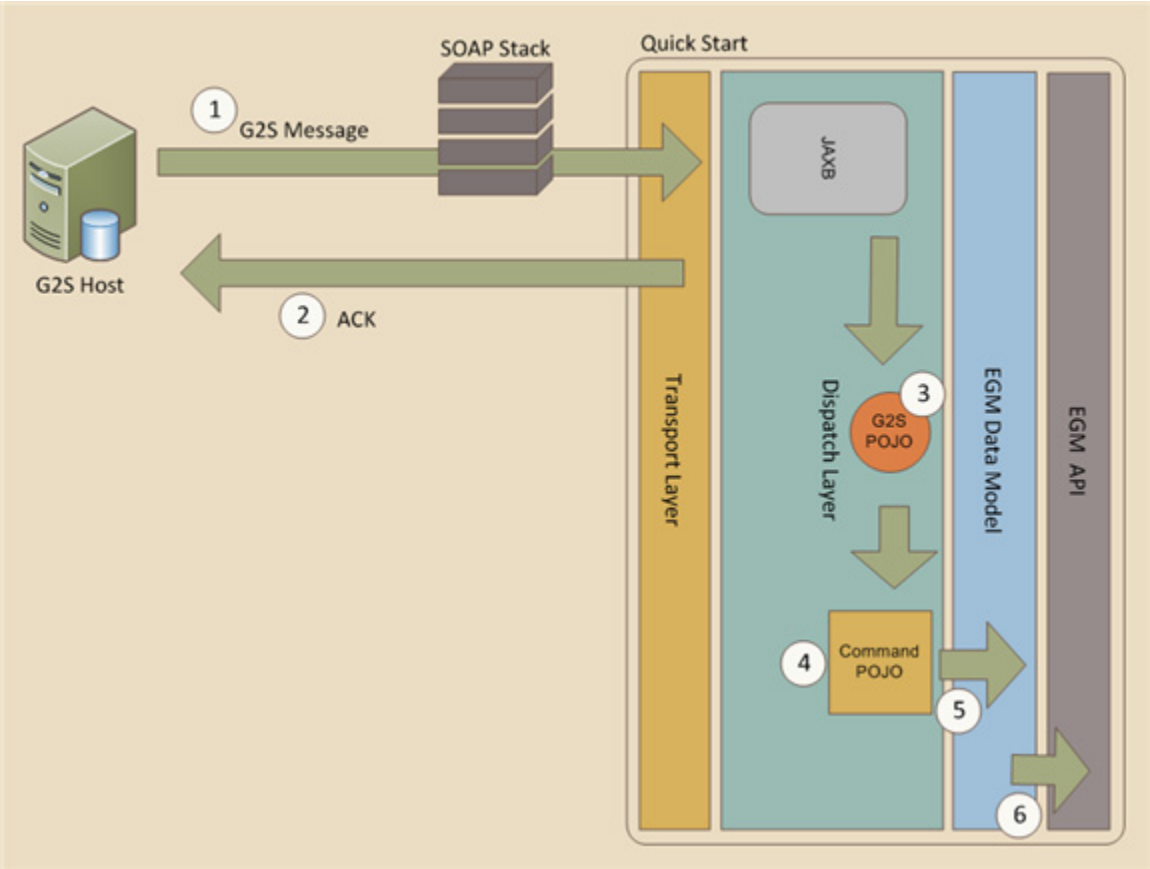
Once the dispatch layer has updated the EGM data model, it passes the command POJO to the application layer.

Application Layer

The top and final layer of the application is responsible for any extended or extra processing. A

number of commands in G2S are merely for updating the data model and usually do not impact the business logic of the application. In these instances, the command can be ignored. However, for some commands, once the data model is updated, additional logic (like database lookups or updates) need to performed. This is so the proper response can be generated. This business logic is encapsulated here, at the top-most layer of the application.

