N4A Web Services Guide

User Guide

EWS0000UG002

Version:1.00

30 January, 2012

enfora®
A Novatel Wireless Company

General

TERMS OF USE OF NEW MATERIALS - PLEASE READ CAREFULLY

From time to time, Enfora, in its sole discretion, may make available for download on its website (www.enfora.com), or may transmit via mail or email, updates or upgrades to, or new releases of, the firmware, software or documentation for its products (collectively, 'New Materials'). Use of such New Materials is subject to the terms and conditions set forth below, and may be subject to additional terms and conditions as set forth in Enfora's Technical Support Policy (posted on its website) and/or any written agreement between the user and Enfora.

All New Materials are provided AS IS. Enfora makes no warranty or representation with respect to the merchantability, suitability, functionality, accuracy or completeness of any such New Materials. The user of such New Materials assumes all risk (known or unknown) of such use. Enfora reserves all rights in such New Materials. The user shall have only a revocable and limited license to use such New Materials in connection with the products for which they are intended. Distribution or modification of any New Materials without Enfora's consent is strictly prohibited.

IN NO EVENT WILL ENFORA BE RESPONSIBLE FOR ANY INCIDENTAL, INDIRECT, CONSEQUENTIAL OR SPECIAL DAMAGES AS A RESULT OF THE USE OF ANY NEW MATERIALS. ENFORA'S MAXIMUM LIABILITY FOR ANY CLAIM BASED ON THE NEW MATERIALS SHALL NOT EXCEED FIFTY U.S. DOLLARS ($50).

Copyright

251 Renner Pkwy

Richardson, TX 75080 USA

Phone: (972) 633-4400

Fax: (972) 633-4444

Email: info@enfora.com

www.enfora.com

Table of Contents

1 Web Services API

1.1 Scope

The purpose of this document is to explain the use of some of the methods available in the 3.1 N4A Web services. Customers and system integrators require a way to programmatically interact with the CMS and Provisioner systems. N4A web services were developed to provide a flexible and customizable interface for accessing data and controlling system features programmatically.

1.2 N4A-WebService Device Methods

There are several device related N4A web services available. However we will focus on just two of these methods in this document. These methods are used to retrieve events and messages for a specific device. The table below describes the purpose of each of these methods.

Method	Description
listDeviceEvents	List device events based on the specified date range, event type and max records allowed to be returned
listDeviceMessages	List device messages based on the nature of the message (probes or not) and max records allowed to be returned

1 API

1.1 Web Service API

1.1.1 Description

There are two ways to access the web service provided by N4A CMS. Users can utilize a web browser to send commands to devices. Alternatively, a well-constructed http client object can be used in the application to access the N4A CMS web service API.

1.1.2 Access from a Web Browser

Sending AT commands to devices is very straightforward using the simple web service API by entering the URL in a browser. However, all commands must be in valid URL format to be interpreted properly by the browser. Please refer to the N4A CMS User Guide (Enfora document EWS0100UG001) for the list of restricted characters and their associated URL encoding requirements.

```
http://<services_GW_IP>:<services_GW_
port>/webservice/get/webservice/get/?device=<DEVICE_ID>&cmd=<AT_COMMAND>&delay=<DELAY>
```

<services GW IP>	Services GW IP address or web address
<services GW web services port>	Web Services GW port number (default port number is 8086)
<DEVICE_ID>	Modem ID of device – case sensitive parameter
<AT_COMMAND>	AT commands to be sent to device. Special characters need to be converted to Hex equivalent value. Ex: <&w> should be sent as <%26w>
<DELAY>	Pause before message will be sent, unit = milliseconds

A unique identifier (UUID) is returned upon successfully adding the command to the transmit queue. Users may check GW database and search in the AT_REQUEST_PARENT_ID column of the AT_RESPONSE table for this UUID to retrieve the response from the device.

1.1.3 HTTP Client

Since the N4A CMS Web Service API is accessed thru the HTTP protocol, the Java HttpClient component may be of interest to anyone building customized Java applications to interact with N4A CMS Web Service API. The example shown below illustrates how to encode an AT command message in a URL object and send it to the N4A CMS thru the Web Service API.

Example:

```
import org.apache.commons.httpclient.*;
import org.apache.commons.httpclient.methods.*;
import org.apache.commons.httpclient.params.HttpMethodParams;
void send2Device(String State) {
String gwServer = "dal12345.enfora.com";
String port = "8086";
String deviceID = "1234567890";
String ATCmd = "AT$MSGSND=0,%22%7EA%5C20%5C20%22";
String Url = "http://" + gwServer + ":" + port +
"/webservice/get/webservice/get/?device=" + deviceID + "&cmd=" + ATCmd + "&delay=0";

// Create an instance of HttpClient.
HttpClient client = new HttpClient();
GetMethod method = null;

// Create a method instance.
method = new GetMethod(Url);
if (method != null) {
// Provide custom retry handler is necessary
method.getParams().setParameter(HttpMethodParams.RETRY_HANDLER,
new DefaultHttpMethodRetryHandler(3, false));
try {
// Execute the method.
int statusCode = client.executeMethod(method);

if (statusCode != HttpStatus.SC_OK) {
System.err.println("Method failed: " +
method.getStatusLine());
```

```
}

// Read the response body.
byte[] responseBody = method.getResponseBody();

// Deal with the response.
// Use caution: ensure correct character encoding and
// is not binary data
System.out.println(new String(responseBody));

} catch (HttpException e) {
System.err.println("Fatal protocol violation: " + e.getMessage());
e.printStackTrace();
} catch (IOException e) {
System.err.println("Fatal transport error: " + e.getMessage());
e.printStackTrace();
} finally {
// Release the connection.
method.releaseConnection();
}
}
else {
System.out.println("method is null");
}
}
```

1.2 JMX API

1.2.1 Description

To efficiently communicate with devices, N4A CMS offers a set of JMX APIs to allow developers to manage the deployment of services to devices, retrieve responses/results of commands sent to devices, and other tasks. For example, the JMX API sendSynchCommand can be used to deploy a service equipped with a

single command to a single device. For each N4A CMS's JMX API listed in next section, a brief description, a detailed list of signatures, return type, thrown exception (if any) and a snippet are provided.

