(Java SDK) Getting Started

Monnit Corporation

Version 2.1.0

Table of Contents

Who this Software Development Kit (SDK) is for

This is an SDK developed for use with the Monnit Hardware product line found at www.monnit.com. Proficiency with Object Oriented Programing (OOP) in Java is recommended.

1 GETTING STARTED

The Mine Java Library (MJL) and MineGUI Sample Application (MGSA) can be downloaded from http://mine.imonnit.com/. To point a gateway to the sample application requires the gateway to be unlocked. If your gateway isn't unlocked you can purchase an unlock code from the Monnit store www.monnit.com.

Your unlock code will be sent by email. You can find information on unlocking your gateway and pointing to a custom server at www.monnit.com/support/. After your gateway is unlocked you will want to become familiar with the Sample Application before creating your own Application.

2 START THE SAMPLE APPLICATION

In the Mine Java zip file that you downloaded you will have a MineGUI folder. Navigate to the MineGUI folder and open the dist folder. You will have two items, the lib folder and the MineGUI.jar. Open the MineGUI.jar to begin the sample application.

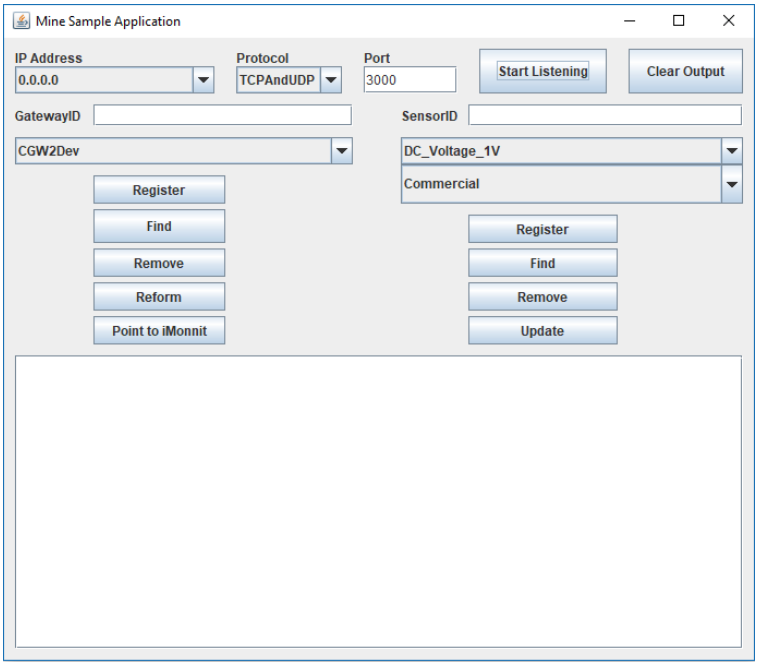
With the MGSA now running you will want to choose 0.0.0.0 for the IP Address. This will apply the localhost address to the server. Choose the type of protocol you would like to use, and the port number. The default port number we use is 3000. You can now click Start Listening button to start the server.

You will want to make sure to add an exception to any existing firewalls, so that the port that you choose is available. Making sure to also include whichever protocols you are using as well.

3 REGISTER A GATEWAY

Monnit Gateways allow sensors to communicate with your server. To be able to get the MGSA to start collecting data you must first register a gateway.

To register a Gateway you will need the gateway id and the gateway type. The MGSA provides a dropdown with every gateway type available in the MJL. Next click the Register button that is underneath the gateway type dropdown.

