PTC®

PTC® IoT Academic Program

How to send data to the ThingWorx Server from Java Edge SDK programs running on your microcontroller

This tutorial guides you through the procedure of sending data to a ThingWorx server using ThingWorx Java Edge SDK, based on the SteamSensor example that you can find in the archive containing the SDK. To learn more about ThingWorx Edge SDKs follow the tutorial How to deploy programs on your microcontroller on Learning Exchange site.

If you don't have Java or a Java IDE already installed follow this tutorial to learn how to do this.

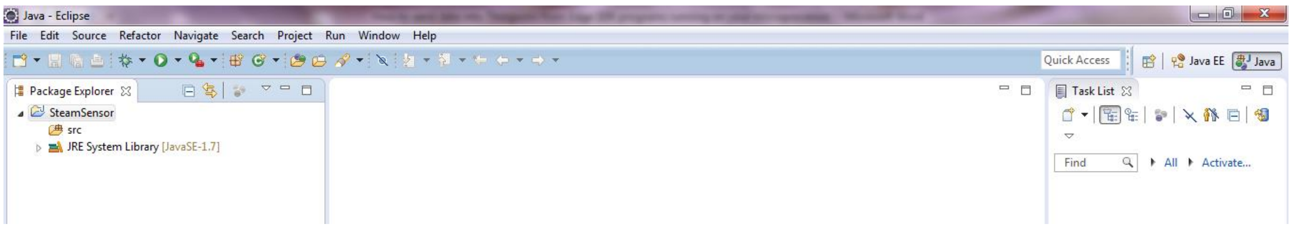
The SteamSensor example can be found in the archive containing the Thingworx Java Edge SDK which you can download from Thingworx Market Place.

The two classes that build this example are SteamThing and SteamSensorClient. The SteamThing class models the actual SteamSensor with its properties, events and services, representing basically the skeleton of the program. The SteamSensorClient class works as the brain of the program, the Main function if you'd like, in which the connection to a ThingWorx Server instance is established. This class also takes care of the binding[1] process of the Things to the ThingWorx server instance you choose to connect to and of the simulated process of reading the properties for several SteamSensorThings.

To set up this project in Eclipse and start working on it follow the next steps :

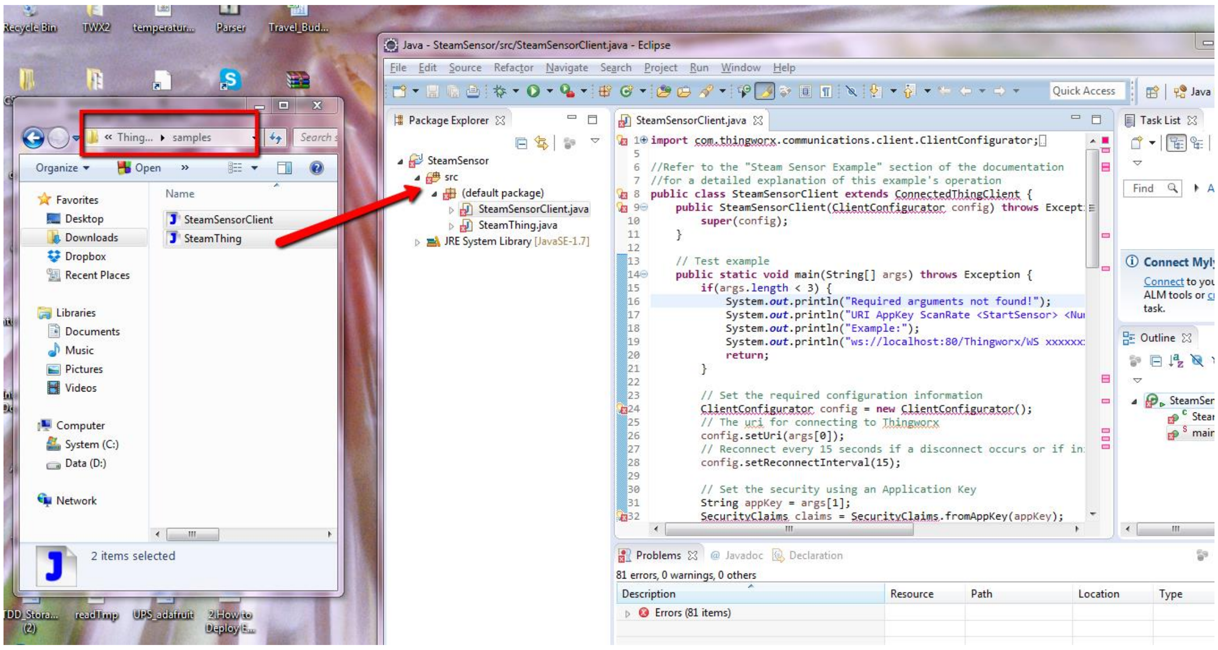
- First download the ThingWorx Java Edge SDK and extract the files to a destination of your choice.
- Then open up your Java IDE. I will use Eclipse in this tutorial. Go to File → New → Java Project. Name the Project SteamSensor.

You will be presented with the following structure:


Java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
Java EE
Java
Package Explorer
SteamSensor
src
JRE System Library [JavaSE-1.7]
Task List
Find
All
Activate...

- Drag and drop the two .java files from the samples folder in the src folder of your IDE.

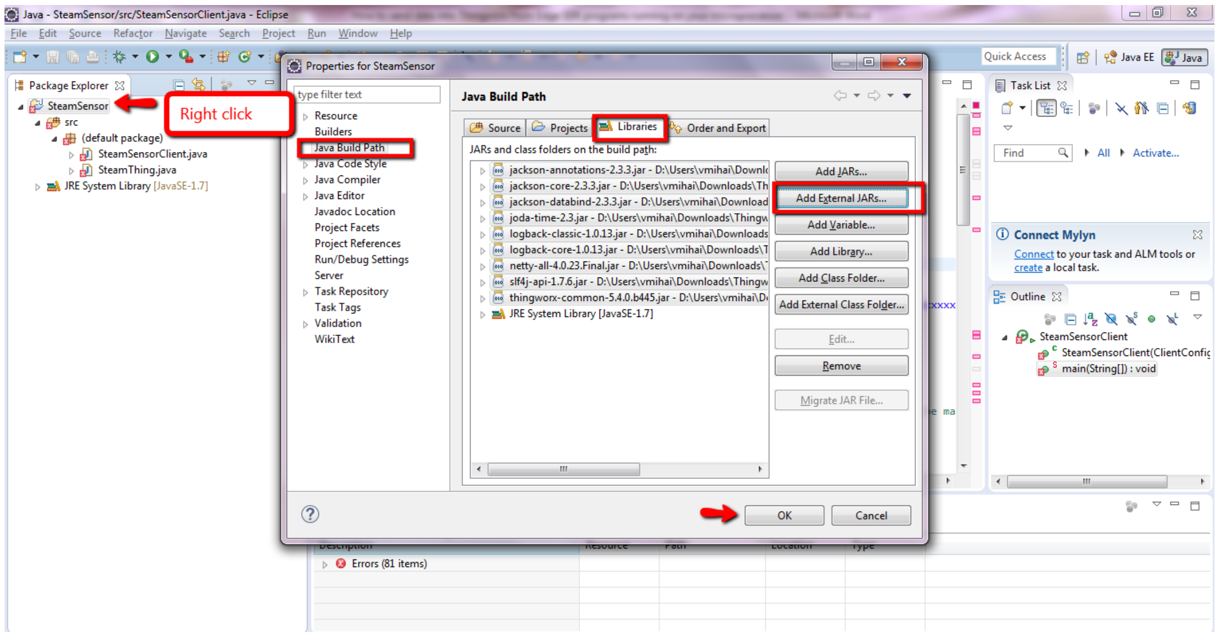
[1] Through this binding, the Thingworx Web application will receive requests that a Remote Thing wants to connect and send data to it.



```
import com.thingworx.communications.client.ClientConfigurator;

//Refer to the "Steam Sensor Example" section of the documentation
//for a detailed explanation of this example's operation
public class SteamSensorClient extends ConnectedThingClient {
    public SteamSensorClient(ClientConfigurator config) throws Except
        super(config);
    }

    // Test example
    public static void main(String[] args) throws Exception {
        if(args.length < 3) {
            System.out.println("Required arguments not found!");
            System.out.println("URI AppKey ScanRate <StartSensor> <Nu
            System.out.println("Example:");
            System.out.println("ws://localhost:80/Thingworx/WS xxxxxx
            return;
        }

        // Set the required configuration information
        ClientConfigurator config = new ClientConfigurator();
        // The uri for connecting to Thingworx
        config.setUri(args[0]);
        // Reconnect every 15 seconds if a disconnect occurs or if in
        config.setReconnectInterval(15);

        // Set the security using an Application Key
        String appKey = args[1];
        SecurityClaims claims = SecurityClaims.fromAppKey(appKey);
```

```
Problems
81 errors, 0 warnings, 0 others
Description          Resource   Path   Location   Type
▷ Errors (81 items)
```

- You will get several errors because the methods used in these classes won't be found.
- To resolve this, add the jar files that you can find in the lib folder of the Edge Java SDK archive. To do this in the Eclipse environment follow the next steps :
  - Right click the SteamSensor project and choose Properties.
  - Go to Java Build Path and select the Libraries Tab.
  - Here click add External JARs and locate the lib folder of the Java Edge SDK that you downloaded.
  - Select all of them and click open.



```
Java Build Path
JARs and class folders on the build path:
jackson-annotations-2.3.3.jar - D:\Users\vmihai\Downl
jackson-core-2.3.3.jar - D:\Users\vmihai\Downloads\Th
jackson-databind-2.3.3.jar - D:\Users\vmihai\Download
joda-time-2.3.jar - D:\Users\vmihai\Downloads\Thingw
logback-classic-1.0.13.jar - D:\Users\vmihai\Downloads
logback-core-1.0.13.jar - D:\Users\vmihai\Downloads\T
netty-all-4.0.23.Final.jar - D:\Users\vmihai\Downloads\
slf4j-api-1.7.6.jar - D:\Users\vmihai\Downloads\Thingw
thingworx-common-5.4.0.b445.jar - D:\Users\vmihai\D
JRE System Library [JavaSE-1.7]
```

  - Then click OK. You shouldn't see any more errors in your two Java classes.

