Revised 3 July 2022

C2SIM Server Reference Implementation Documentation– Version 4.8.3.1

User Instructions

Recent additions highlighted.

Overview

GMU C4I and Cyber center is making available C2SIM Server release 4.8.3.1 and C2SIM Client library release 4.8.3.1 as the current version of the C2SIM Reference Implementation. Note that the Server and Client release numbers are now the same, though the last digit may not match if changes to one do not require changes to the other. These packages support the following:

- C2SIM orders and position reports for C2SIM Version 1.0 that was submitted to SISO for standardization, augmented for CWIX 2022 as C2SIM_CWIX2022v1.2
- Legacy protocols C2SIM Version 0.0.9 for Initialization, Orders and Position Reports, MSDL for initialization, and IBML09, and CBML-Light for orders and position reports
- Translation of basic data among all four dialects for orders and reports
- Processing and marshalling of C2SIM initialization messages
- Translation between C2SIM initialization messages and MSDL
- Operator commands and session state messages
- Emulated Cyber attacks
- Collection of response time statistics by the server
- Verification that the version numbers of the client meets the minimum required for a particular version of the server
- Recording and playback of server input, under control of the C2SIMGUIv2.12.1
- Support for the new C2SIM SystemCommand messages in C2SIM_CWIX2022v1.2
- STOMP distribution of C2SIM Orders and Reports selectable by clients

The C2SIM formats used by the current version of the server are in compliance with C2SIM Standard ontology and schema draft Version 1.0 and schema C2SIM_CWIX2022v1.2.

Server Configuration

The server(s) usually run as a VM using the following components

- Linux Centos 7
- Java Version 8
- JDOM 2.0.6
- Apache Tomcat 8.0.30 Web Services (RESTful Web Services)
- Apache Apollo 1.7.1 Messaging (STOMP Messaging Server)

The server comes packaged in a .war (Web Application Archive) file, including code and documentation and all needed pieces to run in this environment.

The basic server configuration is shown below. Note that a single input message may be sent to several destinations. Also, that the same client may submit REST messages and receive STOMP messages.

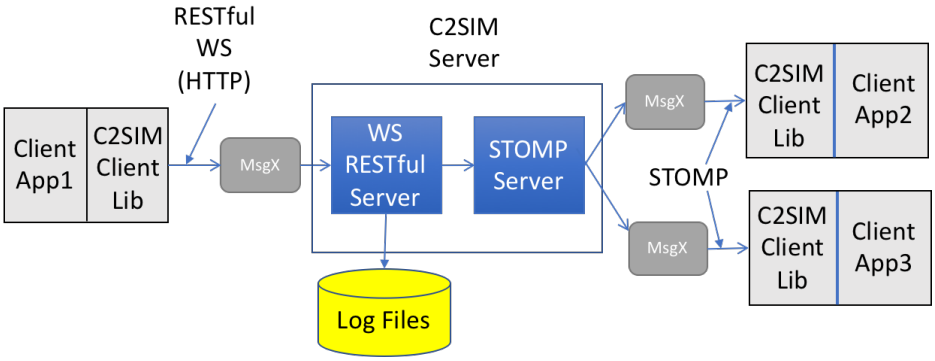


Figure 1: Server Operation

Basic Message Processing

Sending – Messages are sent to the server using RESTful Web Services protocols via the C2SIMClientREST_Lib class in the C2SIM_Client Library. Based on information obtained from the XML schema, the server characterizes the message determining the type of message and the BML/C2SIM version (also known as dialect) used. For backward compatibility, messages may be translated to the legacy three dialects, if that feature is turned on in the server. The message is then sent to the STOMP server where it is published to subscribers. The server adds a number of STOMP headers to the outgoing message so that filtering may be done by the STOMP server or by receiving system. In particular, the client must choose to receive Orders and/or Reports in a separate API call before connecting to STOMP.

Per the REST standard, in any transaction the client disconnects after the message is sent and a response from the server has been received. Submitting a message consists of the following:

- Instantiating a C2SIMClientREST_Lib object
- Setting parameters
- Sending the request
- Receiving the result
- Destroying the C2SIMClientREST_Lib object

Under control of a switch in the C2SIMServer.properties file, the server may request that the response time for the transaction be sent by the client to the server as a separate transaction for use in testing server performance. This is completely handled by the C2SIM Server and the

C2SIMClientREST_Lib class of the C2SIM client library; the client application is not required to participate. The server records the response time in the C2SIM Debug log.