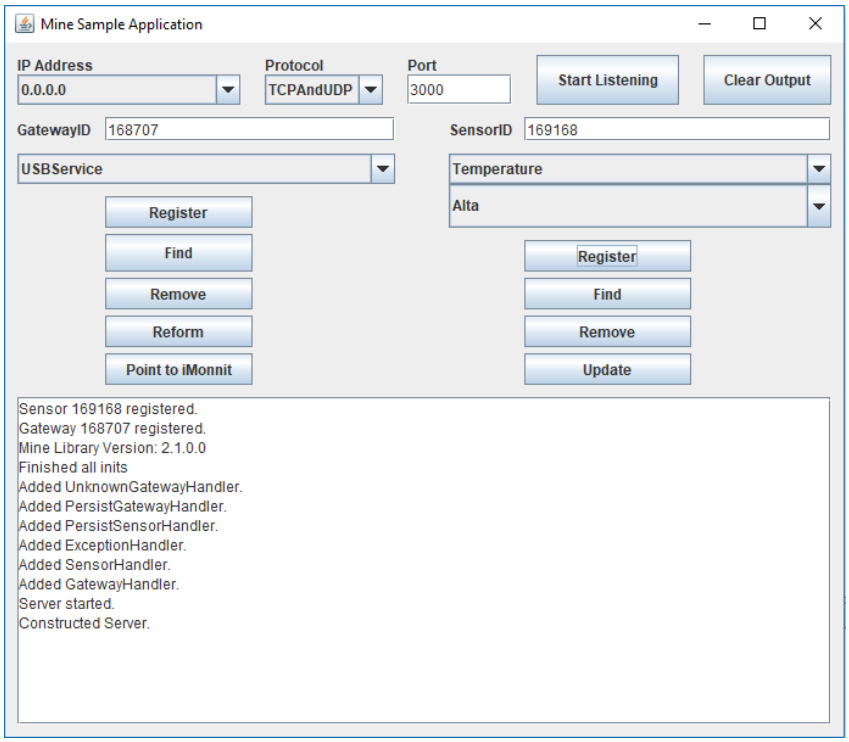


The Find button finds a gateway that is registered to the server. The Remove will delist the gateway from the server and prevent communication. The Reform Button will change the communication channel the gateway talks over, clear the sensor list on the gateway, and download a new sensor list. The Point to iMonnit button will point the gateway back to the iMonnit cloud service.

4 REGISTER A SENSOR

Sensors collect data and send that data to a gateway they are registered to on each heartbeat (Report Interval). The type of data collected is dependent on the type of sensor collecting the data.

To register a sensor it requires you to have a gateway id and sensor id. You will want to put the ids in their respective textboxes. Select the sensor type from the available drop down and click the Register button.



The Find button finds a sensor that has be registered to a gateway. The Remove button removes a sensor from the gateways sensor list. After removing the sensor you will want to reform the specific gateway(s) the sensor was registered to. This way the gateways that had the sensor in its sensor list will no longer collect the specific sensors data. The Update button will update a sensor's heartbeat to one minute. If the button is clicked a second time it will update to 10 second heartbeat. If you click it again it will put it back to one minute.

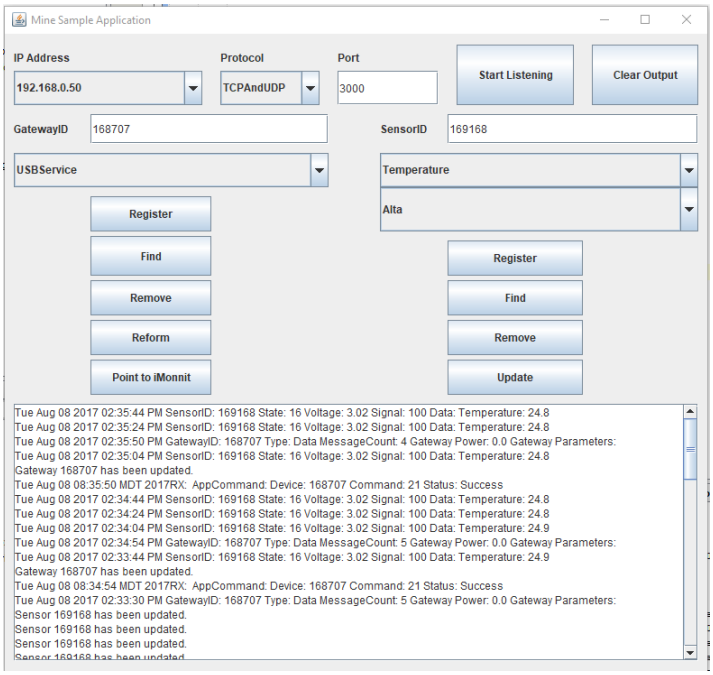
5 POINT GATEWAY AT THE SAMPLE APPLICATION

To point your gateway, you will need to know the gateway id, gateway code, ip address, and the Port the MGSA is using. Will then want to open an internet browser (IE, Firefox, Chrome) and navigate to www.imonnit.com/point. You will put in your gateway id and gateway code into their respective textboxes. If the gateway isn't unlocked yet at this point it will ask for your unlock code you will then need to put it in. The last page will ask for the new servers ip address and port. For more detailed information pointing to a new server you can look up at www.monnit.com/support/.