Inbound Command Flow



1. The G2S message arrives through the SOAP stack.

2. The G2S ACK is crafted and returned.

3. The Dispatch Layer converts the G2S message to a G2S message POJO, using JAXB.

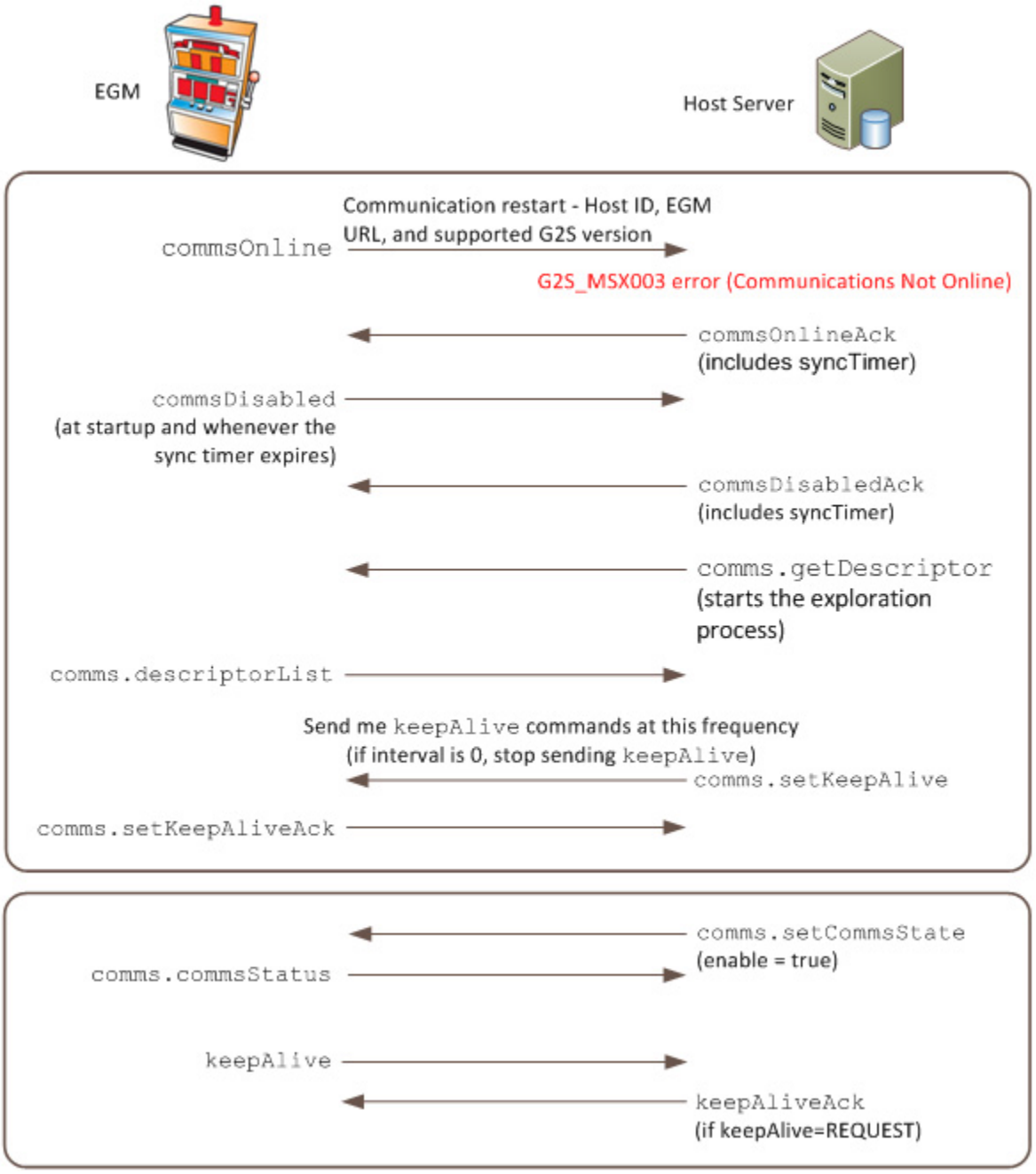
4. The Dispatch Layer converts the message POJO to a command POJO.

5. The command POJO is dispatched to update the data model.

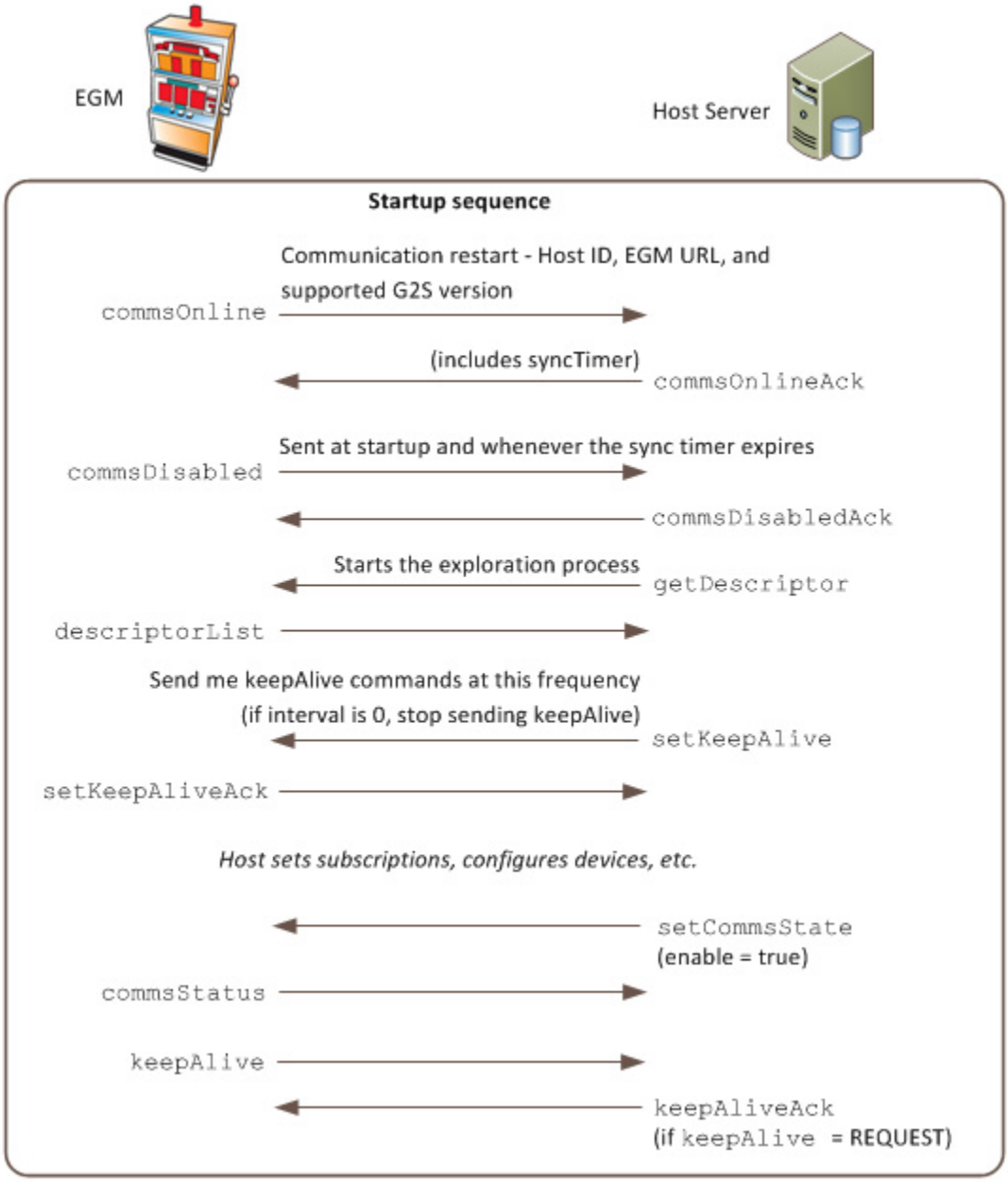
6. Your application is notified by the Dispatch Layer by the hand-off of an G2S command that has already been parsed and in most cases processed. Your application can now take whatever action is appropriate for the command.