To better understandi–this Java Program it helps to familiarize yourself with terms such as Thing, Thing Properties, Thing Events, Thing Services under Thingworx. To find more about this, check out this Glossary of Terms. You can experience more with these terms by exploring the other academic resources on this page.

The SteamSensorThing class in this java Edge example models and simulates an actual physical Steam sensor with different properties:

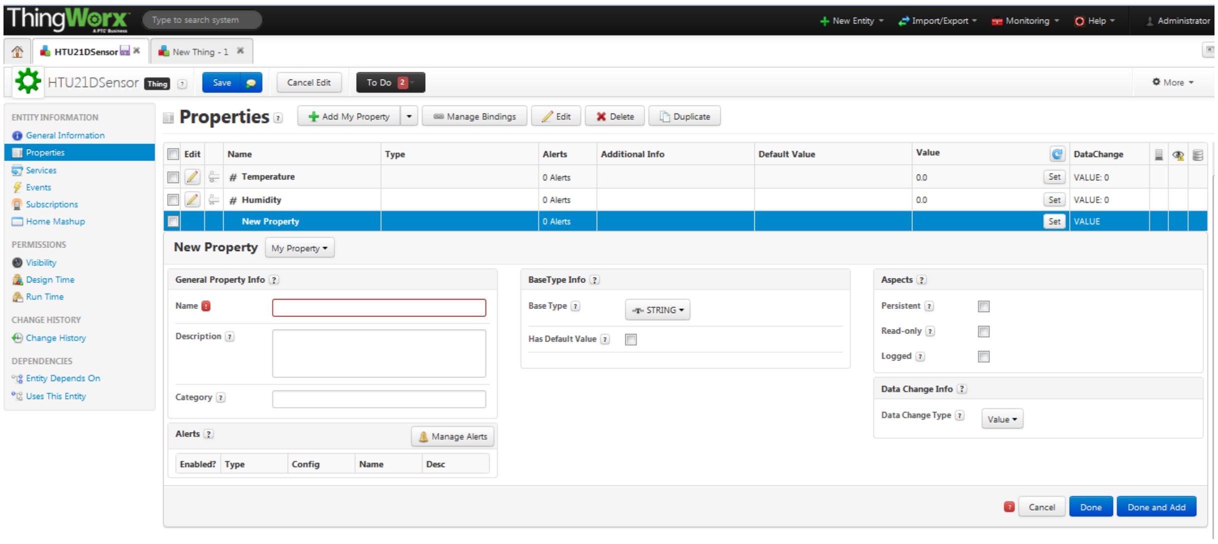
Name	Description	Function
Temperature	Steam Sensor Temperature	Ranges from 400-440
Pressure	Steam Sensor Pressure	Ranges from 18-23
TotalFlow	Total flow through the Steam Sensor	Starts at 0 and increments by 0-1 every scan rate
InletValve	Steam Sensor inlet valve	Usually true, set to false on 15 second interval
TemperatureLimit	Defines the value that causes a Steam Sensor Fault	If greater than 0 and less than temperature then cause a steam sensor fault
FaultStatus	Is the Steam Sensor faulted	True when in fault (see what causes a fault above) false otherwise

In the actual java code these properties are defined using a Java technique called Java Annotations.

Here is how a read-only property of type number used to store the Temperature of the SteamSensor is defined using Java Annotations :

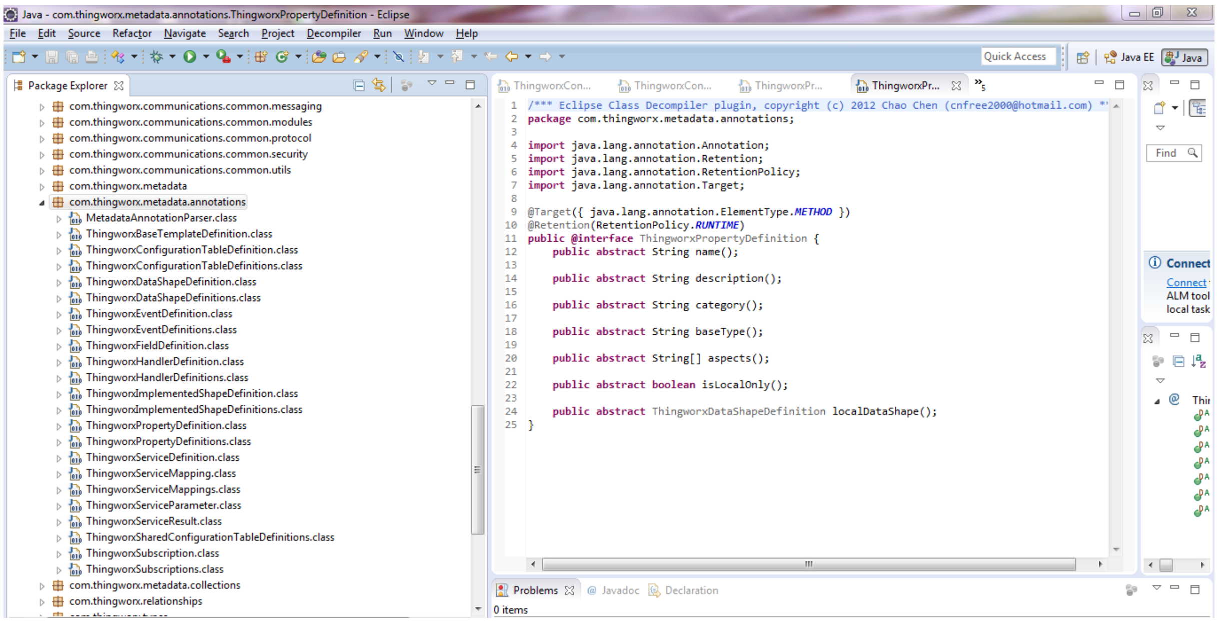
```
@ThingworxPropertyDefinition(name="Temperature", description="Current
Temperature", baseType="NUMBER", category="Status", aspects={"isReadOnly:true"})
```

The keys (name, description, baseType etc.) and values (example „Number" for baseType, isReadOnly for aspects etc.) for the attributes in these properties definitions usually reflect the Thing properties and their values that appear on the ThingWorx platform, when adding a new Property for a Thing :



Edit	Name	Type	Alerts	Additional Info	Default Value	Value	DataChange
	# Temperature		0 Alerts			0.0	VALUE: 0
	# Humidity		0 Alerts			0.0	VALUE: 0
	New Property		0 Alerts				VALUE

The safest way to find out what annotations you can use when defining either a Thing properties, Thing events or Thing services is to check out the package com.thingworx.metadata.annotations in the thingworx-common-5.4.0.b445.jar file that you referenced in your Java Build Path.



```
/*** Eclipse Class Decompiler plugin, copyright (c) 2012 Chao Chen (cnfree2000@hotmail.com) *
package com.thingworx.metadata.annotations;

import java.lang.annotation.Annotation;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

@Target({ java.lang.annotation.ElementType.METHOD })
@Retention(RetentionPolicy.RUNTIME)
public @interface ThingworxPropertyDefinition {
    public abstract String name();

    public abstract String description();

    public abstract String category();

    public abstract String baseType();

    public abstract String[] aspects();

    public abstract boolean isLocalOnly();

    public abstract ThingworxDataShapeDefinition localDataShape();
}
```

The scanDevice() method is where the actual setting of the values for these properties is done. The values set are randomly generated, usually these values come from actual sensors.

```
double temperature = 400 + 40 * Math.random();
super.setProperty("Temperature", temperature);
```

Thingworx doesn't provide any methods for actually reading data from sensors. Nevertheless you can see some examples of how data is read from different sensors using third party libraries on the academic resources page.

Notice that the scanDevice() method is called within the processScanRequest() method and it is this last method that you need to call when you want to update your Things properties values.