When a message is sent to the server, a message number is generated. This message number is returned in the response, echoed back by the client library with response time statistics, and added to the STOMP message as one of the headers. This enables tracking of all messages from end to end.

Receiving – Messages sent to the C2SIM server are sent to all systems per their subscriptions to the STOMP server. Message receipt is via the C2SIMClientSTOMP_Lib class of the C2SIM_Client Library. Most client systems are both senders and receivers. Receiving systems establish a STOMP subscription, using TCP, via the C2SIM_Client library. The TCP connection is kept open as long as the client holds a STOMP connection; multiple messages will be received over this subscription. Both blocking and non-blocking calls to receive the next message are supported by the library. Receiving consists of the following:

- Instantiating a STOMPClientSTOMP_lib object
- Set parameters (Topic should normally be /topic/C2SIM)
- Specifying optional subscription(s)
  Example: c. addSubscription (message-selector = 'C2SIM_Report');
- Connecting by executing the connect() method of C2SIMClientSTOMP_lib.
- Loop and receive messages c.getNextBlock() or c.getNextNoBlock() indefinitely
- Connection stays open until c.disconnect() is called

STOMP messages include a variable number of header parameters, much like HTTP messages. These may be used for filtering after the message has been received by the STOMP server ensuring that only messages of interest are processed. In addition, the connect request may carry a subscribe string (format much like an SQL statement) that will cause filtering to be done by the server, delivering only messages on that connection that satisfy the subscribe string.

Headers (underlined) that may be used for filtering include the following:

protocol (BML or SISO-STD-C2SIM)

submitter - Identifier used when the message was sent to the server

message-selector Indicates the type of message. Possible message selectors are:

MSDL
IBML09_Report
IBML09_Order
CBML_Order
C2SIM_Order
C2SIM_Report
C2SIM_Command
C2SIM_Initialization

message-type – Indicates general type of message

Order
UNIT (Position or General Status Report)

message-dialect – Specific BML version used

IBML09
CBML
C2SIM

The server can translate among all four dialects. Which translations are performed is controlled by server startup options in the c2simServer.properties file.

C2SIM Message Parameters

sender
receiver
communicativeActTypeCode
conversationid

Other Server Functions

Translation - Messages can be translated among four schemata:

- C2SIMv0.0.9
- C2SIMv1.0.0
- IBML09
- CBML-Light

This includes orders and Position/General Status reports.

Initialization data can be translated between:

- C2SIMv0.0.9
- C2SIMv1.0.0
- MSDL

Initialization – Exercise initialization in the past has been done using MSDL. C2SIM is intended to combine the functions of CBML and MSDL. Before the simulation exercise starts, initialization will be performed by processing C2SIM ObjectInitializationBody or C2SIMInitializationBody transactions that define the characteristics and initial positions of Units and other items. Multiple sets of these transactions may be submitted by different sources. An alternative method for entering initialization information is to position a file on the server containing the same information as is submitted in ObjectInitialization transactions and then executing a LOAD command.

When the server receives a SHARE command, the accumulated set of definitions in C2SIMInitializationBody format will be published to all participants and the simulation will be started. Additionally, the accumulated C2SIM initialization information will be translated to MSDL and published. No additional initialization transactions will be accepted after the execution of a SHARE command. A START command starts the scenario and at this point the server is prepared to accept orders and reports. When clients receive a C2SIMInitializationBody message it will contain all units and other items that have been provided through initialization.

C2SIM Initialization messages (as passed to the C2SIMClientLib), like all C2SIM messages, will start with <MessageBody>. The C2SIM Server will process Initialization information contained within <C2SIMInitializationBody>, <ObjectInitializationBody>, or <ObjectInitialization> (Version 0.0.9).

Unit Status Tracking – The initialization database establishes the initial position and other properties of the units in the simulation. After that, the server will maintain the last position received via position report for each unit. (Whether this takes place is configured in the c2simServer.properties file.) A query has been implemented to access this data. Q*UERYINIT* will return the initialization data distributed at the beginning of the exercise, in C2SIMInitializationBody format with updated positions. It also will return the same data in MSDL format. This response can support late joining of clients after an exercise has been started by SHARE and RUN. (See Appendix C.)

C2SIM Message Envelope Support – The new C2SIM standard specifies an XML message header separate from the actual data. This header contains a number of fields used to identify the sender, specific receivers, command indicating the type of message

(CommunicativeActTypeCode), unique identification for this specific message (MessageID) and for a series of messages (ConversationID).