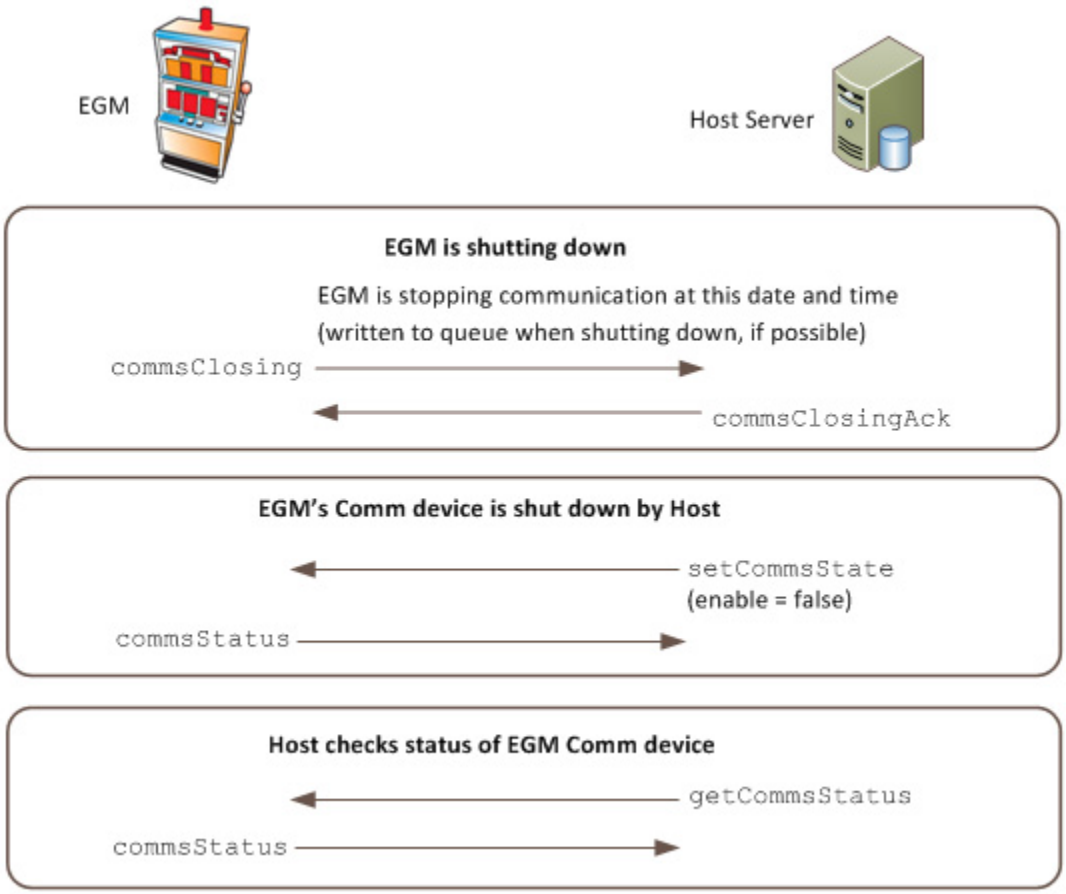
Class Sequence Flows

Start-up Algorithm

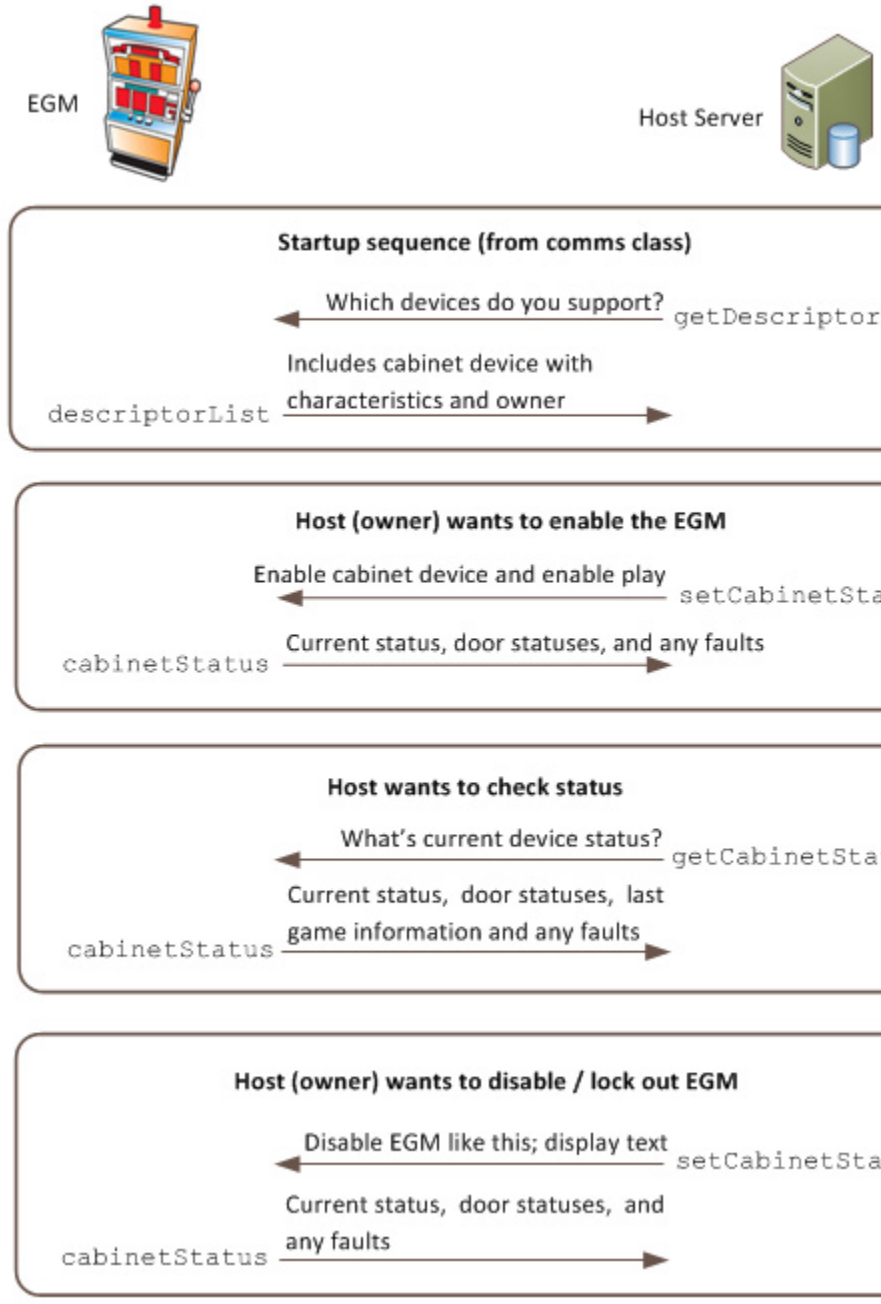


Communications Class




EGM
Host Server
EGM is shutting down
EGM is stopping communication at this date and time
(written to queue when shutting down, if possible)
commsClosing
commsClosingAck
EGM's Comm device is shut down by Host
setCommsState
(enable = false)
commsStatus
Host checks status of EGM Comm device
getCommsStatus
commsStatus