Note: JMX uses RMI as its underlying mechanism. By default the RMI stub returned by a JMX call is bound to the host's loopback address (127.0.0.1). If the JMX API is to be used from a remote host this address should be set to an IP address that is accessible by the remote host by adding the following line to Gateway\ gateway.options (Windows) or Gateway/gateway.sh.options (Linux):

Djava.rmi.server.hostname=<IpAddress>

The N4A CMS Provisioner software uses JMX for communications to the N4A CMS, any changes to this setting need to be considered in terms of their effect on the link between the Provisioner and Gateway.

1.2.2 API SUMMARY

The following paragraphs describe the Gateway JMX APIs.

1.2.2.1 SEND SYNCH COMMAND

void sendSynchCommand(long deploymentId, boolean continueOnError, String dvcId, String command) throws GatewayException

Description: Send a single command to a single device.

<deploymentId>	A unique Deployment ID
<continueOnError>	Whether to continue if error occurs
<dvcId>	Device to receive the command
<command>	AT Command to send
<GatewayException>	Exceptions thrown from Gateway JMX Call

Code Example:

```
public static boolean JMXAPI_sendSynchCommand(String gatewayIP, String gatewayJMXPort)
{
boolean status = true;
String resultAdress = gatewayIP + ":" + gatewayJMXPort;
String urlString = "service:jmx:rmi:///jndi/rmi://" + resultAdress + "/jmxrmi";
String mbeanName = "com.enfora.ewide.gateway:type=ProvisionerHelper";
ObjectName objectName = new ObjectName(mbeanName);
```

```
try {
JMXServiceURL url = new JMXServiceURL(urlString);
JMXConnector jmxConnector = JMXConnectorFactory.connect(url, null) ;
if (jmxConnector != null) {
MBeanServerConnection mbsc = jmxConnector.getMBeanServerConnection();
//Define deployment id
long deploymentId = 100;
//Set the device to receive command
String deviceId = "1234567890";
//Set the command to send
String command = "AT$WAKUP=1,1";
Object[] params = new Object[] {deploymentId, false, deviceId, command};
String signature[] = new String[] {"long", "boolean", "java.lang.String",
"java.lang.String"};

//Making JMX Call
mbsc.invoke(objectName, "sendSynchCommand", params, signature); mbsc.close();
}
}
catch (Exception e) {
status = false;
}
return status;
}
```

1.2.2.2 Send Sync Commands Batch

void sendSynchCommandsBatch(long deploymentId, boolean continueOnError, String[] dvcIds,String[] commands) throws GatewayException

Description: Send a list of AT commands (a service) to a list of devices

<deploymentId>	A unique Deployment ID
<continueOnError>	Whether to continue if error occurs
<dvcIds>	Devices (list of device IDs) to receive the command
<commands>	A list of AT Commands to send
<GatewayException>	Exceptions thrown from GW JMX Call

Code Example:

```
public static boolean JMXAPI_sendSynchCommand(String gatewayIP, String gatewayJMXPort)
{
boolean status = true;
String resultAdress = gatewayIP + ":" + gatewayJMXPort;
String urlString = "service:jmx:rmi:///jndi/rmi://" + resultAdress + "/jmxrmi";
String mbeanName = "com.enfora.ewide.gateway:type=ProvisionerHelper";
ObjectName objectName = new ObjectName(mbeanName);

try {
JMXServiceURL url = new JMXServiceURL(urlString);
JMXConnector jmxConnector = JMXConnectorFactory.connect(url, null) ;
if (jmxConnector != null) {
MBeanServerConnection mbsc = jmxConnector.getMBeanServerConnection();
//Define deployment id
long deploymentId = 100;
//Set the device to receive command
String[] commands = {"AT$WAKEUP=1,1","AT$FRIEND=1",1,"172.16.25.3"};
String[] deviceIds = {"1234567890","2345678901"};
Object[] params = {deploymentId, false, deviceIds, commands};
String signature = {"long","boolean","[Ljava.lang.String";[Ljava.lang.String;"};
msbc.invoke(objectname, "sendSynchCommandsBatch", params, signature);
//Set the command to send
String command = "AT$WAKUP=1,1";
Object[] params = new Object[] {deploymentId, false, deviceId, command};
String signature[] = new String[] {"long", "boolean", "java.lang.String",
"java.lang.String"};
//Making JMX Call
mbsc.invoke(objectName, "sendSynchCommand", params, signature);
mbsc.close();
}
}
catch (Exception e) {
status = false;
}
```

```
return status;
}
```

1.2.2.3 Pause Commands Batch

void pauseCommandsBatch(long deploymentId)

Description: Pause a deployment

<deploymentId>	Deployment ID to pause

Code Example:

```
public static boolean JMXAPI_pauseCommandsBatch(String gatewayIP, String
gatewayJMXPort) {
boolean status = true;
String resultAdress = gatewayIP + ":" + gatewayJMXPort;
String urlString = "service:jmx:rmi:///jndi/rmi://" + resultAdress + "/jmxrmi";
String mbeanName = "com.enfora.ewide.gateway:type=ProvisionerHelper";
ObjectName objectName = new ObjectName(mbeanName);

try {
JMXServiceURL url = new JMXServiceURL(urlString);
JMXConnector jmxConnector = JMXConnectorFactory.connect(url, null) ;
if (jmxConnector != null) {
MBeanServerConnection mbsc = jmxConnector.getMBeanServerConnection();
long deploymentId = 100;
Object[] params = new Object[] { deploymentId };
String signature[] = new String[] { "long" };
//Making JMX Call
mbsc.invoke(objectName, ""pauseCommandsBatch", params, signature); mbsc.close();
}
}
catch (Exception e) {
status = false;
}
return status;
}
```