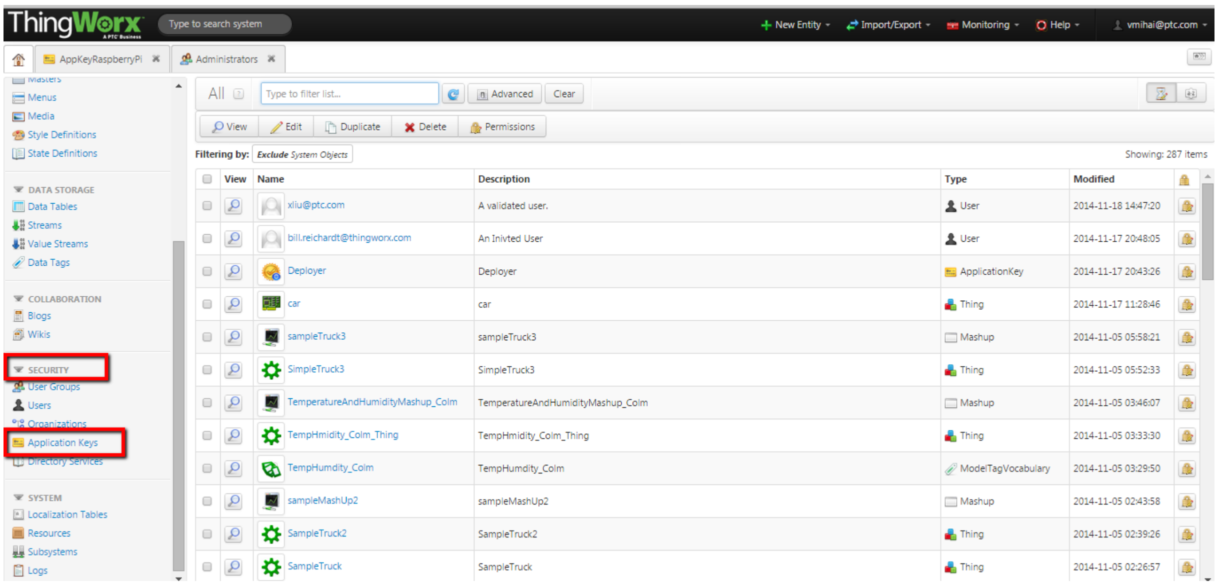
As mentioned earlier, the connection to the Thingworx server is done in the StreamSensorClient class, more precisely in these lines of code :

```
	// The uri for connecting to Thingworx
		config.setUri(args[0]);
		// Reconnect every 15 seconds if a disconnect occurs or if initial
connection cannot be made
		config.setReconnectInterval(15);
```

The variable args[0] represents an input argument that you need to provide in the Run Configuration for this program. Other input arguments that you need to provide are an AppKey, the ScanRate (the refresh interval at which you want data to be sent to the Thingworx Server) , StartSensor (the sensor at which to start reading data and sending it into TWX), Number Of Sensors (how many SteamSensorThings you would like to connect to Thingworx)

The App key is a ThingWorx created object that is used in the authentication into the ThingWorx Server process. Application keys are used whenever one would like to interact with a ThingWorx Application outside of the actual ThingWorx Web Application. Thus an app key replaces the login credentials input that is normally required for each communication with ThingWorx.

To get this value go to your ThingWorx Composer home tab and locate Application Key in the Security section.

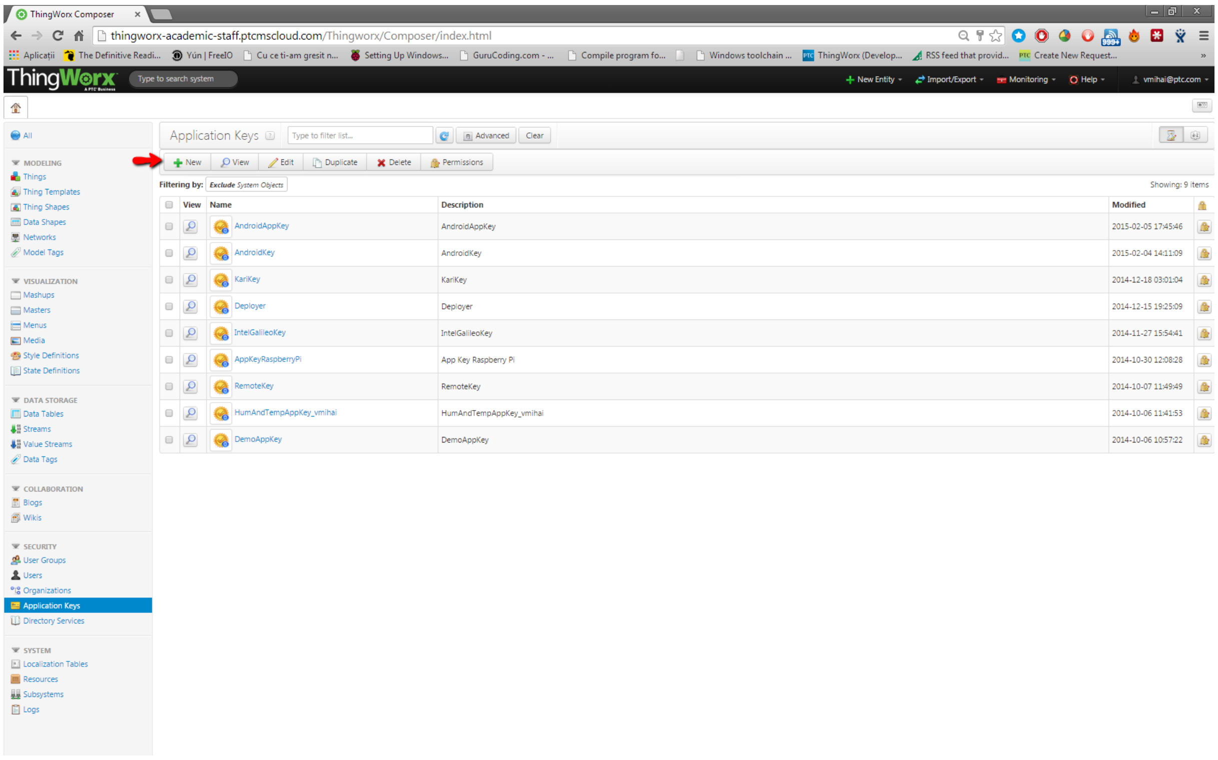


Filtering by: Exclude System Objects

Showing: 287 items

View	Name	Description	Type	Modified
	xliu@ptc.com	A validated user.	User	2014-11-18 14:47:20
	bill.reichardt@thingworx.com	An Inivted User	User	2014-11-17 20:48:05
	Deployer	Deployer	ApplicationKey	2014-11-17 20:43:26
	car	car	Thing	2014-11-17 11:28:46
	sampleTruck3	sampleTruck3	Mashup	2014-11-05 05:58:21
	SimpleTruck3	SimpleTruck3	Thing	2014-11-05 05:52:33
	TemperatureAndHumidityMashup_Colm	TemperatureAndHumidityMashup_Colm	Mashup	2014-11-05 03:46:07
	TempHmidity_Colm_Thing	TempHmidity_Colm_Thing	Thing	2014-11-05 03:33:30
	TempHumdity_Colm	TempHumdity_Colm	ModelTagVocabulary	2014-11-05 03:29:50
	sampleMashUp2	sampleMashUp2	Mashup	2014-11-05 02:43:58
	SampleTruck2	SampleTruck2	Thing	2014-11-05 02:39:26
	SampleTruck	SampleTruck	Thing	2014-11-05 02:26:57

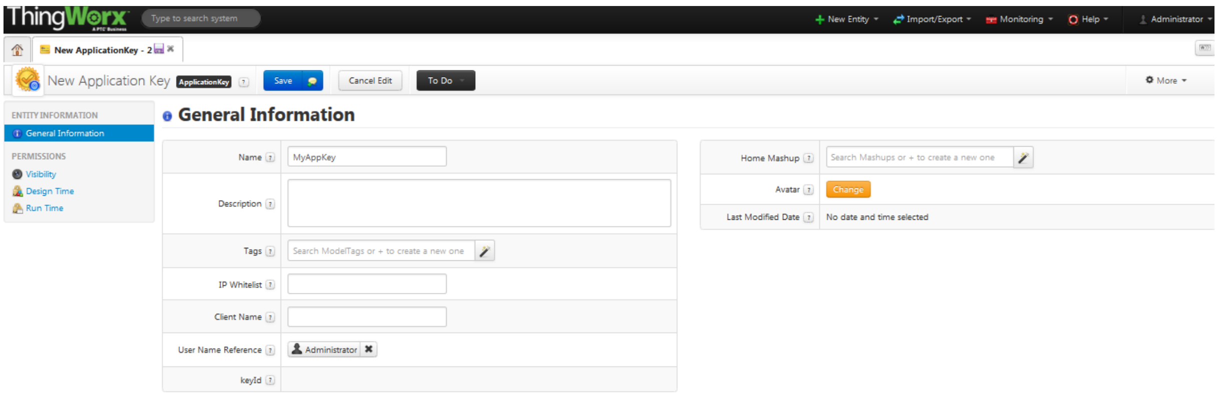
Click it and click the +New Button:



Filtering by: *Exclude* System Objects — Showing: 9 items

View	Name	Description	Modified
	AndroidAppKey	AndroidAppKey	2015-02-05 17:45:46
	AndroidKey	AndroidKey	2015-02-04 14:11:09
	KariKey	KariKey	2014-12-18 03:01:04
	Deployer	Deployer	2014-12-15 19:25:09
	IntelGalileoKey	IntelGalileoKey	2014-11-27 15:54:41
	AppKeyRaspberryPi	App Key Raspberry Pi	2014-10-30 12:08:28
	RemoteKey	RemoteKey	2014-10-07 11:49:49
	HumAndTempAppKey_vmihai	HumAndTempAppKey_vmihai	2014-10-06 11:41:53
	DemoAppKey	DemoAppKey	2014-10-06 10:57:22

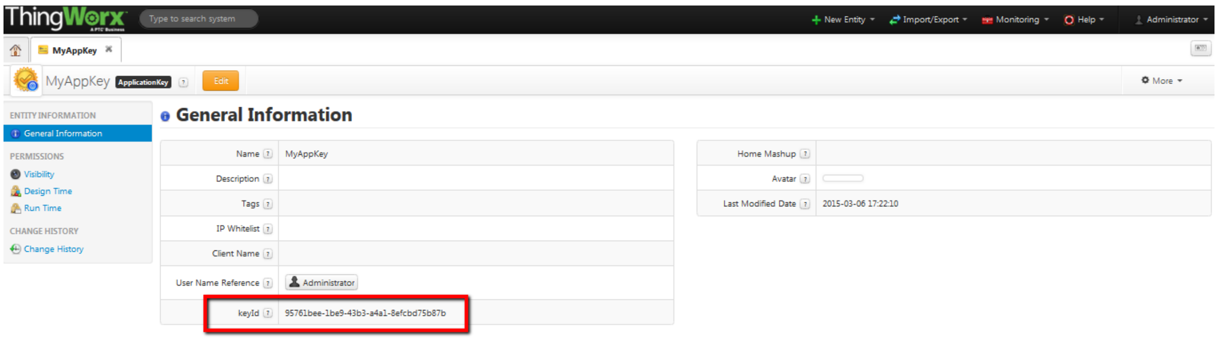
Enter a name, for example: myAppKey and select your User in the User Name Reference field :



Name	MyAppKey
Description	
Tags	Search ModelTags or + to create a new one
IP Whitelist	
Client Name	
User Name Reference	Administrator
keyId	

Home Mashup	Search Mashups or + to create a new one
Avatar	Change
Last Modified Date	No date and time selected

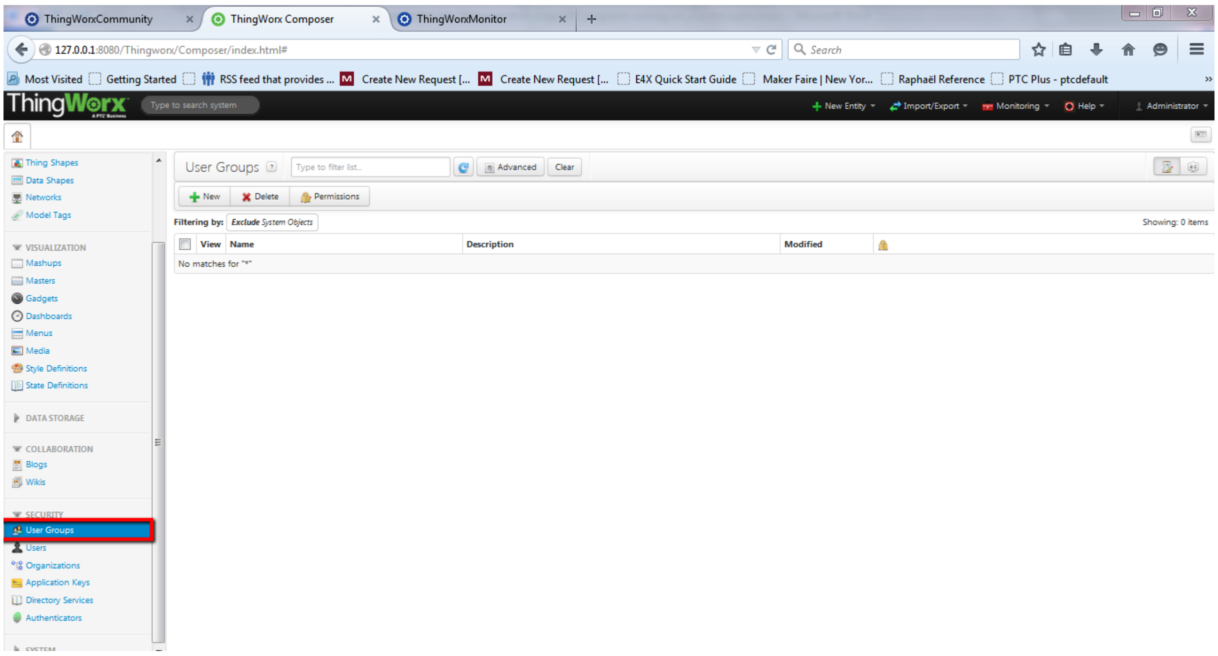
Save the App key and copy the keyId.



Name	MyAppKey
Description	
Tags	
IP Whitelist	
Client Name	
User Name Reference	Administrator
keyId	95761bee-1be9-43b3-a4a1-8efcbd75b87b

Home Mashup	
Avatar	
Last Modified Date	2015-03-06 17:22:10

To be able to visualize data sent remotely from devices, you will need at least developer rights for the App Key. Make sure the user you choose for the App key is the user you are logged in on the Thingworx platform and that it has at least developer rights. To verify if your user belongs to the administrators or the developers group, go to your Thingworx Composer, User Group section.

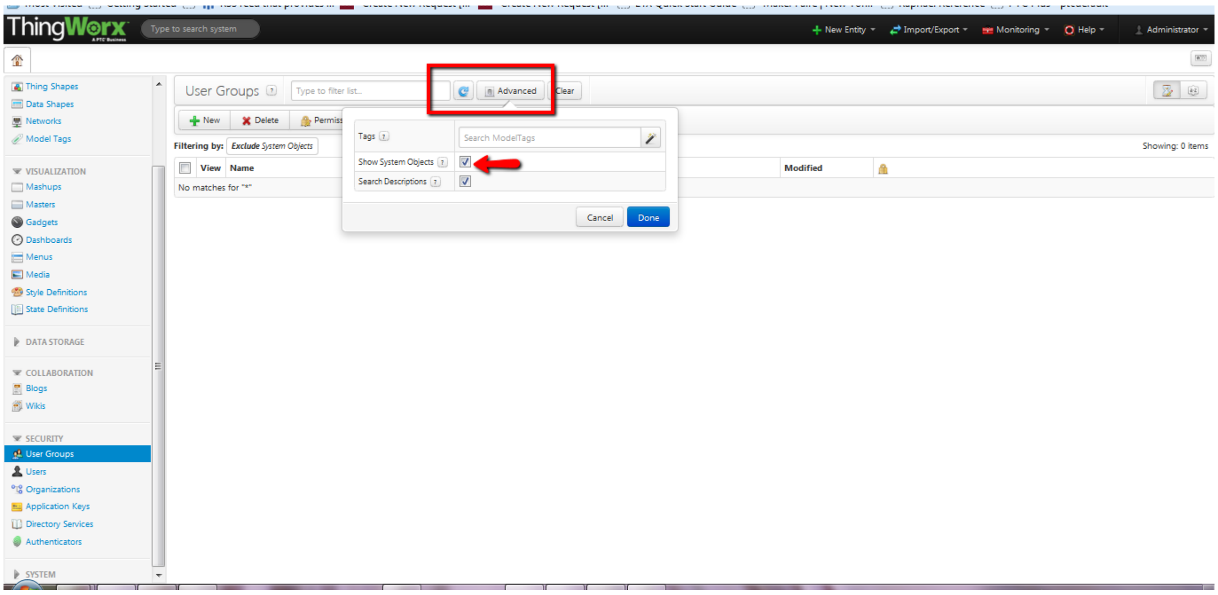


Filtering by: Exclude System Objects

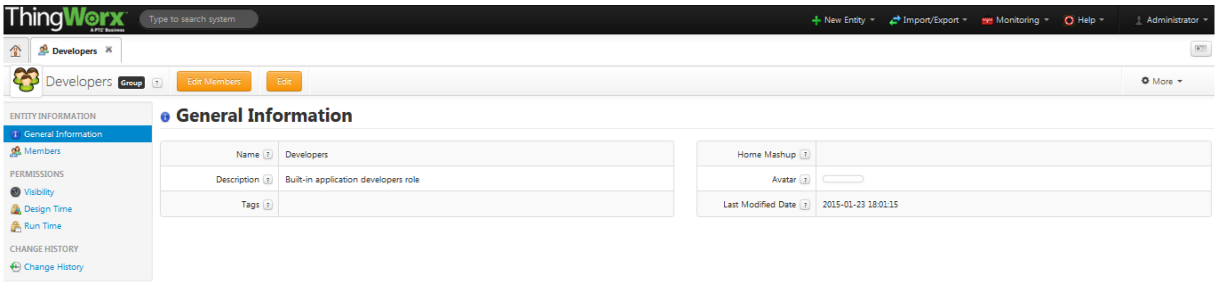
Showing: 0 items

View	Name	Description	Modified
No matches for "*"			

Here click on Advanced and check the Show System Objects Checkbox. Next click Done.