Cabinet Class



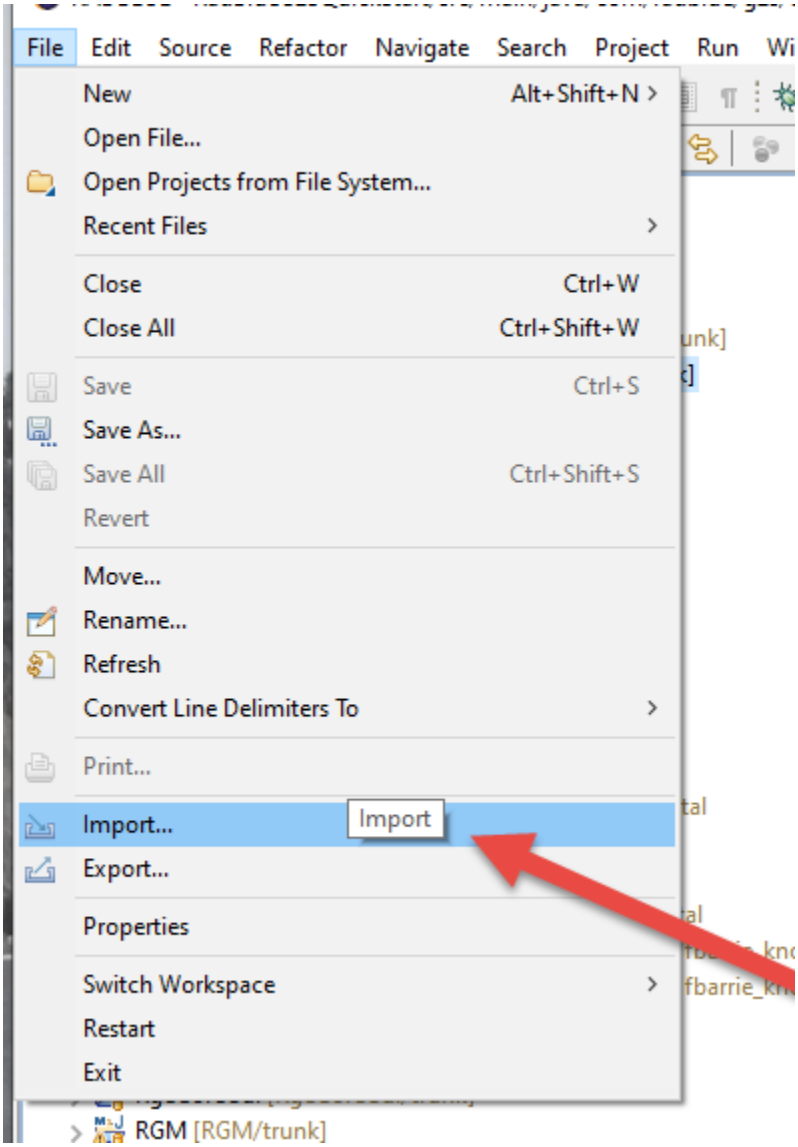
Chapter 3

Getting Started

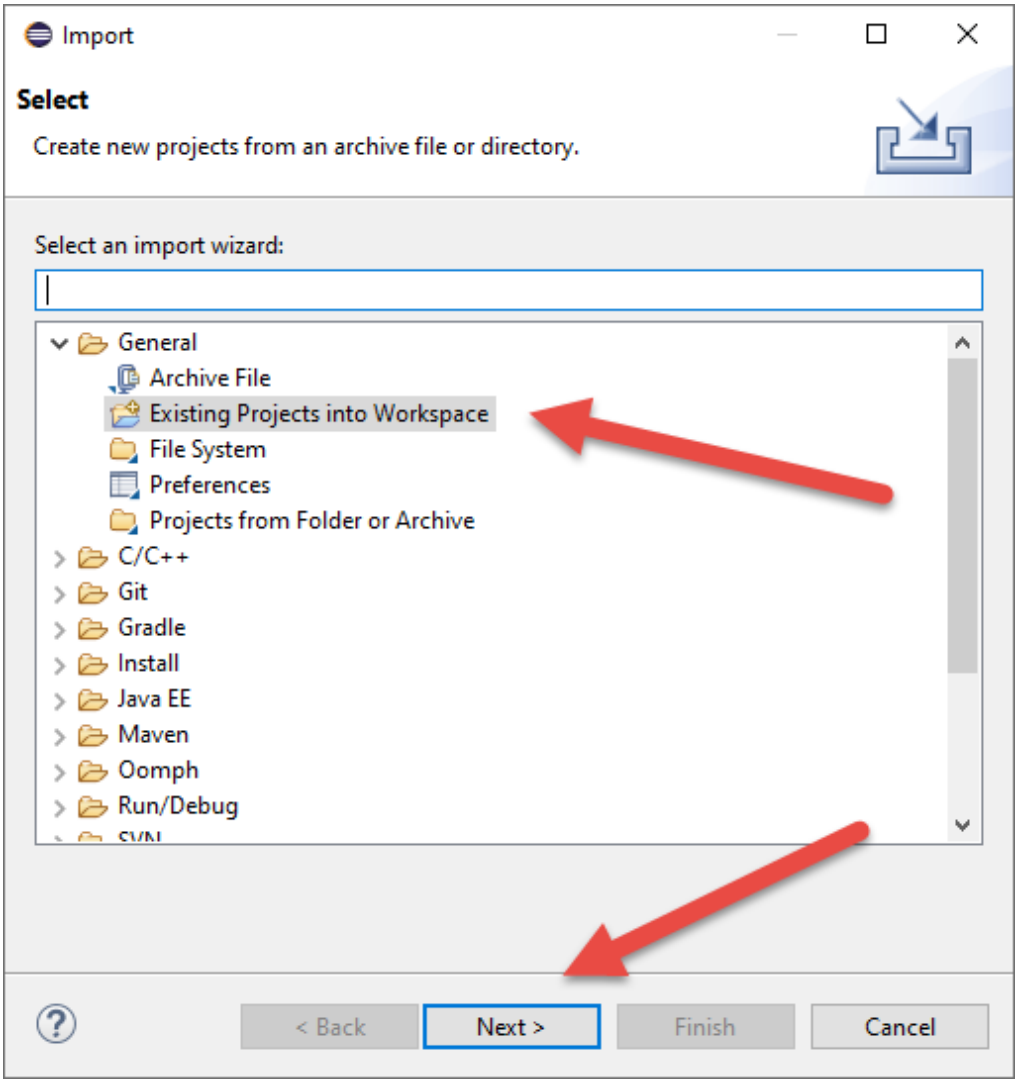
Import Quick Start into Eclipse

1. Open Eclipse.

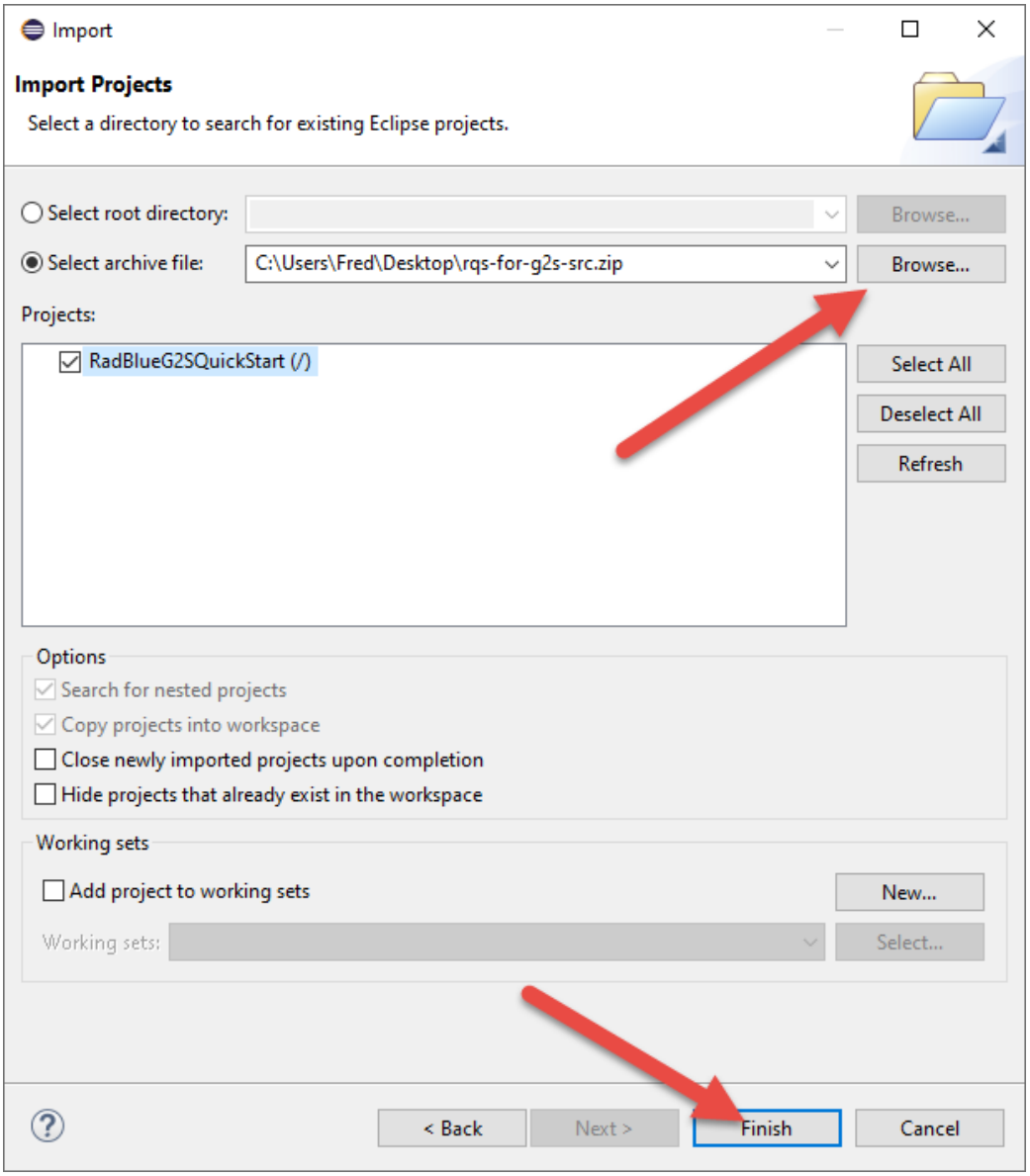
2. Select **File > Import...**



3. Select **Existing Projects Into Workspace**



4. Select the **rqs-for-g2s-src.zip** archive file



Quick Start Directories

The Quick Start project (root) contains the following directories:

- **bin** – Directory of .class files. If you are using a source code control system, exclude this directory.
- **conf -** files used during generation of project artifacts
- **src/main/c** - Java Native Interfaces for GSA's Multicast transport
- **src/main/includes** - Includes needed for Multicast
- **src/main/java** - Quick Start source code