1.2.2.4 Resume Commands Batch

void resumeCommandsBatch(long deploymentId) throws GatewayException

Description: Resume a deployment

<deploymentId>	Deployment Id to resume
<GatewayException>	Exceptions thrown from Gateway JMX Cal

l

Code Example:

```
public static boolean JMXAPI_resumeCommandsBatch(String gatewayIP, String
gatewayJMXPort) {
boolean status = true;
String resultAdress = gatewayIP + ":" + gatewayJMXPort;
String urlString = "service:jmx:rmi:///jndi/rmi://" + resultAdress + "/jmxrmi";
String mbeanName = "com.enfora.ewide.gateway:type=ProvisionerHelper";
ObjectName objectName = new ObjectName(mbeanName);

try {
JMXServiceURL url = new JMXServiceURL(urlString);
JMXConnector jmxConnector = JMXConnectorFactory.connect(url, null) ;
if (jmxConnector != null) {
MBeanServerConnection mbsc = jmxConnector.getMBeanServerConnection();
long deploymentId = 100;
Object[] params = new Object[] { deploymentId };
String signature[] = new String[] { "long" };
//Making JMX Call
mbsc.invoke(objectName, ""resumeCommandsBatch", params, signature); mbsc.close();
}
}
catch (Exception e) {
status = false;
}
return status;
}
```

1.2.2.5 Stop Send Commands

void stopSendCommands(String deviceId, String[] requestsToDelete)

Description: Stop the send-command process for a list of command requests

<deviceId>	Device to which commands are being sent
<requestsToDelete>	List of Send Command Requests to remove

Code Example:

```
public static boolean JMXAPI_stopSendCommands(String gatewayIP, String gatewayJMXPort)
{
boolean status = true;
String resultAdress = gatewayIP + ":" + gatewayJMXPort;
String urlString = "service:jmx:rmi:///jndi/rmi://" + resultAdress + "/jmxrmi";
String mbeanName = "com.enfora.ewide.gateway:type=ProvisionerHelper";
ObjectName objectName = new ObjectName(mbeanName);

try {
JMXServiceURL url = new JMXServiceURL(urlString);
JMXConnector jmxConnector = JMXConnectorFactory.connect(url, null) ;
if (jmxConnector != null) {
MBeanServerConnection mbsc = jmxConnector.getMBeanServerConnection();
String deviceId = "1234567890";
String reqIdsToDelete[] = getRequestIds();
Object[] params = new Object[] { deviceId, reqIdsToDelete };
String signature[] = new String[] { "java.lang.String" ,
"[Ljava.lang.String"};
//Making JMX Call
mbsc.invoke(objectName, ""stopSendCommands", params, signature); mbsc.close();
}
}
catch (Exception e) {
```

```
status = false;
}
return status;
}
```

1.2.2.6 Get Response Results

Map<String, String> getResponseResults(String[] requestIds)

Description: Retrieve the results of AT commands sent to a device

<requestIds>	List of Send Command Request IDs

Code Example:

```
public static boolean JMXAPI_getResponseResults(String gatewayIP, String
gatewayJMXPort) {
boolean status = true;
String resultAdress = gatewayIP + ":" + gatewayJMXPort;
String urlString = "service:jmx:rmi:///jndi/rmi://" + resultAdress + "/jmxrmi";
String mbeanName = "com.enfora.ewide.gateway:type=ProvisionerHelper";
ObjectName objectName = new ObjectName(mbeanName);

try {
JMXServiceURL url = new JMXServiceURL(urlString);
JMXConnector jmxConnector = JMXConnectorFactory.connect(url, null) ;
if (jmxConnector != null) {
MBeanServerConnection mbsc = jmxConnector.getMBeanServerConnection();
String requestIds[] = getRequestIds();
Object[] params = new Object[] { requestIds };
String signature[] = new String[] { "String[]” };
//Making JMX Call
Map<String, String> response = mbsc.invoke(objectName, "getResponseResults”, params,
signature);
mbsc.close();
}
}
catch (Exception e) {
```

```
status = false;
}
return status;
}
```

1.2.2.7 Get Command Batch Result

String getCommandBatchResult(long deploymentId)

Description: Get deployment status from Gateway

<deploymentId>	Deployment ID

Code Example:

```
public static boolean JMXAPI_getCommandBatchResult(String gatewayIP, String
gatewayJMXPort) {
boolean status = true;
String resultAdress = gatewayIP + ":" + gatewayJMXPort;
String urlString = "service:jmx:rmi:///jndi/rmi://" + resultAdress + "/jmxrmi";
String mbeanName = "com.enfora.ewide.gateway:type=ProvisionerHelper";
ObjectName objectName = new ObjectName(mbeanName);

try {
JMXServiceURL url = new JMXServiceURL(urlString);
JMXConnector jmxConnector = JMXConnectorFactory.connect(url, null) ;
if (jmxConnector != null) {
MBeanServerConnection mbsc = jmxConnector.getMBeanServerConnection();
long deploymentId = 100;
Object[] params = new Object[] { deploymentId };
String signature[] = new String[] { "long” };
//Making JMX Call
String result = mbsc.invoke(objectName, "” getCommandBatchResult”, params, signature);
mbsc.close();
}
}
catch (Exception e) {
status = false;
}
```

```
return status;
}
```

1.2.2.8 Get Device Result

String getDeviceResult(long deploymentId, String dvcId)

Description Retrieve the deployment status for a particular device

<deploymentId>	Deployment ID
<dvcId>	Device ID

Code Example:

```
public static boolean JMXAPI_getDeviceResult(String gatewayIP, String gatewayJMXPort) {
boolean status = true;
String resultAdress = gatewayIP + ":" + gatewayJMXPort;
String urlString = "service:jmx:rmi:///jndi/rmi://" + resultAdress + "/jmxrmi";
String mbeanName = "com.enfora.ewide.gateway:type=ProvisionerHelper";
ObjectName objectName = new ObjectName(mbeanName);

try {
JMXServiceURL url = new JMXServiceURL(urlString);
JMXConnector jmxConnector = JMXConnectorFactory.connect(url, null) ;
if (jmxConnector != null) {
MBeanServerConnection mbsc = jmxConnector.getMBeanServerConnection();
long deploymentId = 100;
String deviceId = "1234567890";
Object[] params = new Object[] { deploymentId, deviceId };
String signature[] = new String[] { "long", "java.lang.String" };
//Making JMX Call
String result = mbsc.invoke(objectName, ""getDeviceResult", params, signature);
mbsc.close();
}
}
catch (Exception e) {
```

```
status = false;
}
return status;
}
```