6 DATA MESSAGES

After the gateway has been pointed to the correct ip address and port. The gateway will now be able to communicate with the server. You will be able to watch the messages come in at the bottom of the SA.

Typical data messages will contain the gateway message and the specific data message from that gateway's sensors.



7 CREATING A SERVER

The basis of the MJL is the ability of creating a server application that will parse your Monnit Hardware data. In the MGSA the class titled GUIListenerFunctions.java there is a function named startButtonPressed(). This function creates a new MineServer object by passing in the protocol to be used (tcp, udp, or both), ip address selected, and the port number. After the new object is instantiated you are able to call the StartServer() function as shown below starting on line number 39 of the GUIListenerFunctions.java class:

```
38      public static void startButtonPressed() throws Exception {
39          int port = getPort();
40          if (port < 1) {
41              gui.println("Invalid Port Number.");
42              return;
43          }
44          eMineListenerProtocol Protocol;
45          try {
46              Protocol = eMineListenerProtocol.valueOf(gui.protocolDropdown.getSelectedItem().toString());
47          } catch (Exception e) {
48              print("Invalid Protocol");
49              return;
50          }
51
52          InetAddress ip = InetAddress.getByName(gui.ipDropdown.getSelectedItem().toString());
53
54          _Server = new MineServer(Protocol, ip, port);
55          print("Constructed Server.");
56          _Server.StartServer();
57          print("Server started.");
```

Your server is now able to be started. Next you will want to add in all your processing handlers.

8 ADDING PROCESSING HANDLERS

There are seven handlers used in the MJL that you will want to "hook up". Here is the list of handlers you will need to create.

1. GatewayMessageHandler – prints gateway messages
2. SensorMessageHandler – prints sensor messages
3. ExceptionHandler – handles possible incoming exceptions
4. PersistSensorHandler – persists the sensor object and lets you know when the sensor has been updated
5. PersistGatewayHandler – persists the gateway object and lets you know when the gateway has been updated
6. UnknownGatewayHandler – how to process unknown gateways
7. ResponseHandler – processing of all message responses

Below we show you how to add the handlers to a server starting on line number 61 of the GUIListenerFunctions.java class:

```
60          if (_Server.addGatewayDataProcessingHandler(new GatewayMessageHandler())) {
61              print("Added GatewayHandler.");
62          } else {
63              print("Failed to add handler.");
64          }
65          if (_Server.addSensorDataProcessingHandler(new SensorMessageHandler())) {
66              print("Added SensorHandler.");
67          } else {
68              print("Failed to add handler.");
69          }
70          if (_Server.addExceptionProcessingHandler(new ExceptionHandler())) {
71              print("Added ExceptionHandler.");
72          } else {
73              print("Failed to add handler.");
74          }
75          if (_Server.addPersistSensorHandler(new PersistSensorHandler())) {
76              print("Added PersistSensorHandler.");
77          } else {
78              print("Failed to add handler.");
79          }
80          if (_Server.addPersistGatewayHandler(new PersistGatewayHandler())) {
81              print("Added PersistGatewayHandler.");
82          } else {
83              print("Failed to add handler.");
84          }
85          if (_Server.addUnknownGatewayHandler(new UnknownGatewayHandler())) {
86              print("Added UnknownGatewayHandler.");
87          } else {
88              print("Failed to add handler.");
89          }
90
91          ResponseHandler responseHandler = new ResponseHandler();
92          _Server.addGatewayResponseHandler(responseHandler);
```

You are now ready to start registering gateways.

9 GATEWAY REGISTRATION

To be able to register your gateway there are a few things we are going to want to know.

1. Gateway ID
2. Gateway Type
3. Gateway Firmware Version
4. Radio Firmware Version
5. Host Address (gateway should be using to talk to the server)
6. Port (gateway should be using to talk to the server)

You can get the Gateway Firmware Version and Radio Firmware Version by visiting www.imonnit.com/lookup and entering in your gateway id and gateway code found on your gateway hardware.