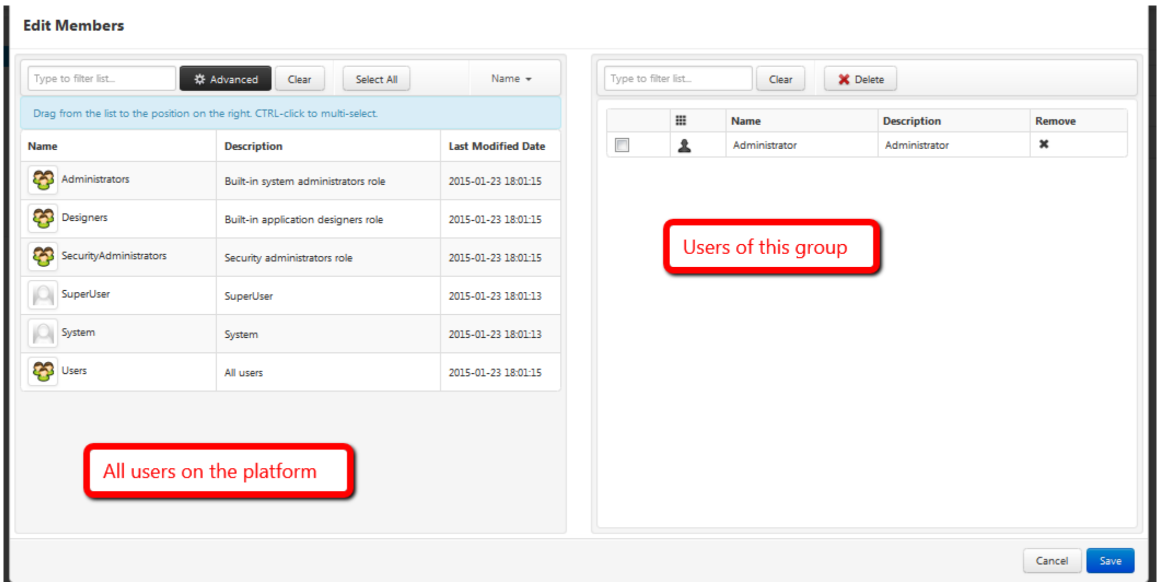
This will show you all the existing User Groups on the Platform. To verify if your user belongs to any of the groups, click on the one you want to verify and go to Edit Members.



General Information

Name	Developers	Home Mashup	
Description	Built-in application developers role	Avatar	
Tags		Last Modified Date	2015-01-23 18:01:15

Here you will be able to see if your user is among the members or not. You could also add it to a certain group if you have administrator or developer rights given by the one who created your account on the platform.



Drag from the list to the position on the right. CTRL-click to multi-select.

Name	Description	Last Modified Date
Administrators	Built-in system administrators role	2015-01-23 18:01:15
Designers	Built-in application designers role	2015-01-23 18:01:15
SecurityAdministrators	Security administrators role	2015-01-23 18:01:15
SuperUser	SuperUser	2015-01-23 18:01:13
System	System	2015-01-23 18:01:13
Users	All users	2015-01-23 18:01:15

		Name	Description	Remove
☐	👤	Administrator	Administrator	✖

Coming back to the java code of the SteamSensor Edge Application, you can see in this for loop how the number of SteamSensorThings that you stated in the input argument are first created and than bound to the client representing the Thingworx Server Instance.

```
for(int sensor=1;sensor <= nSensors; sensor++) {
   int sensorID = startSensor + sensor;
   SteamThing steamSensorThing = new SteamThing("SteamSensor" +   sensorID,"Steam
Sensor #" + sensorID,"SN000" + sensorID,client);
   client.bindThing(steamSensorThing);
}
```

Another for loop is afterwards used to simulate the reading of data from these created and bound to the Thingworx Server SteamSensorThings :

```
for(VirtualThing thing : client.getThings().values()) {
      try {
         thing.processScanRequest();
      }
     catch(Exception eProcessing) {
```

```
System.out.println("Error Processing Scan Request for [" + thing.getName() + "] : 
" + eProcessing.getMessage());
                                    }
```

The interval at which the reading is done being established by this line of code :

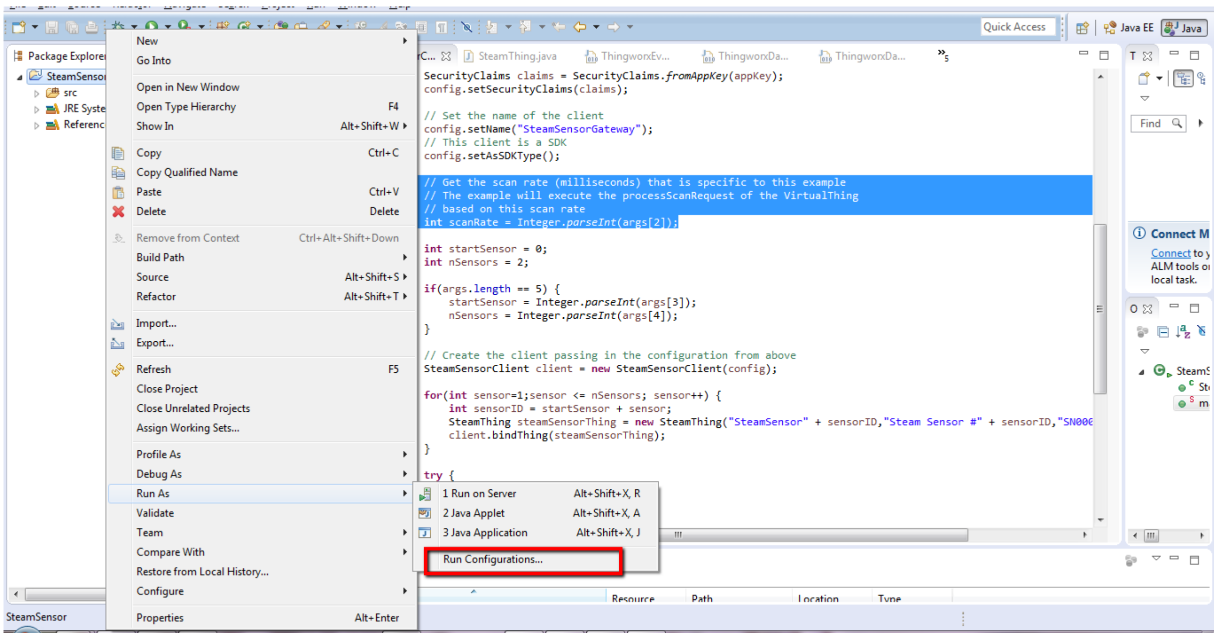
```
// Suspend processing at the scan rate interval
         Thread.sleep(scanRate);
```

Therefore, if you'd like to change this value you would have to change the value for the variable scanRate, which is given as the third input argument in your Run Configuration.

```
// Get the scan rate (milliseconds) that is specific to this example
// The example will execute the processScanRequest of the VirtualThing
// based on this scan rate
        int scanRate = Integer.parseInt(args[2]);
```

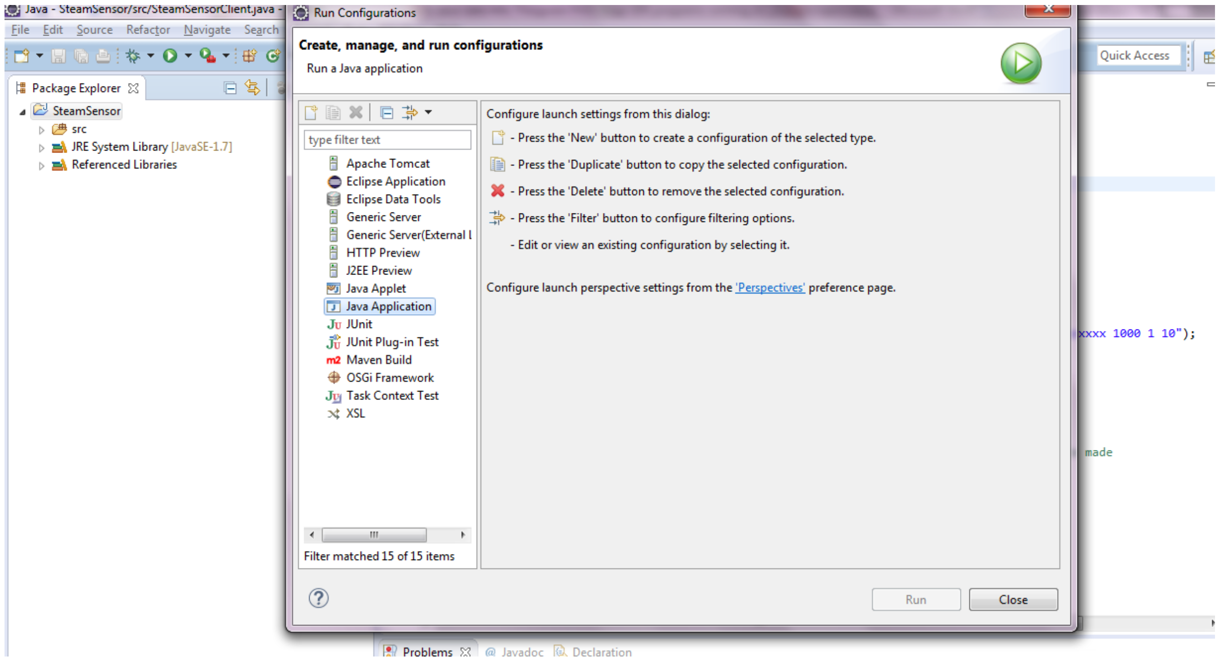
To set up your Run Configuration on Eclipse follow the next steps :