1.2.2.9

Delete Commands Batch

void deleteCommandsBatch(long deploymentId)

Description Stop and remove the deployment

<deploymentId>	Deployment ID

Code Example:

```
public static boolean JMXAPI_getResponseResults(String gatewayIP, String
gatewayJMXPort) {
boolean status = true;
String resultAdress = gatewayIP + ":" + gatewayJMXPort;
String urlString = "service:jmx:rmi:///jndi/rmi://" + resultAdress + "/jmxrmi";
String mbeanName = "com.enfora.ewide.gateway:type=ProvisionerHelper";
ObjectName objectName = new ObjectName(mbeanName);

try {
JMXServiceURL url = new JMXServiceURL(urlString);
JMXConnector jmxConnector = JMXConnectorFactory.connect(url, null) ;
if (jmxConnector != null) {
MBeanServerConnection mbsc = jmxConnector.getMBeanServerConnection();
long deploymentId = 100;
Object[] params = new Object[] { deploymentId };
String signature[] = new String[] { "long” };
//Making JMX Call
mbsc.invoke(objectName, ""”deleteCommandsBatch”, params, signature);
mbsc.close();
}
}
catch (Exception e) {
status = false;
```

```
}
return status;
}
```

2 Generating Run-time Artifacts

2.1 WSDL

The device web services are defined in a WSDL (Web Services Description Language). WSDL is a document written in XML and it describes a Web service. It specifies the location of the service and the operations (or methods) the service exposes.

2.2 Generate Run-Time Artifacts from WSDL

There are many ways to generate a web service client from a provided WSDL. In this document we show how to use the JAX-WS reference implementation for generating the run-time artifacts from the WSDL.

The normal way to retrieve the WSDL content is to connect to the server. Assuming the N4A web services are deployed in JBoss under Provisioner on your local box, a full list of available services should be displayed by typing the URL http://localhost:8087/n4a-webservice in the browser. The corresponding WSDL link for each service is located on the right. The N4ADevice web service WSDL is found at http://localhost:8087/n4a-webservice/N4ADevice?wsdl". This is the parameter you need to specify when you run the JAX-WS "wsimport" tool to generate the run-time artifacts. Once you have installed JAX-WS you should be able to run the wsimport command. He following example shows how to generate the run-time artifacts for the N4A Device web services.

```
wsimport -p <your chosen package name> –s src –d bin –XadditionalHeaders
http://localhost:8087/n4a-webservice/N4ADevice?wsdl
```

After running the "wsimport" command, the run-time artifacts for the web service should be generated under the src and bin directories.

2.3 Web service methods

Once the run-time artifacts are created, you can see that there are several device related web service methods available. In this document, we only focus on two of these methods.

2.3.1 listDeviceEvents method

The input parameters for listDeviceEvents method are listed below:

Input Parameter	Data Type	Description

DeviceID	String	Unique device identifier
StartDate	XMLGregorianCalendar	Starting date for the created events in range
EndDate	XMLGregorianCalendar	Ending date for the created events in range
EventType	integer	Event type
Max	Integer	Maximum records to be returned

This method returns a list of DeviceEvent objects. The attributes of the DeviceEvent object are listed below:

Field	Data Type
EventId	String
DeviceId	String
RawDataId	String
CreateDate	XMLGregorianCalendar
UserNumberParam1	int
FormatCodeParam2	int
DeviceEventId	int
DeviceEventSeqNo	long
AD1	int
AD2	int
RTC	XMLGregorianCalendar
GPSStatus	int
GPSLatitude	double
GPSLongitude	double
GPSSpeed	double
GPSHeading	double
GPSSatCount	int
EMInfo	int
EMMcc	int
EMMnc	int
EMLac	int
EMCid	int
EMArfcn	int
EMRxlev	int
EMTav	int
EMNc0Lac	int
EMNc0Cid	int
EMNc0Arfcn	int
EMNc0Rxlev	int
EMNc1Lac	int
EMNc1Cid	int

EMNc1Arfcn	int
EMNc1Rxlev	int
EMNc2Lac	int
EMNc2Cid	int
EMNc2Arfcn	int
EMNc2Rxlev	int
EMNc3Lac	int
EMNc3Cid	int
EMNc3Arfcn	int
EMNc3Rxlev	int
EMNc4Lac	int
EMNc4Cid	int
EMNc4Arfcn	int
EMNc4Rxlev	int
EMNc5Lac	int
EMNc5Cid	int
EMNc5Arfcn	int
EMNc5Rxlev	int
LBSSource	String
LBSHaccuracy	int
LBSVaccuracy	int
LBSLatitude	double
LBSLongitude	double
LBSStatus	int

If there is any OBD2 information associated with the event, it is attached as part of the device event object. The attributes for OBD2 event are listed below:

Field	**Data Type**
obd2EventId	String
obd2InstFuelRate	int
obd2InstFuelEcon	int
obd2FuelLevel	int
obd2TripFuelEcon	int
obd2Vin	String
obd2Protocol	int
obd2FwVer	int
obd2RSSI	int
obd2MilCnt	int
obd2MilCode1	int
obd2MilCode2	int

obd2MilCode3	int
obd2MilCode4	int
obd2MilCode5	int
obd2MilCode6	int
obd2MilCode7	int
obd2MilCode8	int
obd2MilCode9	int
obd2MilCode10	int
obd2MilCode11	int
obd2MilCode12	int
obd2TripOdometer	int

2.3.2 listDeviceMessages method

The input parameters for listDeviceMessages method are listed below:

Input Parameter	Data Type	Description
DeviceID	String	Unique device identifier
ShowProbesSwitch	Boolean	Whether to show probing related at requests
ShowServices	Boolean	Whether to show services related at requests
Max	integer	Maximum records to be returned

This method returns a list of DeviceMessage objects. The attributes of the DeviceMessage object are listed below:

Field	Data Type
RequestId	String
DeviceId	String
RawDataId	String
Command	String
SequenceNo	long
CreatedDate	XMLGregorianCalendar

3 Developing A Client Application

Once the run-time artifacts are generated, we can now develop a client application. The generated artifacts provide the methods for calling the web services as well as the objects that will be returned. The following sample code is a simple web client that calls both methods and simply prints out some attributes of the returned objects. It can be used as a starting point for a more complete application. The import statements may differ depending on the package you used for the wsimport command. The values for deviceID, startDate, endDate, and eventType should correspond to valid values for your instance of CMS.

```
package com.enfora.webservice.client;
import com.enfora.webservice.client.N4ADevice;
import com.enfora.webservice.client.N4ADeviceImplService;
import com.enfora.webservice.client.Device;
import com.enfora.webservice.client.ListDeviceEventsFailedFaultException;
import com.enfora.webservice.client.ListDeviceMessagesFailedFaultException;

/**
*
* @author
* /
public class N4AWebServiceClient {
/* Demonstrate how the service is instantiated and the web method invoked
*
*/
public void callListDeviceWebService() {
// Create our handler resolver that adds the authentication information
HeaderHandlerResolver handlerResolver = new HeaderHandlerResolver();
// Setup the device webservice to get device events
N4ADeviceImplService deviceService = new N4ADeviceImplService();
deviceService.setHandlerResolver(handlerResolver);
N4ADevice devicePort = deviceService.getN4ADeviceImplPort();
try {
String deviceID = "moo1";
Date end = new Date();
GregorianCalendar gc = new GregorianCalendar();
```

```
gc.setTime(end);
DatatypeFactory df = DatatypeFactory.newInstance();
XMLGregorianCalendar endDate = df.newXMLGregorianCalendar(gc);
gc.add(Calendar.DAY_OF_YEAR, -1);
XMLGregorianCalendar startDate = df.newXMLGregorianCalendar(gc);
int eventType = 21000;
int max = 5;

// Get the events for the device
DeviceEventList eventList = devicePort.listDeviceEvents(deviceID, startDate, endDate,
eventType, max);
List<DeviceEvent> events = eventList.getDeviceEvent();
// Print out the event GPS data
for(DeviceEvent event : events)
{
System.out.println(event.getGPSLatitude());
System.out.println(event.getGPSLongitude());
}

// Get the messages for the device
DeviceMessageList msgList = devicePort.listDeviceMessages(deviceID, true, true, max);
List<DeviceMessage> msgs = msgList.getDeviceMessage();

// Print out the messages
for(DeviceMessage msg : msgs)
{
System.out.println(msg.getRequestID());
}
}
catch (SOAPFaultException e) {
e.printStackTrace();
}
catch (DatatypeConfigurationException e) {
e.printStackTrace();
}
```

```
catch (ListDeviceEventsFailedFaultException e) {
e.printStackTrace();
}
catch (ListDeviceMessagesFailedFaultException e) {
e.printStackTrace();
}
}
public static void main(String[] args) {
new N4AWebServiceClient().callListDeviceWebService();
}
}
```

The HeaderHandler is used to add the authentication information (username and password) into the request header. HeaderHandler is created to implement the SOAPHandler interface. HeaderHandlerResolver is the class that decides what order the handlers should be called. This code for these classes is shown below. Obviously the values you should use for the username and password should match valid credentials for your instance of CMS.

```
package com.enfora.webservice.client;
import java.util.Set;
import javax.xml.namespace.QName;
import javax.xml.soap.SOAPElement;
import javax.xml.soap.SOAPEnvelope;
import javax.xml.soap.SOAPHeader;
import javax.xml.soap.SOAPMessage;
import javax.xml.ws.handler.MessageContext;
import javax.xml.ws.handler.soap.SOAPHandler;
import javax.xml.ws.handler.soap.SOAPMessageContext;
/**
* @author
*/
public class HeaderHandler implements SOAPHandler<SOAPMessageContext> {
public boolean handleMessage(SOAPMessageContext smc) {
Boolean outboundProperty = (Boolean) smc.get(MessageContext.MESSAGE_OUTBOUND_PROPERTY);
if (outboundProperty.booleanValue()) {
SOAPMessage message = smc.getMessage();
try {
```

```
SOAPEnvelope envelope = smc.getMessage().getSOAPPart().getEnvelope();
SOAPHeader header = envelope.addHeader();
SOAPElement security = header.addChildElement("Security", "wsse", "http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd");
SOAPElement usernameToken = security.addChildElement("UsernameToken", "wsse");
usernameToken.addAttribute(new QName("xmlns:wsu"), "http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd");
SOAPElement username = usernameToken.addChildElement("Username", "wsse");
username.addTextNode("username");
SOAPElement password = usernameToken.addChildElement("Password", "wsse");
password.setAttribute("Type", "http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0#PasswordText");
password.addTextNode("password");

//Print out the outbound SOAP message
message.writeTo(System.out);

} catch (Exception e) {
e.printStackTrace();
}
} else {
try {
//This handler does nothing with the response from the Web Service so
//we just print out the SOAP message.
SOAPMessage message = smc.getMessage();
message.writeTo(System.out);

} catch (Exception ex) {
ex.printStackTrace();
}
}
return outboundProperty;
}
public Set getHeaders() {
//throw new UnsupportedOperationException("Not supported yet.");
return null;
```

```
}
public boolean handleFault(SOAPMessageContext context) {
//throw new UnsupportedOperationException("Not supported yet.");
return true;
}
public void close(MessageContext context) {
//throw new UnsupportedOperationException("Not supported yet.");
}
}
package com.enfora.webservice.client;
import java.util.ArrayList;
import java.util.List;
import javax.xml.ws.handler.Handler;
import javax.xml.ws.handler.HandlerResolver;
import javax.xml.ws.handler.PortInfo;
/**
* @author
*/
public class HeaderHandlerResolver implements HandlerResolver {

public List<Handler> getHandlerChain(PortInfo portInfo) {
List<Handler> handlerChain = new ArrayList<Handler>();
HeaderHandler hh = new HeaderHandler();
handlerChain.add(hh);
return handlerChain;
}
}
```