After getting all the information you would create a new gateway object using the information and pass it as a parameter to the MineServer function RegisterGateway(Gateway gateway). As shown below starting at line number 156 of the GUIListenerFunctions.java class:

```
155     public static void RegisterGateway() throws Exception {
156         int gid = getGid();
157         if (gid > 0) {
158             eGatewayType egt = eGatewayType.valueOf(gui.gType.getSelectedItem().toString());
159             int port = getPort();
160             Gateway gate = new Gateway(
161                     gid,
162                     egt,
163                     "3.4.1.3",
164                     "1.0.1.0",
165                     "0.0.0.0",
166                     port
167             );
168             gate.IsDirty = false;
169             _Server.RegisterGateway(gate);
170 
171             // gate.UpdateReportInterval(.5);
172             print("Gateway " + gid + " registered.");
173         }
174     }
```

We are setting the isDirty flag to false, so that the gateway checks into the server for the first time instead of trying to update. It will read in all the configurations from the gateway and update the gateway object automatically.

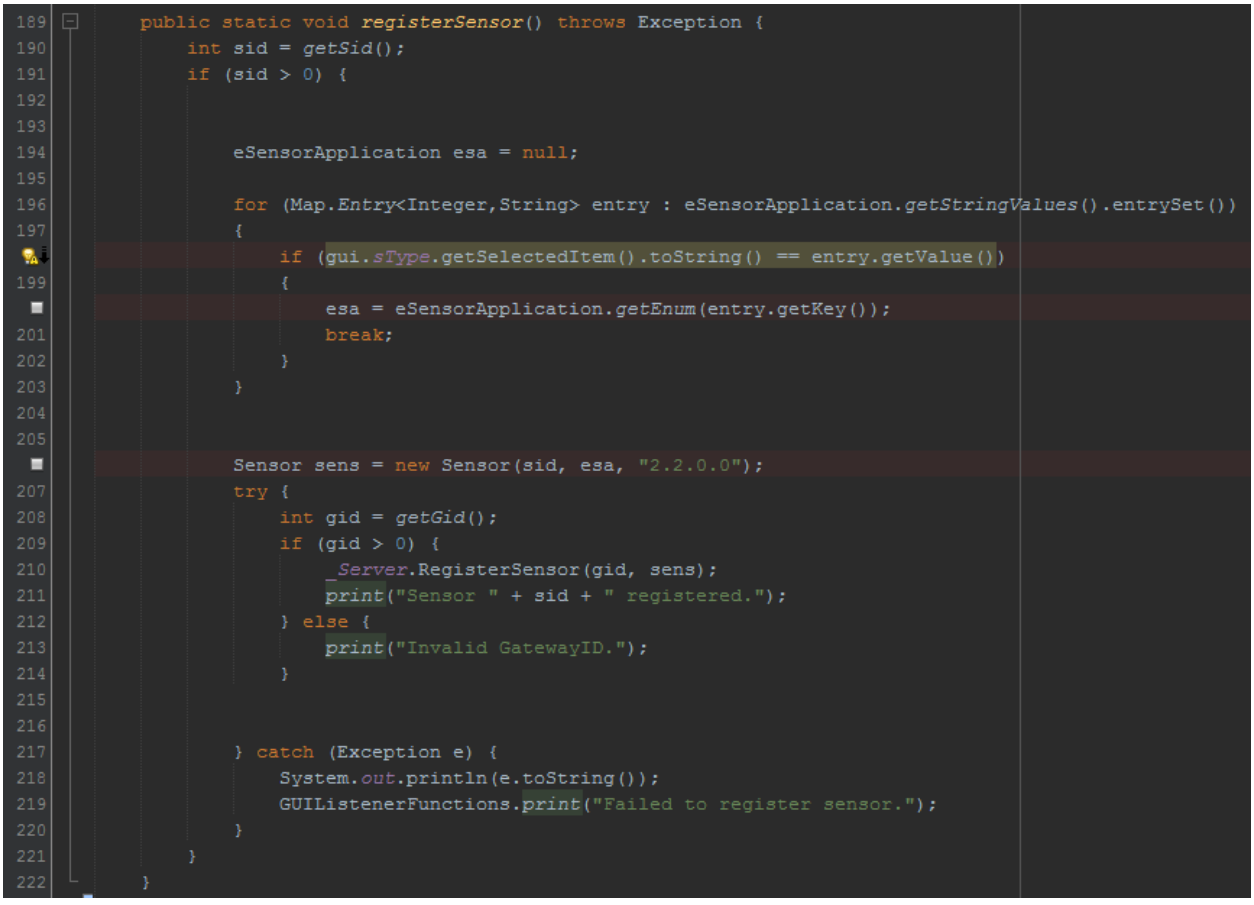
10 SENSOR REGISTRATION

To register a sensor you need four pieces of information.