The C2SIM Client Library does most of the processing of the C2SIM message header, including header creation, stripping off the header before delivering the original message, and sending a response where required and other functions. See the C2SIM_ClientLib Javadoc file for more information.

Header Creation – When a C2SIMClientREST_Lib object is created with a parameter list that includes sender, receiver, performative, and protocol version indicating that the transaction is to be encapsulated In a C2SIM message, the Client Library creates the C2SIM header in preparation for sending the message. The C2SIM standard doesn't describe the format of the Sender and Receiver fields. A coalition planning on using C2SIM should establish a plan so that all Sender and Receiver entities have unique names. The messageID and conversationID are created as new UUIDs by the library code. These can be accessed and/or changed before the message is transmitted.

Header Access - A getC2SIMHeader() method in the C2SIMClientREST_Lib object will return a reference to the C2SIMHeader, which then may be queried and/or modified before the actual message is sent. On receipt of a message the C2SIMClientSTOMP_Lib class returns a C2SIMSTOMPMessage object. This object also supports a getC2SIMHeader() method which returns the C2SIM header of the message that was just received.

Recording and Playback of the C2SIM Message Stream

Previous versions of the server provided for logging of output messages to directory c2simFiles/c2simReplay. A Player module (also separately present in the C2SIMGUI) re-sends messages from the logfile so all connected clients. A control panel for this capability has been included in the C2SIMGUI starting with version 2.12.1.

Interfacing without the use of the Client Library

Two versions of the client library are provided, in Java and C++. The Java library consists of a single jar file and was built on Java 1.8.0_181 and NetBeans 8.1. The C++ library consists of a lib64 folder containing .lib files and was built on Microsoft Visual Studio 16.4.6. The C2SIM Server components conform to industry standards and may be used without the supplied client libraries although **this is not recommended**. The library routines are straightforward to use, are provided with source code, and are well documented. Using them saves considerable implementation time.

The C2SIM server is implemented using REST procedures. The following URLs are used to access the server:

URL: http://hostname:8080/C2SIMServer/c2sim Submission of BML and C2SIM documents

REST Parameters:

protocol	C2SIM or BML
submitterID	Identification of the submitter
sender	Identifier of sending process (C2SIM Only)
receiver	Identifier of intended recipient (C2SIM Only)
communicativeActTypeCode	Type of communication (C2SIM Only)
conversationID	Identity of a series of messages
version	Version of client software – Currently 4.8.3.1

URL: http://hostname:8080/C2SIMServer/command Submission of session commands

REST Parameters:

command	Command as described later in this document
parm1, parm2	Parameters used with above command
submitter	Identification of the submitter
version	Version of client software – Currently 4.6.3

The STOMP server is an off-the-shelf copy of Apache Apollo 1.7.1 and implements standard STOMP version 1.2.

C2SIM Client Library

The ClientLib is available in both Java and C++. The details of the Java C2SIM Client Library are contained in the JavaDoc file which accompanies this document. The C++ version provides equivalent implementations of all Java methods. Appendix A below gives sample code using the Java ClientLib.

C2SIM Client Utilities

In the following, xxx indicates the version number of the program being described.

Several standalone utilities are provided, primarily as examples of how to program the C2SIM Client library. Most of their functions also are available under a user-friendly interface in the open source C2SIMGUI, available at https://OpenC2SIM.github.io.

The use of these utilities is documented below. The "_ALL" suffix is in indication that all dependencies are included in the jar file and the jar file is executable as-is. Note that the source code is also included in the jar file. The source code may be obtained by completely unzipping the jar file as follows:

```
jar  -xvf  C2SIM_WSClient2-xxx_ALL.jar protocol  version
```

C2SIM_WSClient2-4.8.0.x_All – Submit an xml document to the C2SIM server via RESTful Web Services.

java -jar C2SIM_WSClient2-xxx_ALL.jar hostname xml_file submitterID protocol

hostname	Name or IP address of the C2SIM Server
xml_file	File containing the xml data to be submitted
submitterID	Name or initials identifying the submitter.
protocol	BML or SISO-STD-C2SIM or Cyber
version	C2SIM Protocol version (0.0.9 or 1.0.0)