4 Device Webservice WSDL

While the WSDL is available from the web service deployment it is provided here as well.

```
<?xml version='1.0' encoding='UTF-8'?><wsdl:definitions name="N4ADeviceImplService"
targetNamespace="http://device.webservice.enfora.com/"
xmlns:ns1="http://webservice.enfora.com/"
xmlns:ns2="http://cxf.apache.org/bindings/xformat"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:tns="http://device.webservice.enfora.com/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<wsdl:import location="http://localhost:8087/n4a-
webservice/N4ADevice?wsdl=N4ADevice.wsdl" namespace="http://webservice.enfora.com/">
</wsdl:import>
<wsdl:binding name="N4ADeviceImplServiceSoapBinding" type="ns1:N4ADevice">
<soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http" />
<wsdl:operation name="addDevices">
<soap:operation soapAction="" style="document" />
<wsdl:input name="addDevices">
<soap:body use="literal" />
</wsdl:input>
<wsdl:output name="addDevicesResponse">
<soap:body use="literal" />
</wsdl:output>
<wsdl:fault name="AddDevicesFailedFault_Exception">
<soap:fault name="AddDevicesFailedFault_Exception" use="literal" />
</wsdl:fault>
</wsdl:operation>
<wsdl:operation name="listDeviceMessages">
<soap:operation soapAction="" style="document" />
<wsdl:input name="listDeviceMessages">
<soap:body use="literal" />
</wsdl:input>
<wsdl:output name="listDeviceMessagesResponse">
<soap:body use="literal" />
</wsdl:output>
```

```
<wsdl:fault name="ListDeviceMessagesFailedFault_Exception">
<soap:fault name="ListDeviceMessagesFailedFault_Exception" use="literal" />
</wsdl:fault>
</wsdl:operation>
<wsdl:operation name="listDevices">
<soap:operation soapAction="" style="document" />
<wsdl:input name="listDevices">
<soap:body use="literal" />
</wsdl:input>
<wsdl:output name="listDevicesResponse">
<soap:body use="literal" />
</wsdl:output>
<wsdl:fault name="ListDevicesFailedFault_Exception">
<soap:fault name="ListDevicesFailedFault_Exception" use="literal" />
</wsdl:fault>
</wsdl:operation>
<wsdl:operation name="listDeviceEvents">
<soap:operation soapAction="" style="document" />
<wsdl:input name="listDeviceEvents">
<soap:body use="literal" />
</wsdl:input>
<wsdl:output name="listDeviceEventsResponse">
<soap:body use="literal" />
</wsdl:output>
<wsdl:fault name="ListDeviceEventsFailedFault_Exception">
<soap:fault name="ListDeviceEventsFailedFault_Exception" use="literal" />
</wsdl:fault>
</wsdl:operation>
<wsdl:operation name="readDevice">
<soap:operation soapAction="" style="document" />
<wsdl:input name="readDevice">
<soap:body use="literal" />
</wsdl:input>
<wsdl:output name="readDeviceResponse">
<soap:body use="literal" />
```

```
</wsdl:output>
<wsdl:fault name="ReadDeviceFailedFault_Exception">
<soap:fault name="ReadDeviceFailedFault_Exception" use="literal" />
</wsdl:fault>
</wsdl:operation>
<wsdl:operation name="listDeviceTypes">
<soap:operation soapAction="" style="document" />
<wsdl:input name="listDeviceTypes">
<soap:body use="literal" />
</wsdl:input>
<wsdl:output name="listDeviceTypesResponse">
<soap:body use="literal" />
</wsdl:output>
<wsdl:fault name="ListDeviceTypesFailedFault_Exception">
<soap:fault name="ListDeviceTypesFailedFault_Exception" use="literal" />
</wsdl:fault>
</wsdl:operation>
</wsdl:binding>
<wsdl:service name="N4ADeviceImplService">
<wsdl:port binding="tns:N4ADeviceImplServiceSoapBinding" name="N4ADeviceImplPort">
<soap:address location="http://localhost:8087/n4a-webservice/N4ADevice" />
</wsdl:port>
</wsdl:service>
</wsdl:definitions>
```

5 N4A Web Services

5.1 Using N4A Web Services

In order to access n4a web services, client-side source code must be generated from the published WSDL file. The client application can then be developed based on the generated source files to invoke proper method calls that are available in the server. There are handful web service frameworks such as Axis, CXF that provide you the tool to generate client-side source code. To make this example plain and easy to follow, here shows only the utilization of the tool "wsimport" from JAX-WS API's reference implementation.

Assuming the n4a web services are deployed in JBoss under Provisioner in the local box, we can start to work on the creation of the client-side java source code using "wsimport". Each step will be explained in details along with related source code.