1. Sensor ID
2. Sensor Application
3. Firmware Version
4. Gateway ID

Sensor application and firmware can be obtained from www.imonnit.com/lookup by entering in your sensor id and sensor code found on the sensor hardware.

After all the information is collected the programmer will create a new sensor object. You will need the id of the gateway you are going to register the sensor to. This will add the sensor into the gateway's sensor list. You will pass the gateway id and new sensor object as parameters to the MineServer function RegisterSensor(int gatewayid,Sensor sensor) as shown below starting at line number 190 of the GUIListenerFunctions.java class:



```
189     public static void registerSensor() throws Exception {
190         int sid = getSid();
191         if (sid > 0) {
192
193
194             eSensorApplication esa = null;
195
196             for (Map.Entry<Integer,String> entry : eSensorApplication.getStringValues().entrySet())
197             {
199                 if (gui.sType.getSelectedItem().toString() == entry.getValue())
                    {
201                     esa = eSensorApplication.getEnum(entry.getKey());
202                     break;
203                 }
204             }
205
207             Sensor sens = new Sensor(sid, esa, "2.2.0.0");
208             try {
209                 int gid = getGid();
210                 if (gid > 0) {
211                     _Server.RegisterSensor(gid, sens);
212                     print("Sensor " + sid + " registered.");
213                 } else {
214                     print("Invalid GatewayID.");
215                 }
216
217
218             } catch (Exception e) {
219                 System.out.println(e.toString());
220                 GUIListenerFunctions.print("Failed to register sensor.");
221             }
222         }
    }
```

When the sensor first joins the network it will send a read configuration. This will pass all of the specific sensor's configurations to the server and will call the ProcessPersistSensor(int SensorID) function from PersistSensorHandle object which will update the sensor. You are now ready to start collecting your sensor data.

11 GATEWAY AND SENSOR MESSAGES

To process Gateway Messages there is a handler that we added earlier in this guide the GatewayMessageHandler class implements the interface iGatewayMessageHandler, so that we can override the ProcessGatewayMessage(GatewayMessage gatewayMessage) function. In the MGSA we simply call a print function and pass gatewayMessage.toString(). As shown below starting at line number 8 of the GatewayMessageHandler.java class:

```
8       @Override
        public void ProcessGatewayMessage(GatewayMessage gatewayMessage)
10              throws Exception {
11          GUIListenerFunctions.print(gatewayMessage.toString());
12      }
```

Similarly, to process sensor messages you will create the SensorMessageHandler class that implements the iSensorMessageHandler interface and override the ProcessSensorMessages(List<SensorMesssage> sensorMessageList,Gateway gateway) function. In theMGSAwe iterate through the sensorMessageList and do a simple print on each individual message using the overridden toString() function from the SensorMessage object. As shown below starting at line number 13 of the SensorMessageHandler.java class:

```
13      @Override
        public void ProcessSensorMessages(List<SensorMessage> sensorMessageList, Gateway gateway) throws Exception {
15          for (SensorMessage msg : sensorMessageList) {
16
17
18
19              if (msg.ProfileID > 65000) {
20                  Sensor sens = GUIListenerFunctions.FindSensorBySensorID(msg.SensorID);
21                  msg.ProfileID = sens.MonnitApplication.Value();
22
23              }
24           GUIListenerFunctions.print(msg.toString());
25          }
26      }
```

You are now ready to start receiving data now and updating configurations of the gateway object and sensor objects.

12 SENSOR AND GATEWAY UPDATE

To update a sensor you will need to know the sensors application type by calling ApplicationBase.GetType(int applicationID) and passing the sensors monnitapplication.value() this will bring back the correct class, so that you can call the correct sensor edit function and

update page you will want to create. Looking at TemperatureBase's SensorEdit function we can see that it contains the following parameters:

1. Sensor sens
2. Boolean isFahrenheit
3. Double heartbeat
4. Double awareStateHeartBeat
5. Integer assessmentsPerHeartBeat
6. Double minimumThreshold
7. Double maximumThreshold
8. Double hysteresis
9. Integer failedTransmissionBeforeLinkMode

In the below example we are just passing a heartbeat and null for the other values that we do not want to change. As Shown below starting at line number 20 of the UpdateSensor.java class:

```
18      public static void updateTempSensor(int sid)
19              throws Exception {
20          try {
21              Sensor sens = GUIListenerFunctions.FindSensorBySensorID(sid);
22              CompassBase.SensorEdit(sens, getHeartbeat(), getHeartbeat(), null,null,null,null,null,null);
23
24          } catch (Exception ex) {
25              ex.printStackTrace();
26          }
27      }
```

This will set only the heartbeat. If the heartbeat is lower than the aware state heartbeat it will also set that as well. You can read more information on every sensor edit function in the Mine Java API documentation found at mine.imonnit.com.