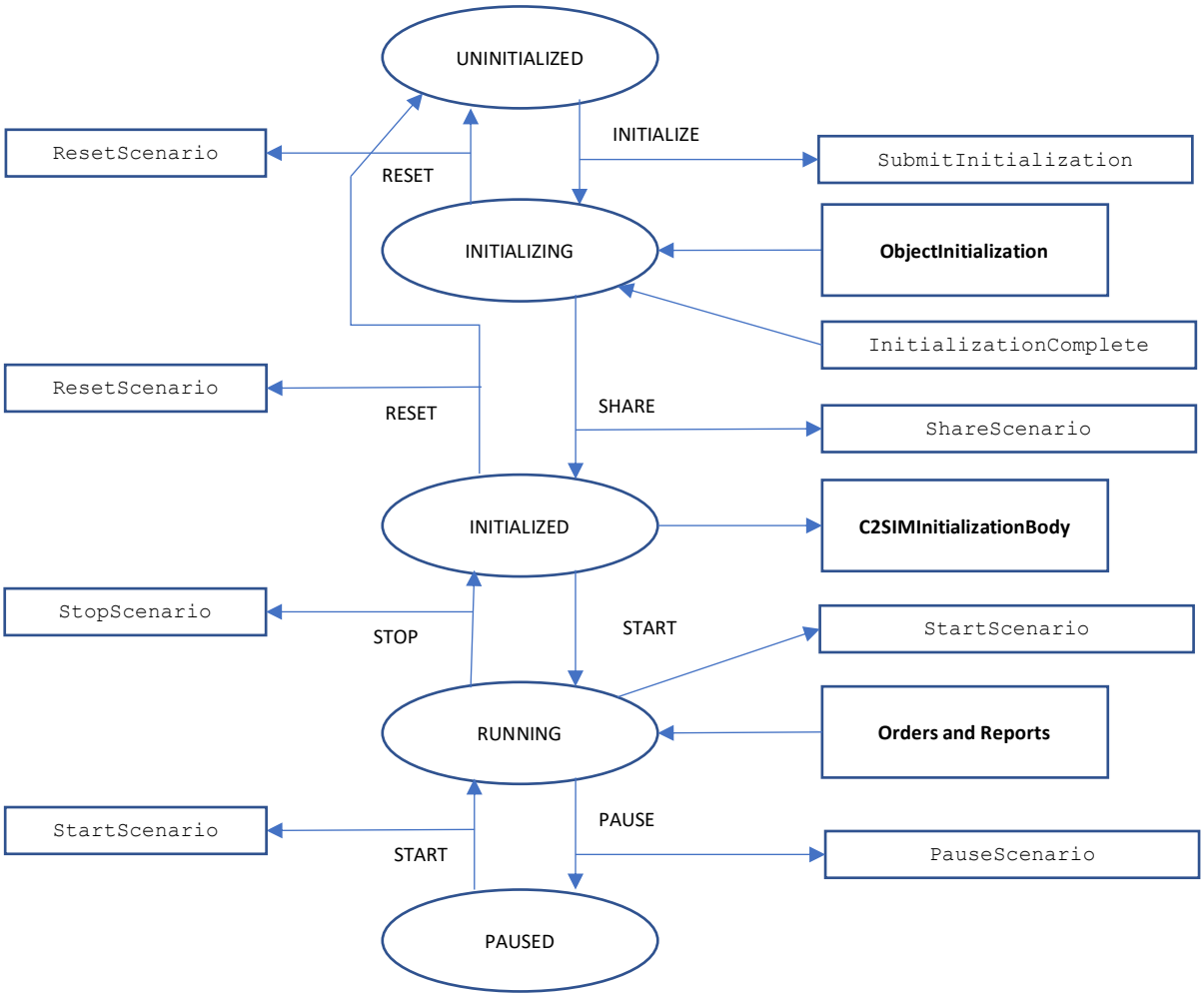
If the protocol is SISO-STD-C2SIM, a C2SIM header will be generated using "ALL" for sender and receiver and "Inform" for the C2SIM performative. An experimental module has to emulate Cyber attacks that delete or change C2 orders and report has been added to the server and to the WS Client. This capability "attacks" incoming messages according to parameters in a control file. This control file is submitted to the server using the WSClient specifying "Cyber" as the protocol. The full capability is described in a separate document.

C2SIM_StompClient2-4.8.0.x_ALL – Connect to a STOMP server, receive all published messages and print them via System.out.println()

java -jar C2SIM_StompClient2-xxx_ALL hostname

hostname Name or IP address of the STOMP server.

C2SIM Commands – A number of commands are used to submit initialization data, manipulate the database of Units and to control the simulation. The diagram below shows the server states (Ovals), Transitions (Arrows) and allowed C2SIM/BML Transactions (Horizontal arrows). Commands are in all caps, C2SIM XML initialization messages in boxes in **bold** and System Command messages showing status and sent to all systems are in boxes (non bold).