5.2 Create Client-side Code from WSDL

The normal way to get the WSDL content is to connect to the server (e.g., http://localhost:8087/n4a-webservice) By typing the URL listed above, a full list of available services should be displayed and the corresponding WSDL link is located on the right for each service. WSDL content can be retrieved by clicking on the link. Take N4ACompany service as an example, the URL shown on the browser such as http://localhost:8087/n4a-webservice/N4ACompany?wsdl is the wsdl location you need to specify last when you run "wsimport" to generate client code.

```
wsimport.exe -verbose -p com.enfora.webservice.client –s src –d bin -XadditionalHeaders
http://localhost:8087/n4a-webservice/N4ACompany?wsdl


Following is the list of the client-side source files generated from running "wsimport"
tool.

C:\stsworkspace\WebServiceClient\N4AClientTest\src\com\enfora\webservice\client>dir

Volume in drive C is OS

Volume Serial Number is B845-7168

Directory of
C:\stsworkspace\WebServiceClient\N4AClientTest\src\com\enfora\webservice\client

08/18/2011 03:09 PM <DIR> .

08/18/2011 03:09 PM <DIR> ..

08/18/2011 12:39 PM 6,265 Company.java

08/18/2011 12:39 PM 2,529 CompanyList.java
```

```
08/18/2011 12:39 PM 1,482 CreateCompany.java
08/18/2011 12:39 PM 1,532 CreateCompanyFailedFault.java
08/18/2011 12:39 PM 1,274 CreateCompanyFailedFaultException.java
08/18/2011 12:39 PM 1,520 CreateCompanyResponse.java
08/18/2011 12:39 PM 1,497 DeleteCompany.java
08/18/2011 12:39 PM 1,532 DeleteCompanyFailedFault.java
08/18/2011 12:39 PM 1,274 DeleteCompanyFailedFaultException.java
08/18/2011 12:39 PM 1,520 DeleteCompanyResponse.java
08/18/2011 12:39 PM 1,186 ListCompanies.java
08/18/2011 12:39 PM 1,532 ListCompaniesFailedFault.java
08/18/2011 12:39 PM 1,274 ListCompaniesFailedFaultException.java
08/18/2011 12:39 PM 1,593 ListCompaniesResponse.java
08/18/2011 12:39 PM 4,225 N4ACompany.java
08/18/2011 12:39 PM 2,631 N4ACompanyImplService.java
08/31/2011 03:39 PM 1,539 N4AWebServiceClient.java
08/18/2011 12:39 PM 13,025 ObjectFactory.java
08/18/2011 12:39 PM 122 package-info.java
08/18/2011 12:39 PM 1,489 ReadCompany.java
08/18/2011 12:39 PM 1,524 ReadCompanyFailedFault.java
08/18/2011 12:39 PM 1,256 ReadCompanyFailedFaultException.java
08/18/2011 12:39 PM 1,515 ReadCompanyResponse.java
08/18/2011 12:39 PM 1,478 ResetCompany.java
08/18/2011 12:39 PM 1,528 ResetCompanyFailedFault.java
08/18/2011 12:39 PM 1,265 ResetCompanyFailedFaultException.java
08/18/2011 12:39 PM 1,516 ResetCompanyResponse.java
27 File(s) 59,123 bytes
2 Dir(s) 397,674,098,688 bytes free
C:\stsworkspace\WebServiceClient\N4AClientTest\src\com\enfora\webservice\client>
```

5.3 Build Client Application

After running the “wsimport” command, client-side java source code along with compiled class files are generated and located under src and bin directories respectively. Those class files are the ones to be incorporate with your client application to invoke web service call. Use the following code as example to create your client application code:

```
package com.enfora.webservice.client;
import com.enfora.webservice.client.N4ACompany;
import com.enfora.webservice.client.N4ACompanyImplService;
import com.enfora.webservice.client.Company;
import com.enfora.webservice.client.ReadCompanyFailedFaultException;
import com.sun.xml.ws.developer.WSBindingProvider;
import javax.xml.ws.soap.SOAPFaultException;
/**
*
* @author www.enfora.com.webservice.client
*/
public class N4AWebServiceClient {
/*
* Demonstrate how the service is instantiated and how the web method is invoked
*
* @param
* @return
*/
public void callWebService() {
try {
//Adding authentication information into soap header
HeaderHandlerResolver handlerResolver = new HeaderHandlerResolver();
//Set up soap handler
service.setHandlerResolver(handlerResolver);

N4ACompany port = service.getN4ACompanyImplPort();
//Invoke readCompany web method by passing in company id 3
Company company = port.readCompany("3");

System.out.println("Name=" + company.getName());
System.out.println("Addr1=" + company.getAddr1());
System.out.println("Addr2=" + company.getAddr2());
System.out.println("City=" + company.getCity());
System.out.println("State=" + company.getState());
System.out.println("Zip=" + company.getZip());
```

```
System.out.println("Email=" + company.getEmail());
}
catch(ReadCompanyFailedFaultException e) {
System.out.println(e.getFaultInfo().getReturn());
}
catch (SOAPFaultException e) {
System.out.println(e.getFault().getFaultString());
}
}
public static void main(String[] args) {
new N4AWebServiceClient().callWebService();
}
}
```

Additional code to set authentication information (username and password) into soap header is also required. HeaderHandler is created to implement the SOAPHandler interface. HeaderHandlerResolver is the class decides what and in what order the handlers should be called.

```
package com.enfora.webservice.client;
import java.util.Set;
import javax.xml.namespace.QName;
import javax.xml.soap.SOAPElement;
import javax.xml.soap.SOAPEnvelope;
import javax.xml.soap.SOAPHeader;
import javax.xml.soap.SOAPMessage;
import javax.xml.ws.handler.MessageContext;
import javax.xml.ws.handler.soap.SOAPHandler;
import javax.xml.ws.handler.soap.SOAPMessageContext;
/**
* @author
*/
public class HeaderHandler implements SOAPHandler<SOAPMessageContext> {
public boolean handleMessage(SOAPMessageContext smc) {
Boolean outboundProperty = (Boolean) smc.get(MessageContext.MESSAGE_OUTBOUND_PROPERTY);
if (outboundProperty.booleanValue()) {
SOAPMessage message = smc.getMessage();
try {
```

```
SOAPEnvelope envelope = smc.getMessage().getSOAPPart().getEnvelope();
SOAPHeader header = envelope.addHeader();
SOAPElement security = header.addChildElement("Security", "S");
SOAPElement username = security.addChildElement("Login");
username.addTextNode("root");
SOAPElement password = security.addChildElement("Password");
password.addTextNode("root");

//Print out the outbound SOAP message
// message.writeTo(System.out);

} catch (Exception e) {
e.printStackTrace();
}
} else {
try {
//This handler does nothing with the response from the Web Service so
//we just print out the SOAP message.
SOAPMessage message = smc.getMessage();
// message.writeTo(System.out);

} catch (Exception ex) {
ex.printStackTrace();
}
}
return outboundProperty;
}
public Set getHeaders() {
//throw new UnsupportedOperationException("Not supported yet.");
return null;
}
public boolean handleFault(SOAPMessageContext context) {
//throw new UnsupportedOperationException("Not supported yet.");
return true;
}
```

```
public void close(MessageContext context) {
//throw new UnsupportedOperationException("Not supported yet.");
}
}
package com.enfora.webservice.client;
import java.util.ArrayList;
import java.util.List;
import javax.xml.ws.handler.Handler;
import javax.xml.ws.handler.HandlerResolver;
import javax.xml.ws.handler.PortInfo;
/**
* @author
*/
public class HeaderHandlerResolver implements HandlerResolver {

public List<Handler> getHandlerChain(PortInfo portInfo) {
List<Handler> handlerChain = new ArrayList<Handler>();
HeaderHandler hh = new HeaderHandler();
handlerChain.add(hh);
return handlerChain;
}
}
```