- **src/main/resources** - Resource files that are needed by Quick Start.

- **src/test/java** – Quick Start unit tests.

- **vendor/lib** – Directory where all third-party .jar files reside. This is also where the JAXB-generated .jar files reside.

- **vendor/src** – 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.

Chapter 4

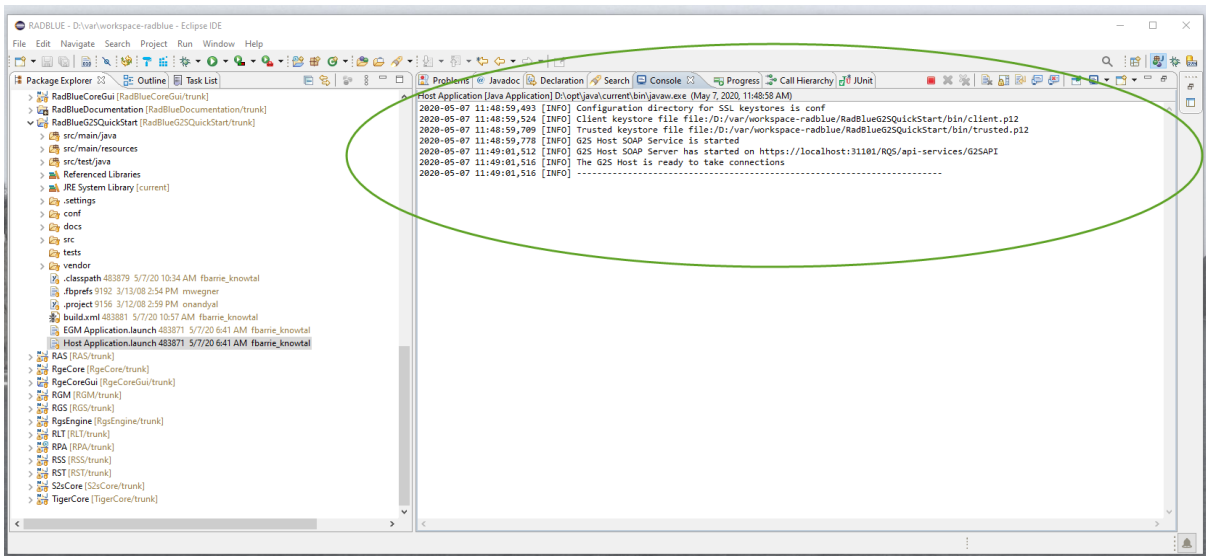
Run the G2S Host Demonstration Program

A stand-alone demonstration program for the G2S host system is included with Quick Start. The demonstration program uses Jetty as the web container, Apache CXF for the SOAP stack. You can run the demonstration program from Eclipse.

1. Open Eclipse.

2. Open the **RadBlueG2SQuickStart** project

3. Right-click **Host Application.launch**, select Run As > Java Application.



By default, the Host application listens on the following URL:
http://localhost:31101/RQS/api-services/G2SAPI
You can change that value by editing the HostConfig constructor in the application main method.

4. Click the **Console** tab view G2S Host messages.

Connecting through the RadBlue System Tester

If you are connecting to the G2S host implementation through the System Tester SmartEGM:
1. Modify your SmartEGM configuration file to use the following host URL:
http://localhost:31101/RQS/api-services/G2SAPI

2. Start the SmartEGM.
The SmartEGM sends the commsOnLine command to the G2S host application. The G2S host application communicates with the SmartEGM, sending commands back and forth until both the communications and cabinet devices are enabled.