UNINITIALIZED
INITIALIZE
SubmitInitialization
ResetScenario
RESET
INITIALIZING
ObjectInitialization
InitializationComplete
ResetScenario
RESET
SHARE
ShareScenario
INITIALIZED
C2SIMInitializationBody
StopScenario
STOP
START
StartScenario
RUNNING
Orders and Reports
StartScenario
START
PAUSE
PauseScenario
PAUSED

C2SIM Server Commands

Command	Parm1	Parm2	Required Simulation State	Actions
Unit Database Manipulation Commands				
LOAD	filename		UNINITIALIZED	Load contents of named file process the initialization data found there as if received over network..
Simulation State Commands Initial state when server starts is UNINITIALIZED				
RESET	Password		INITIALIZING	Reset database and state back to "UNINITIALIZED". Delete initialization data. Reset emulated Cyber attacks
SHARE	Password		INITIALIZING	Publish existing database Terminate initialization phase Format is C2SIM_MilitaryOrganization Set simulation state to "INITIALIZED" Save Unit DB for late joiners
START	Password		INITIALIZED Or PAUSED	Start simulation Set simulation state to "RUNNING"
STOP	Password		RUNNING	Stop simulation Don't delete initialization data. Set simulation state to "INITIALIZED"
PAUSE	Password		RUNNING	Pause the simulation. Set simulation state to "PAUSED"
STATUS			ANY	Return state of server using the same format returned to submitters of XML documents.

Command	Parm1	Parm2	Required Simulation State	Actions
Initialization Information Query Commands				
QUERYINIT			INITIALIZED or RUNNING	Return all initialization data as originally specified for initialization in C2SIMInitializationBody format. Also translate to MSDL and return that as well.
RESTful POST of C2SIM_ObjectInitialization Document			INITIALIZING	Save in local initialization database.
RESTful POST of MSDL Document			N/A	Translate to C2SIM and store in initialization database
RESTful POST of Position or General Status Report			RUNNING	Update unit position from report

RESTful POST of documents is done via the c2simRequest method in the C2SIMClientREST_Lib class in the C2SIM Client Library.

Commands are submitted via the c2simCommand method in the same class, or as C2SIM messages with a SystemCommandBody.

Command line utilities for submission of C2SIM/BML documents (C2SIM_WSClient2_xxx_ALL.jar) and for commands (C2SIM_Command-xxx_ALL.jar) described earlier in this document.

As of v4.8.1.1, the server also supports SystemCommands in SISO draft C2SIM schema 1.0.2. In general, these are MagicMove; Simulation and Playback time scaling; Server Recording, and Server Playback. The control client for these is included in the C2SIMGUI, available as open source from OpenC2SIM.github.io.

Accessing Log Files

The server collects information critical for problem analysis in two files, replayLog and debugLog. The replay log contains a record for each message received, each server command received, and each message published via STOMP. A critical piece of data is the message number which is assigned as the message is received, appears in most debug log messages and appears in the response returned to the client. Whether messages are logged to replayLog is controlled by C2SIM SystemCommands for "Recording" for C2 user access. If recording is turned off, replayLog messages are directed to the debugLog.

The debug log contains miscellaneous messages showing the progress of each message through the server and errors detected either via checks in the server code or as the result of an exception that might be thrown.

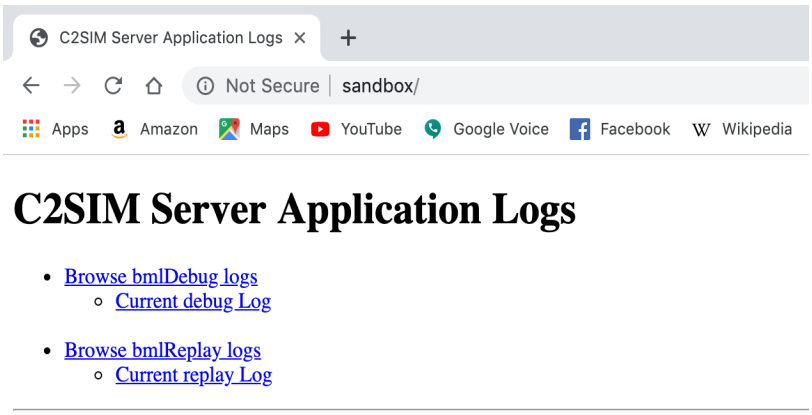
Both log files are collected during the day and then rolled over at midnight (local to the server). The names and locations for the two log files are for the current day:

/home/bmluser/c2simFiles/c2simDebug/debug.log

/home/bmluser/c2simFiles/c2simReplay/replay.log

Log files from previous days are located with the daily log and are named yyyy-mm-dd.debug.log and yyyy-mm-dd.replay.log.