5.4 Compile Client Application

Use the following command to compile the code:

```
C:\stsworkspace\WebServiceClient\N4AClientTest\src>javac
.\com\enfora\webservice\client\*.java
```

5.5 Running Client Application

Use the following command to run the client:

```
C:\stsworkspace\WebServiceClient\N4AClientTest\src>java
com.enfora.webservice.client.N4AWebServiceClient
```

This is what the soap header looks like in the outbound call:

```
<S:Header>
<wsse:Security xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd">
<wsse:UsernameToken xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd">
<wsse:Username>root</wsse:Username>
<wsse:Password Type="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0#PasswordText">root</wsse:Password>
</wsse:UsernameToken>
</wsse:Security>
</S:Header>
```

Here are the outputs from running the client:

```
Name=MyCompany_3
Addr1=251 Renner Pkwy
Addr2=?
City=Richardson
State=TX
Zip=75080
Email=johndoe@enfora.com
```

We now have demonstrated the readCompany call from N4ACompany web service. By calling this service and pass in company id as 3, a company object is returned. In the similar fashion, you can invoke n4a web services that are available in the server by changing the operation and the input parameters.

6 Configuring JBoss for HTTPS

This appendix explains the process for the HTTPS configuration using a self-signed certificate on a JBoss server.

6.1 Self-signed certificate on JBoss

The instructions assume a new JBoss installation.

1. Identify the hostname for the computer hosting the server. (These instructions will use "myHostname" as an example. Replace myHostname with the actual host name when following these instructions.)
2. Identify the jBoss server type (all, default, production). I.e.: default for these instructions

Note: JBoss recommends using the same file as both keystore and trustore. This will be server.keystore. In a new installation there should be noserver.keystore in the default/conf folder. If you have one, you must decide whether to delete it (to use these instructions) or whether to adapt the instructions to suit your situation.

6.1.1 Creating the keystore and private key

1. Open a command prompt or shell and go to the default/conf folder.

2. Type the following:

```
keytool -genkey -alias jbosskey -keypass changeit -keyalg RSA -keystore server.keystore
```

3. Answer the prompts. Use myHostname when asked for first/last name. This is critical.

server.keystore is generated.

4. Type the following:

```
keytool -list -keystore server.keystore
```

You should see the PrivateKeyEntry named jbosskey in the listing.

6.1.2 Generating and storing the certificate

1. Type the following:

```
keytool -export -alias jbosskey -keypass changeit -file server.crt -keystore
server.keystore
```

server.crt is generated.

2. Type the following:

```
keytool -import -alias jbosscert -keypass changeit -file server.crt -keystore
server.keystore
```

Note: You receive a warning that it already exists in the keystore. Ignore this warning. It is because Java expects separate keystore and trustore files and we are using only one.

3. Type the following:

```
keytool -list -keystore server.keystore
```

You should see a TrustedCertEntry named jbosscert in the listing.

6.1.3 Server launch configuration

Ensure that you start the server with:

```
-c default -b 0.0.0.0 -Djavax.net.ssl.trustStore=
"<C:/yourServerLocation>/server/default/conf/server.keystore"
```

Note:
-c specfies your server type
-b is required to use the server as anything but localhost, with a server name if you only have 1 network card, with 0.0.0.0 if you have multiple network cards
-Djavax.net.ssl.trustStore specifies the location of your truststore.

- In Windows you may place these parameters in a shortcut you use to execute run.bat.
- In Unix you may place them in your startup script.
- In Eclipse, RAD or any other Eclipse-derivative your best bet is to use the jBossTools plugin.

1. Go to the jBossServer view
2. Double-click on the server
3. Verify that your hostname is set to myHostname
4. Click OpenLaunchConfiguration
5. Add to the program arguments.
6. Enable JBoss' Tomcat for HTTPS:
7. Edt "<C:/yourServerLocation>/server/default/deploy/jboss-web.deployer/server.xml" as follows:
    1. Uncomment the section that begins with <Connector port="8443"
    2. At the end of the section (but still inside of it) add:
        - keystoreFile="/conf/server.keystore"
        - keystorePass="changeit"

6.1.4 Post-setup tests

All of these test should succeed. If they fail, check to ensure that you properly completed the previous steps. Your browser will warn you about untrusted sites/certificates - this is OK, you are using a self-signed certificate. To get rid of the warnings you must get a certificate from a certificate authority.

- Vanilla access to jBoss' home page: _http://myHostname:8080_
- HTTPS access to jBoss' home page: _https://myHostname:8443_
- Vanilla access to a non-CAS application: _http://myHostname:8080/myApp_
- HTTPS access to a non-CAS application: https://myHostname:8443/myApp
- Vanilla access to CAS:
- _http://myHostname:8080/login_

- _http://myHostname:8080/logout_
- Access a CAS-enabled application:
- _http://myCAS-enabledApp_
- Unprotected pages should be accessible without going to CAS.
- You should be redirected to the CAS login the first time you access a protected page. After the login you should reach the page.
- Subsequent accesses to protected pages should not redirect you to the CAS login unless you time out or close your browser.