- Right click your SteamSensor java project folder and select Run as → Run Configurations



```
SecurityClaims claims = SecurityClaims.fromAppKey(appKey);
config.setSecurityClaims(claims);

// Set the name of the client
config.setName("SteamSensorGateway");
// This client is a SDK
config.setAsSDKType();

// Get the scan rate (milliseconds) that is specific to this example
// The example will execute the processScanRequest of the VirtualThing
// based on this scan rate
int scanRate = Integer.parseInt(args[2]);

int startSensor = 0;
int nSensors = 2;

if(args.length == 5) {
    startSensor = Integer.parseInt(args[3]);
    nSensors = Integer.parseInt(args[4]);
}

// Create the client passing in the configuration from above
SteamSensorClient client = new SteamSensorClient(config);

for(int sensor=1;sensor <= nSensors; sensor++) {
    int sensorID = startSensor + sensor;
    SteamThing steamSensorThing = new SteamThing("SteamSensor" + sensorID,"Steam Sensor #" + sensorID,"SN000
    client.bindThing(steamSensorThing);
}

try {
```

- Next go to Java Application, right click it and select New.


Create, manage, and run configurations

Run a Java application

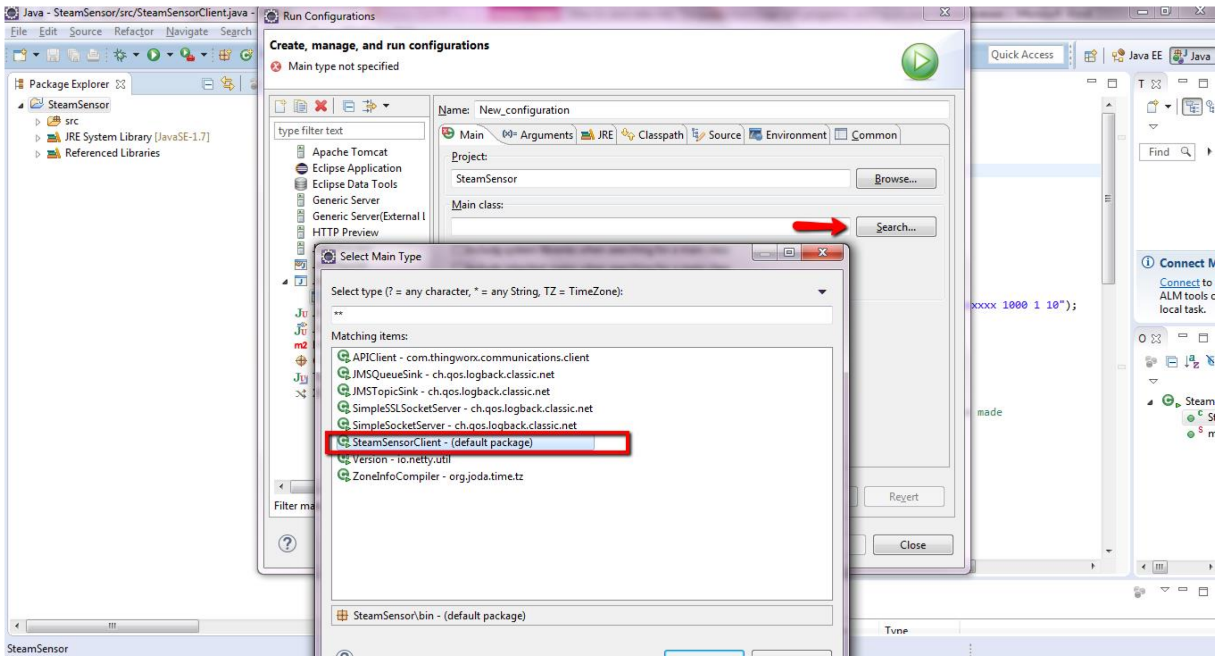
Configure launch settings from this dialog:

- Press the 'New' button to create a configuration of the selected type.
- Press the 'Duplicate' button to copy the selected configuration.
- Press the 'Delete' button to remove the selected configuration.
- Press the 'Filter' button to configure filtering options.
- Edit or view an existing configuration by selecting it.

Configure launch perspective settings from the 'Perspectives' preference page.

Filter matched 15 of 15 items

- Click the Search Button next to the Main Class field, select SteamSensorClient and click OK.


Create, manage, and run configurations

Main type not specified

Name: New_configuration

Project: SteamSensor

Main class:

Select Main Type

Select type (? = any character, * = any String, TZ = TimeZone):

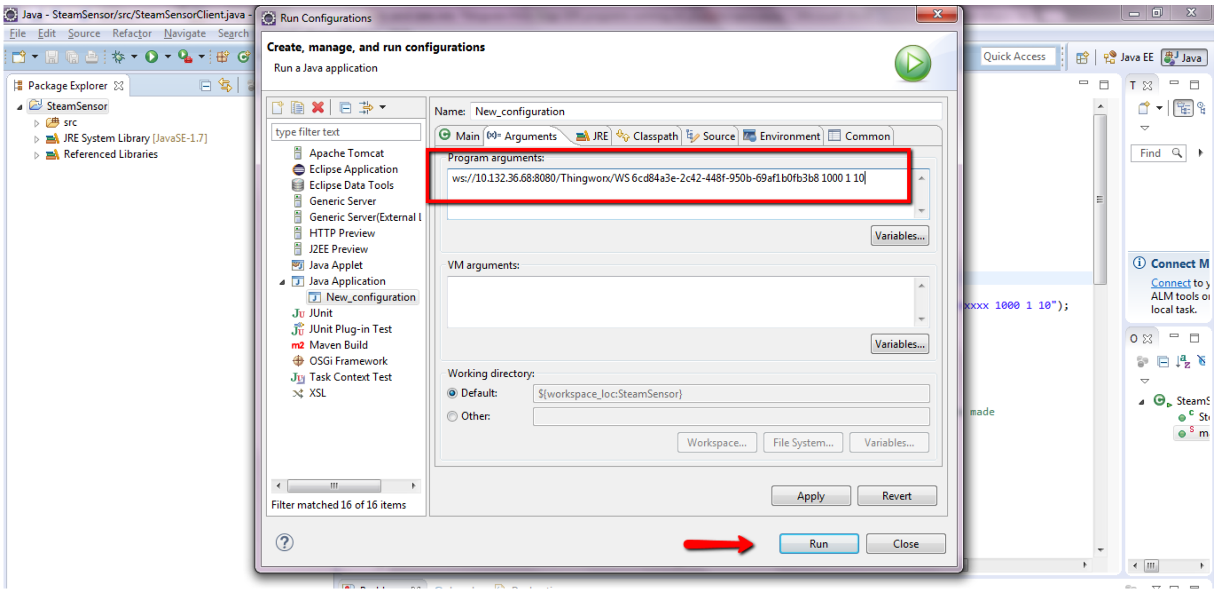
**

Matching items:

APIClient - com.thingworx.communications.client
JMSQueueSink - ch.qos.logback.classic.net
JMSTopicSink - ch.qos.logback.classic.net
SimpleSSLSocketServer - ch.qos.logback.classic.net
SimpleSocketServer - ch.qos.logback.classic.net
SteamSensorClient - (default package)
Version - io.netty.util
ZoneInfoCompiler - org.joda.time.tz

SteamSensor\bin - (default package)

- Then select the Arguments Tab and fill in a String of the Type :
  ws://10.132.36.68:8080/Thingworx/WS baebbf44-8afc-4dcd-a76c-4f340a265437 1000 1 10
- The Arguments follow this pattern :
  URI AppKey ScanRate <StartSensor> <Number Of Sensors>



Run Configurations

Create, manage, and run configurations

Run a Java application

Name: New_configuration

Program arguments:

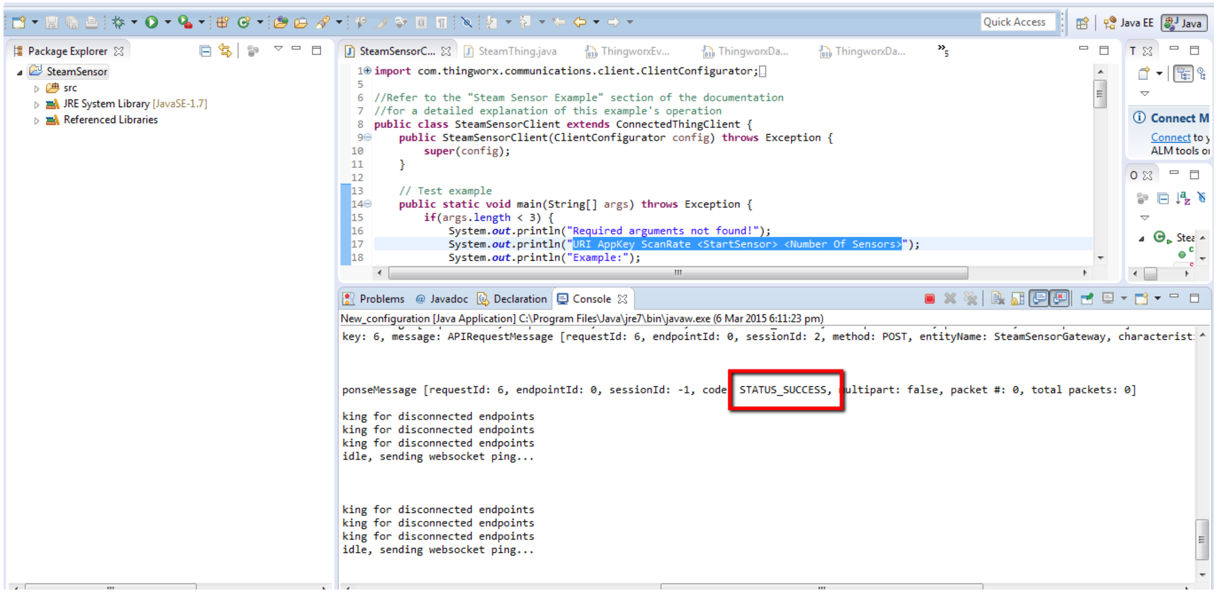
```
ws://10.132.36.68:8080/Thingworx/WS 6cd84a3e-2c42-448f-950b-69af1b0fb3b8 1000 1 10
```

Working directory: Default: ${workspace_loc:SteamSensor}

Filter matched 16 of 16 items

- Next you can click the Run Button.
- The IP address is the IP address of your Thingworx Instance, together with the port on which it is listening and receiving data. Notice that the communication protocol used is Web Sockets, and thus the port is a non-secure one, the port 80. If you are using Secure Web Sockets, you will need to use a different port, most probably 443.
- If you prefer using the name of the server (for example : thingworx-academic-staff.ptcmscloud.com), don't state the port anymore. So the URI of your Thingworx Server would be *ws://thingworx-academic-staff.ptcmscloud.com/Thingworx/WS.*

- When running the application you will see several Log messages in your Console window telling you if the connection was succesful or not.

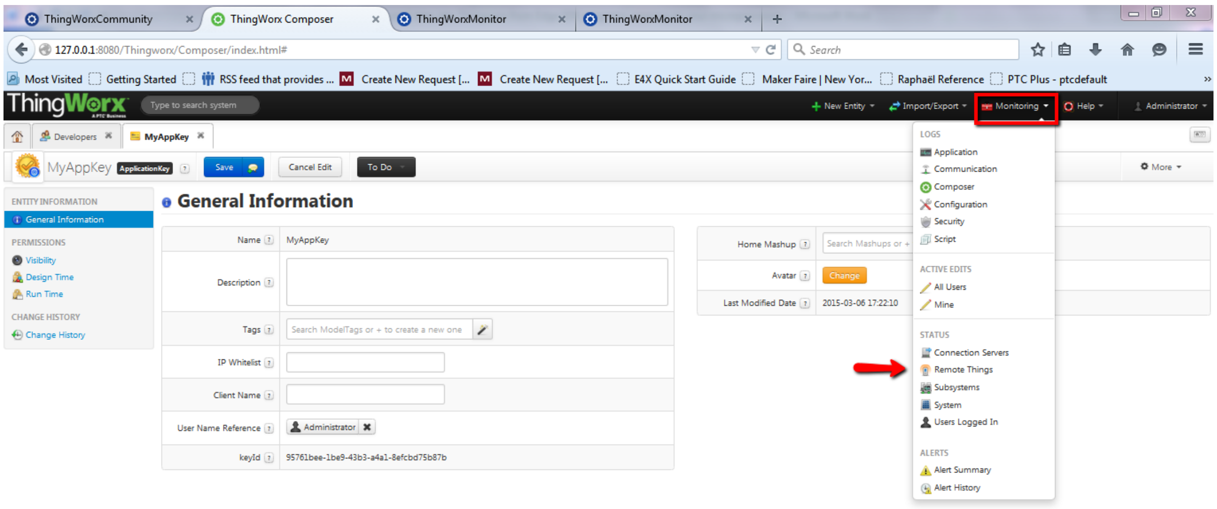


```
import com.thingworx.communications.client.ClientConfigurator;

//Refer to the "Steam Sensor Example" section of the documentation
//for a detailed explanation of this example's operation
public class SteamSensorClient extends ConnectedThingClient {
    public SteamSensorClient(ClientConfigurator config) throws Exception {
        super(config);
    }

    // Test example
    public static void main(String[] args) throws Exception {
        if(args.length < 3) {
            System.out.println("Required arguments not found!");
            System.out.println("URI AppKey ScanRate <StartSensor> <Number Of Sensors>");
            System.out.println("Example:");
```

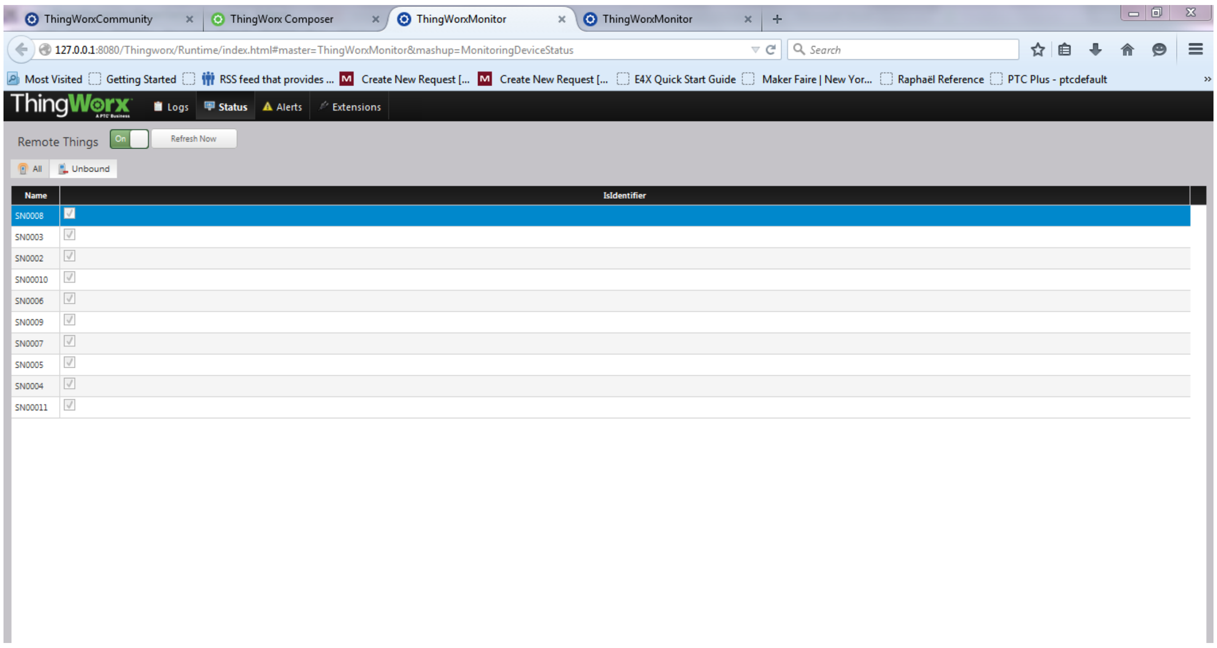
New_configuration [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (6 Mar 2015 6:11:23 pm)

```
key: 6, message: APIRequestMessage [requestId: 6, endpointId: 0, sessionId: 2, method: POST, entityName: SteamSensorGateway, characterist


ponseMessage [requestId: 6, endpointId: 0, sessionId: -1, code: STATUS_SUCCESS, multipart: false, packet #: 0, total packets: 0]

king for disconnected endpoints
king for disconnected endpoints
king for disconnected endpoints
idle, sending websocket ping...



king for disconnected endpoints
king for disconnected endpoints
king for disconnected endpoints
idle, sending websocket ping...
```

- To see your Remote Things go to your Thingworx Composer and Select Monitoring → Remote Things → Unbound.