To update the gateway Heartbeat (read in the gateway object as ReportInterval.), Server Host Address, Port, or Poll Interval. You will want to set the corresponding attributes of the gateway object and set the isDirty flag to true. In the example below we set the gateway object, so that it's ServerHostAddress and Port are able to be pointed back to the iMonnit Cloud Service. You can find this starting at line number 301 of the GUIListenerFunctions.java class:

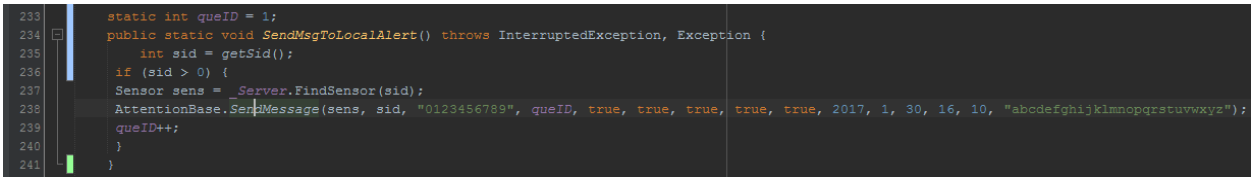
```
300         public static void PointToiMonnit()
301         {
302             Gateway gate = _Server.FindGateway(getGid());
303              if (gate != null) {
304 
305                  gate.ServerHostAddress = "u1.sensorsgateway.com";
306                  gate.Port = 3000;
307                  gate.setIsDirty(true);
308 
309                 GUIListenerFunctions.print("update pending.");
310             } else {
311                 GUIListenerFunctions.print("Gateway does not exist.");
312             }
313         }
```

13 Local Alert Messaging

The Monnit Wireless Local Alert provides an additional way to receive critical notifications and sensor readings. Upon receiving a message the Local Alert can flash an LED, sound an audible alarm, and display critical notification information. The Local Alert can also be used to display sensor readings. To send a message to the Local Alert you will want to pass the following parameters to the SendMessage function in the AttentionBase Class.

1. Sensor sens: the local alert sensor object you want to send the message to.
2. Int deviceID: this is the id of the sensor or gatway that the message is coming from.
3. String deviceName: this is the name of the device that the message is coming from.
4. Int queID: every specific message must have a queId to keep track of.
5. Boolean isNotification: is the message a normal message vs a critical message
6. Boolean led: turns this on or off on the local alert for the specific message.
7. Boolean buzzer: turns this on or off on the local alert for the specific message.
8. Boolean scroll: turns this on or off on the local alert for the specific message.
9. Boolean backlight: turns this on or off on the local alert for the specific message.
10. Int year: the year the message took place.
11. Int month: the month the message took place.
12. Int day: the day the month took place.
13. Int hour: the hour the message took place (uses a 24 hour clock model, so 1 would be 1 am and 13 would be 1 pm).
14. Int minute: the minute the message took place (0-59)
15. String message: the body of the actual message you want to send

You can find this starting at line number 234:



```
233     static int queID = 1;
234     public static void SendMsgToLocalAlert() throws InterruptedException, Exception {
235         int sid = getSid();
236      if (sid > 0) {
237      Sensor sens = _Server.FindSensor(sid);
238      AttentionBase.SendMessage(sens, sid, "0123456789", queID, true, true, true, true, true, 2017, 1, 30, 16, 10, "abcdefghijklmnopqrstuvwxyz");
239      queID++;
240      }
241     }
```

14 REVIEW

In this document we have shown how to use the MGSA. How to create a server, register handlers, register gateways, register sensors, print gateway messages, print sensor messages, update sensors, and update gateways. To get more advanced with the MJL you will want to look over the Java Mine API documentation at mine.imonnit.com.

For questions or support please contact your Monnit Sales Rep using our toll free number 877-561-4555.