3. Once the communications and cabinet devices are enabled, begin sending messages to the G2S host application.

Connecting throught the RadBlue System Tester

If you are connecting to the G2S host implementation through the System Tester SmartEGM:

1. Modify **HostApplicationMain.java** and change the arguments in the HostConfig object to use the following URLs:
    a. hostId : 1
    b. hostURL : http://localhost:31101/RQS/api-services/G2SAPI

    Note: Since the Quick Start and RST do not share a common certificate authority you must use the non-SSL URL.

2. Modify your SmartEGM configuration file to use the following host URL:
http://localhost:31101/RQS/api-services/G2SAPI

3. Start the SmartEGM.

    The SmartEGM sends the commsOnLine command to the G2S host application. The G2S host application communicates with the SmartEGM, sending commands back and forth until both the communications and cabinet devices are enabled.

4. Once the communications and cabinet devices are enabled, begin sending messages to the G2S host application.

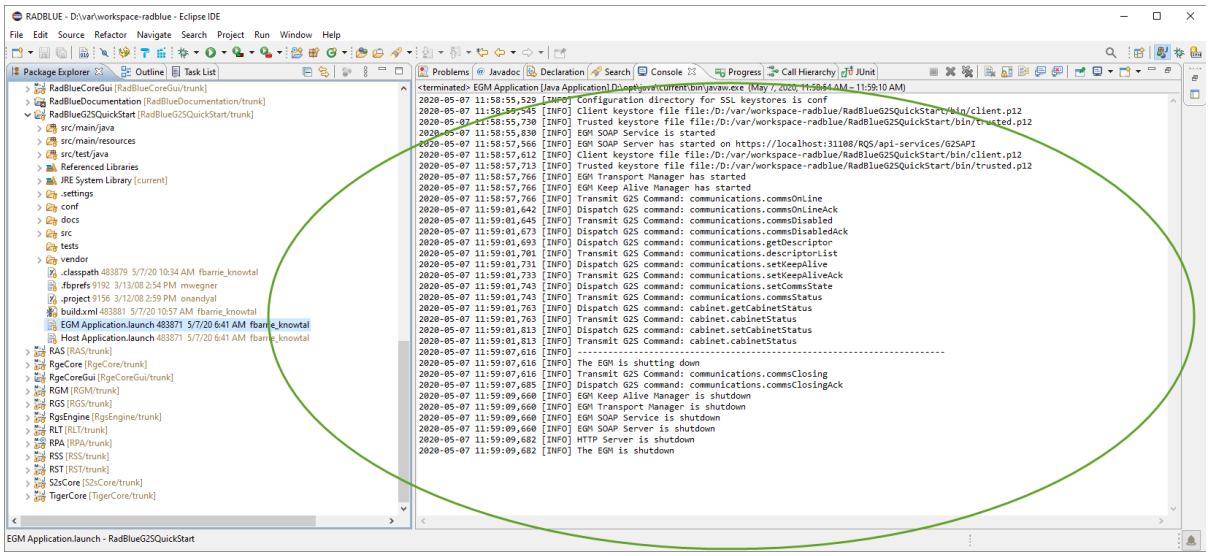
Run the G2S EGM Demonstration Program

A stand-alone demonstration program for the G2S EGM is included with Quick Start. The demonstration program uses Jetty as the web container, Apache CXF for the SOAP stack. You can run the demonstration program from Eclipse.

1. Open Eclipse.

2. Open the **RadBlueG2SQuickStart** project

3. Right-click **EGM Application.launch**, select Run As > Java Application.



By default, the EGM application listens on the following URL:
https://localhost:31108/RQS/api-services/G2SAPI and connects to Host ID 1 at https://localhost:31101/RQS/api-services/G2SAPI.

You can change that value by editing the EgmConfig constructor in the application main method.

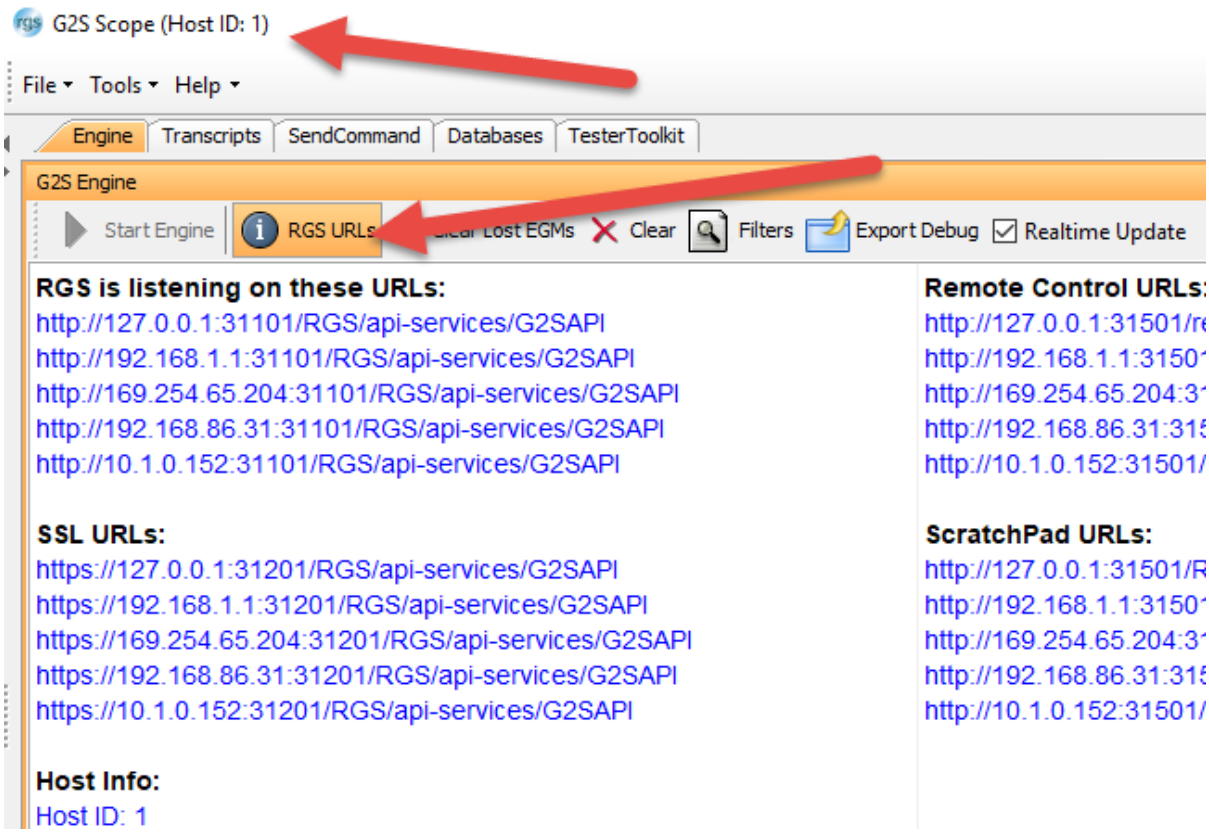
4. Click the **Console** tab view G2S messages.