The log files can be accessed via port 80 of the C2SIM Server platform using a web browser (port 80).



Server deployment, initialization and control

Server initialization and operation is controlled with a (java) properties file, which is described in Appendix B to this document. This file contains a number of settings that may be used to select a number of options. The file named c2simServer.properties is located at:

/home/bmluser/NetBeansProjects/Server/C2SIMServer/src/main/resources.

During the build process the properties file is added to the web archive file, C2SIMServer.war, used to deploy the server application to Tomcat. To deploy a war file it is copied to /opt/tomcat/apache-tomcat-8.0.30/webapps. When Tomcat starts and detects a new war file it unpacks in the webapps folder. After Tomcat initialization the webapps folder will contain a folder named, C2SIMServer##a.b.c.d and the original war file C2SIMServer##a.b.c.d.war. The a.b.c.d is the release number of the server application. After unpacking the properties file will be located at

/opt/tomcat/apache-tomcat-8.0.30/webapps/C2SIMServer##a.b.c.d/WEB-INF/classes

To change one or more properties in the properties, file modify it with a text editor and restart tomcat by executing stop-all and then start-all. These script files are in the bmluser home directory.

The current properties file is displayed in Appendix B below. Note that the file contains initialization information for starting the server, location of various files on the server, the password required for command submission, switches that control what BML/C2SIM dialects are to be translated and other miscellaneous values.

There is a property referencing a "Cyber Attack". This is a capability that simulates an attack on incoming messages and is controlled by a separate configuration file. A separate document will be published describing this capability.

```
# Implement simulated cyber attack
server.cyberAttack = T
```

Capturing Response Time Statistics

The server.collectResponsTime property indicates that server response time statistics are to be captured. (Note that doing this consumes server and network capacity.) The C2SIM REST client measures the response time of each transaction as it is submitted to the server. If the response from the server, e.g. :

```
<?xml version="1.0" encoding="UTF-8"?>
<result>
      <status>OK</status>
      <message>Message processed successfully</message>
      <serverInitialized>true</serverInitialized>
      <serverVersion>4.8.3.1</serverVersion>
      <sessionState>INITIALIZING</sessionState>
      <unitDatabaseName>default</unitDatabaseName>
      <unitDatabaseSize>2</unitDatabaseSize>
      <msgNumber>2</msgNumber>
      <time> 0.033</time>
      <collectResponseTime>T</collectResponseTime>
</result>
```

Contains <collectResponseTime>T</collectResponseTime> the client code will submit an additional message to the C2SIMServer/Stats URL containing the measured response time from the previous transaction. An example is shown here:

```
<?xml version="1.0" encoding="UTF-8"?>
<C2SIM_Statistics xmlns="http://www.sisostds.org/schemas/c2sim/1.0">
    <REST_ResponseTime>
        <submitterID>dsc</submitterID>
        <msgNumber>3</msgNumber>
        <startTime>2019-01-29 18:11:51,212</startTime>
        <endTime>2019-01-29 18:11:55,968</endTime>
        <elapsedTime>4.756</elapsedTime>
        <serverTime> 0.010</serverTime>
    </REST_ResponseTime>
</C2SIM_Statistics>
```

The server writes each response time message preceded with the word “Stats” to the debug log located at /home/bmluser/c2simFiles/c2simDebug. These messages can be extracted from the debug log for a particular date and analyzed.

Appendix A: Sample Code Using Java ClientLib (comparable API calls required in C++)

Sending a REST message

```
import edu.gmu.c4i.c2simclientlib2.*;

String xmlMsg = "xxxxx";
String response = "";
C2SIMClientREST_lib c2s;

// Create new C2SIMClientREST_Lib object
c2s = new C2SIMClientREST_Lib();

// Set parameters
c2s.setHost("localhost");
c2s.setSubmitter("myID");

// Send the message
try {
      response = c2s.c2simRequest(xmlMsg);// single REST transaction;
                                          // connection does not persist
    }
catch (C2SIMClientException e)
      {System.out.println("C2SIMException: " + e.getMessage() + " Cause:"
    + e.getCauseMessage());
 return;
}

// Print the result
System.out.println(response);
```

Receiving STOMP messages

```
import edu.gmu.c4i.c2simclientlib2.*;

// Create the Client Object
C2SIMClientSTOMP_Lib c = new C2SIMClientSTOMP_Lib();

// Set parameters
c.setHost("localhost");
c.setDestination("/topic/C2SIM");
c2s.addSubscription("C2SIM_Order"); // C2SIM_Command and C2SIM_Initialization
                                    // always included by library;
                                    // C2SIM_Order and C2SIM_Report
                                    // must be chosen here
C2SIMSTOMPMessage resp;

try {
   resp = c.connect();// STOMP connection persists until closed
}
catch (C2SIMClientException e)
{
   // Error during connect print message and return
   System.out.println("Error during connection to STOMP host"
       + c.getHost() + " " + e.getMessage() + " - " +  e.getCauseMessage());
   return;
}
System.out.println(resp.getMessageType();

// Start listening and loop forever
while (true) {
   try {
      resp = c.getNext_Block();
   }
   catch (C2SIMClientException e)
   {
      System.out.println("Exception while reading STOMP message "
         + e.getMessage() + " - " + e.getCauseMessage());
      return;
   }
   // Print received message
   System.out.println(resp.getMessageBody());
}
```

Subscribing to STOMP message-descriptors for C2SIM

The following code is used to subscribe to one or more categories of messages from the C2SIM schema. **It is important that applications not consuming C2SIM_Report messages not include that subscription, because the reports category typically produces the most C2SIM messages, thus subscribing to it introduces significant network and server loading.**

```
String as = "(message-selector = 'C2SIM Command')" +
                    " or (message-selector = 'C2SIM Initialization')" +
                    " or (message-selector = 'C2SIM Order')" +
                    " or (message-selector = 'C2SIM Report')";
c.addAdvSubscription(as);
```

Sending a C2SIM Message requesting a response

```
import edu.gmu.c4i.c2simclientlib2.*;

C2SIMClientREST_Lib c2s;
String xml = "xml xml xml xml";
String convID = "";

// Instantiate C2SIMClientREST object for C2SIM message
c2s = new C2SIMClientREST_Lib("C2_Host","SIM_Host", "Request");

// Remember the conversationID for the C2SIM message we are sending
convID = c2s.getC2SIMHeader().getConversationID();

// Set parameters
c2s.setHost("localhost");
c2s.setSubmitter("C2Tester");
c2s.setPath("C2SIMServer/c2sim"

// Send the message
try {
   response = c2s.c2simRequest(xml);
}
catch (C2SIMClientException e) {
   System.out.println("C2SIMException: " + e.getMessage() + " Cause:"
       + e.getCauseMessage());
}

// Received Web Services response.  Print it
System.out.println("Response to WS request: " + response);

// Open up STOMP connection to receive response from C2SIM_SIM
C2SIMClientSTOMP_Lib c = new C2SIMClientSTOMP_Lib();

// Set parameters
c.setHost("localhost");
c.setDestination("/topic/C2SIM");

// Add subscription to listen for same conversationID we just used to send
c.addAdvSubscription("message-selector = '" + convID + "'");

try {
   C2SIMStompMessage sm = c.connect();
   // Print response to connect
   System.out.println(sm.getMessageType().toString());

   // Get next message - Should be a response to the order sent via WS
   sm = c.getNext_Block();

   if (!sm.getC2SIMHeader().getPerformative().equals("Accept"))
      System.out.println("C2SIM Message not accepted");
}
catch (C2SIMClientException e) {
   System.out.println("Exception while communicating with STOMP server" + e);
```

```
}
Continue . . . .
```