5. The EGM application will disconnect from the host after 6 seconds and will exit.

Connecting to RadBlue G2S Scope

If you are connecting to the G2S EGM implementation to RadBlue's G2S Scope:

1. Determine the host ID and URL of RGS



a. Click the **RGS URLs** button to display the URLs that the RGS is currently using.
b. Select the non-SSL URL on the localhost (127.0.0.1).
c. Note the host ID in RGS's title bar.

Note: Since the RGS and Quick Start do not have a shared certificate you must use non-SSL URLs.

2. Modify **EgmApplicationMain.java** and change the arguments in the EGMConfig object to use the following URLs:

a. egmURL: http://localhost:31108/RQS/api-services/G2SAPI
b. hostURL : From RGS (by default: http://localhost:31101/RGS/api-services/G2SAPI)
c. hostId : From RGS (by default: 1)

3. Run the EGM application.

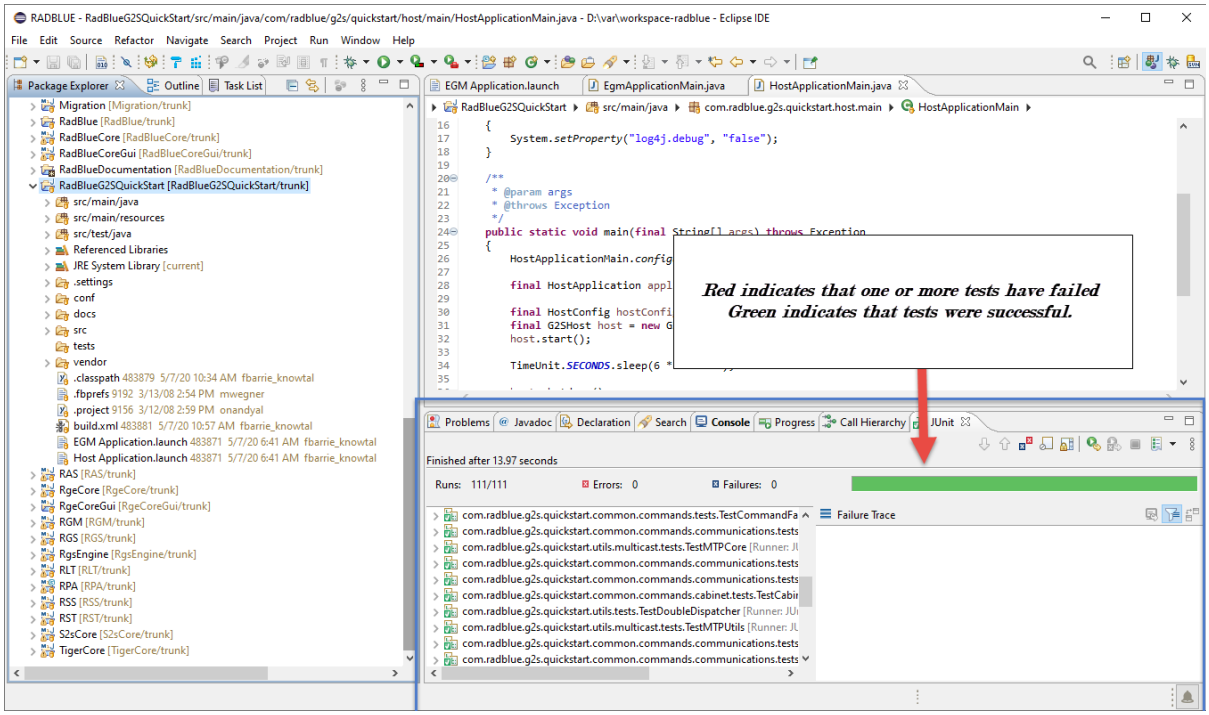
Running Unit Tests

Radical Blue Gaming is a proponent of Extreme Programming and the practice of writing unit tests. Quick Start is no different. At the present time, there are 111 unit tests that exercise much of Quick Start. We encourage you to continue this practice
as you add code to Quick Start for G2S. You can run the full suite of Quick Start unit tests by executing the following JUnit
test:

1. Open **Eclipse**.

2. Click on the root of the Eclipse project (**RadBlueG2SQuickStart**).

3. From the menu bar, select **Run** > **Run As** > **JUnit Test**.



4. Select the **JUnit Tests** tab to display the JUnit view and start the progress bar.

5. Once the test is finished, look at the JUnit view, and verify the indicator bar is green (indicating a successful test outcome).