Responding to a C2SIM message that requires a response

```
import edu.gmu.c4i.c2simclientlib2.C2SIMClientException;
String conversationID = "";
String order = "";
C2SIMHeader c2s;

// Create the STOMP Client Object
C2SIMClientSTOMP_Lib c = new C2SIMClientSTOMP_Lib();

// Set host
c.setHost("localhost");

// Set the topic
c.setDestination("/topic/C2SIM");

// Subscribe to get C2SIM messages
c.addAdvSubscription("protocol = 'C2SIM'");

// Connect to the STOMP server
try {
   System.out.println("Connecing to STOMP host");
   resp = c.connect();
}
catch (C2SIMClientException e) {
   System.out.println("Error during connection to STOMP server " +
       e.getMessage() + " - " + e.getCauseMessage());
   return;
}
// Start listening for an order
try {
   System.out.println("Waiting for order");
   resp = c.getNext_Block();
}
catch (C2SIMClientException e) {
   System.out.println("Exception while reading STOMP message "
      + e.getMessage() + " - " + e.getCauseMessage());
   return;
}

// Did we get a request?
if (resp.getC2SIMHeader().getPerformative().equals("Request")) {


// Get the xml order without the C2SIM Header
String order = resp.getMessageBody();

// Save the incoming C2SIM Header
C2s = resp.getC2SIMHeader();
```

```
// Send an Accept response
try {
   c.sendC2SIM_Response(resp, "Accept", "ACK");

   // Close STOMP circuit
   c.disconnect();
}
catch (C2SIMClientException e) {
   System.out.println("Exception while sending response to C2SI message"
   + e);
}
```

Note – These samples were taken from a pair of reference applications C2SIM_C2 and C2SIM_SIM. The source code for these applications is available as separate files.

Appendix B: C2SIMServer.properties file

```
# STOMP Interface setup (Apache Apollo Server)
stomp.serverHost=localhost
stomp.port=61613
stomp.topicName=/topic/C2SIM

#Location of bmlFiles ($BML_HOME)
server.bmlFiles = /home/bmluser/c2simFiles

# Location of C2SIM initialization file (ObjectInitialization)
relative to $BML_HOME (Name is supplied with LOAD command)
server.initDB = /InitializationFiles/

# Name of Schema Database
server.schema_db_name=C2SIMSchemaDB

# Password used for submission of commands controlling initialization
and server state
server.c2sim_password = v0lgenau

# Just publish the document without any other processing
server.justDocumentMode = F

# Just parse the document and publish it - This will catch structural
xml errors
server.justParseDocument = F

# Just determine what kind of message, e.g. IBMLReport and publish it.
# This will be included in the STOMP header when message is published
server.justIdentifyMessage = F

# Controls translation - This is translation to different format.  If
# serverTranslateIBML09_Order then none of the other
# orders will translated to IBML09

server.TranslateToIBML09 = F
server.TranslateToCBML = F

# Translate V9 to 1.0 or V1.0 to V9
server.Translate9To1 = F

# Is C2SIM to be translated to MSDL?
server.TranslateMSDL = F

# Capture Unit position from position reports
# In which case late-join initialization will get these positions
server.CaptureUnitPosition = T
```

```
# Implement simulated cyber attack
server.cyberAttack = F

# Request response time statistics from REST client
server.collectResponseTime = T

# Minimum client version to be accepted by this server version
server.enforceVersion = T
server.minimumClientVersion = 4.7.0.0
```

Appendix C: Requesting Late Join

C2SIM clients need initialization information to start. The server distributes this information at Coalition startup (see state diagram under C2SIM Commands above.) Sometimes clients must connect late, or reconnect, and so need a fresh delivery of the initialization. The server can provide this in response to a “QUERYINIT” request. Moreover, the initialization can optionally include latest reported unit locations in place of initial locations (this is controlled by the server properties file). An example of the required Java code is shown below. The fresh initialization message is delivered in the REST response.

```
import edu.gmu.c4i.c2simclientlib2.*;

// start REST connection using performative for Initialize
try {
      C2SIMClientREST_Lib c2simClient =  new C2SIMClientREST_Lib(
           "C2SImGUI@"+localAddress,
            serverName,
            “Initialize”,
            c2simProtocolVersion);
      c2simClient.setProtocol(c2simProtocol);
      c2simClient.setPath(c2simPath);
      c2simClient.setHost(serverName);
      c2simClient.setPort(“8080”);
}
catch(C2SIMClientException cce){
      printError("C2SIMClientException in newRESTLib:"  +
            cce.getMessage() + " cause:" + cce.getCauseMessage());
      return null;
}
c2simClient.setHost(serverName);
c2simClient.setSubmitter(bml.submitterID);
c2simClient.setPath("C2SIMServer/c2sim");

// send the QUERYINIT command – initialization will return in response
String pushShareResponseString = "";
try{
     pushShareResponseString =
     c2simClient.c2simCommand(“QUERYINIT”,bml.serverPassword,"","");
} catch (C2SIMClientException bce) {
     printError("exception pushing C2SIMserver control:" +
           bce.getMessage()+" cause:" + bce.getCauseMessage(),
          "C2SIM Server Control Push Message”);
     printError("RESPONSE:" + pushShareResponseString);
        return pushShareResponseString;
}
```