Name	MyAppKey
Description	
Tags	
IP Whitelist	
Client Name	
User Name Reference	Administrator
keyId	95761bee-1be9-43b3-a4a1-8efcbd75b87b
Home Mashup	
Avatar	
Last Modified Date	2015-03-06 17:22:10



Name	IsIdentifier
SN0008	✓
SN0003	✓
SN0002	✓
SN00010	✓
SN0006	✓
SN0009	✓
SN0007	✓
SN0005	✓
SN0004	✓
SN00011	✓

You will see the number of sensors that you gave as input argument.

To be able to interact with the data that these Remote Things are sending to the Thingworx server, you will need to bind them to a Thing created using a RemoteThing Template on your Thingworx platform.

Binding a remote device (like a Sensor sending data through a microcontroller) can be done in several way on the Thingworx Platform:

1. through the name of the Virtual Thing (when no identifier is set in the constructor for the VirtualThing)
2. Through an identifier.

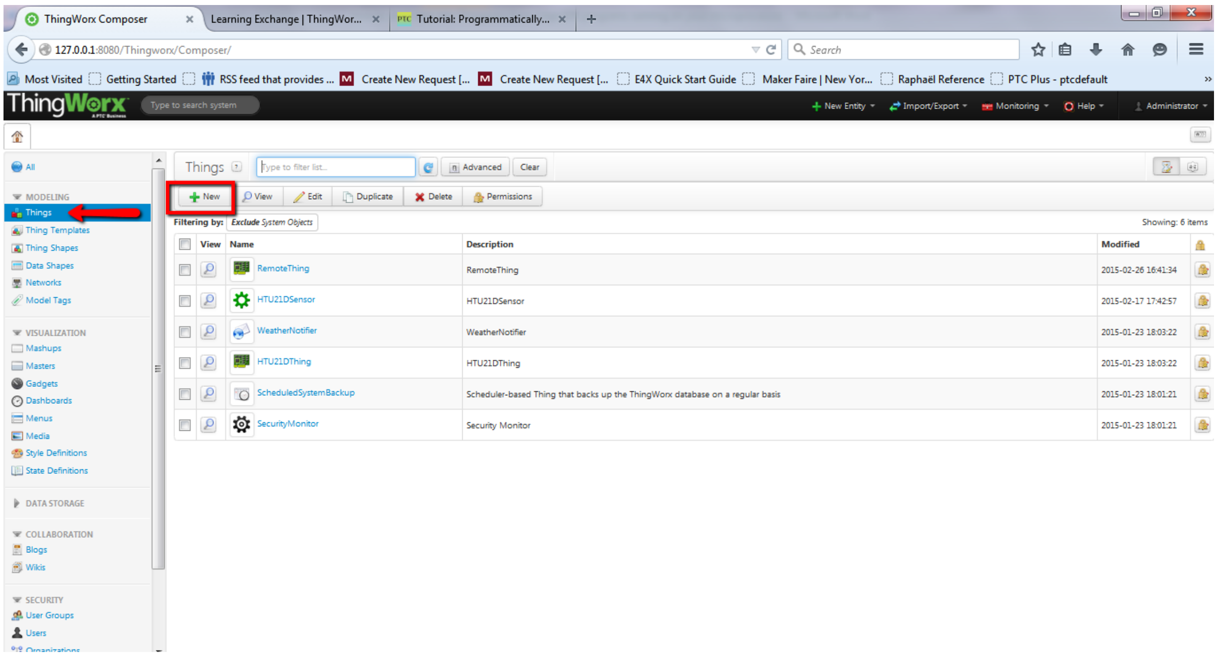
Both of these values are associated with the Remote Thing in this line of code:

```
SteamThing    steamSensorThing    =    new    SteamThing("SteamSensor"    +
sensorID,"Steam   Sensor #" + sensorID,"SN000" + sensorID,client);
                   client.bindThing(steamSensorThing);
//The parameters given in the constructor of the SteamThing class stand for
the SteamThing Name, SteamThing description, SteamThing identifier and the
client the SteamThing will send data to.
```

In this example you will use the second method, because the value for the identifier is present.

Follow these steps to bind a remote SteamSensorThing from your java code to an actual Thing on your Thingworx Server so that you can use its data to later build Mashups or simply to store it into ValueStreams and use these data in data analysis.

- Go to your Thingworx Composer and click the + new button from the Things section to create a new Thing.

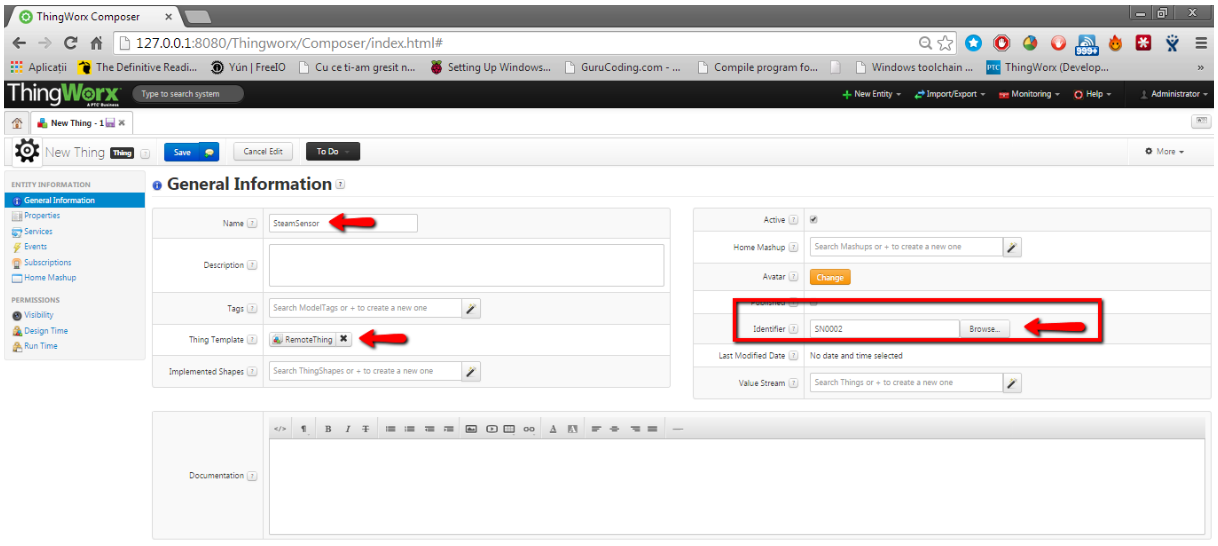


Filtering by: *Exclude System Objects*

Showing: 6 items

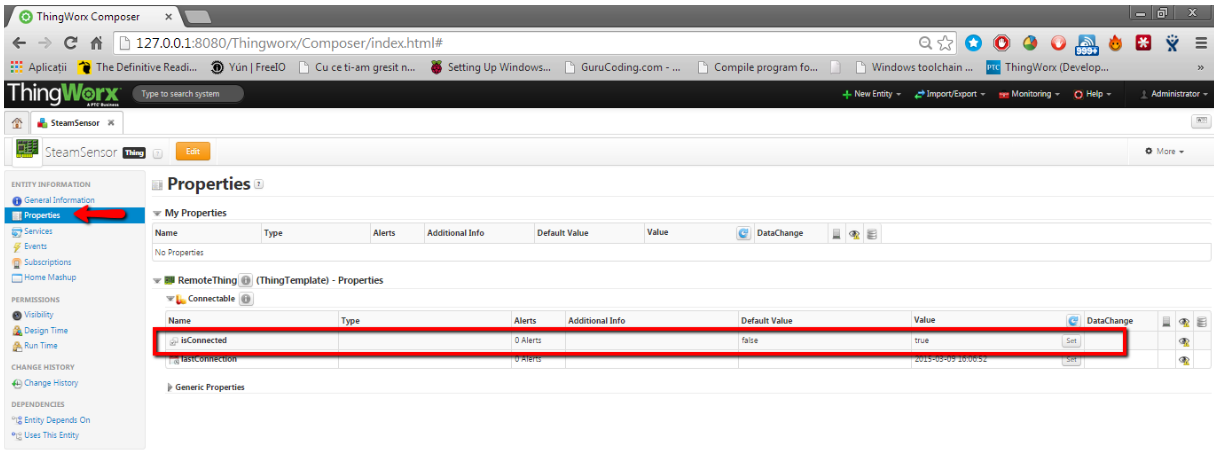
View	Name	Description	Modified
	RemoteThing	RemoteThing	2015-02-26 16:41:34
	HTU21DSensor	HTU21DSensor	2015-02-17 17:42:57
	WeatherNotifier	WeatherNotifier	2015-01-23 18:03:22
	HTU21DThing	HTU21DThing	2015-01-23 18:03:22
	ScheduledSystemBackup	Scheduler-based Thing that backs up the ThingWorx database on a regular basis	2015-01-23 18:01:21
	SecurityMonitor	Security Monitor	2015-01-23 18:01:21

- Enter any name you'd like for the SteamSensorThing. This will store the data from your java code that you would like to bind.
- Choose RemoteThing as Template and save your Thing.
- In the Identifier field, click Browse and select the Identifier you would like to bind.



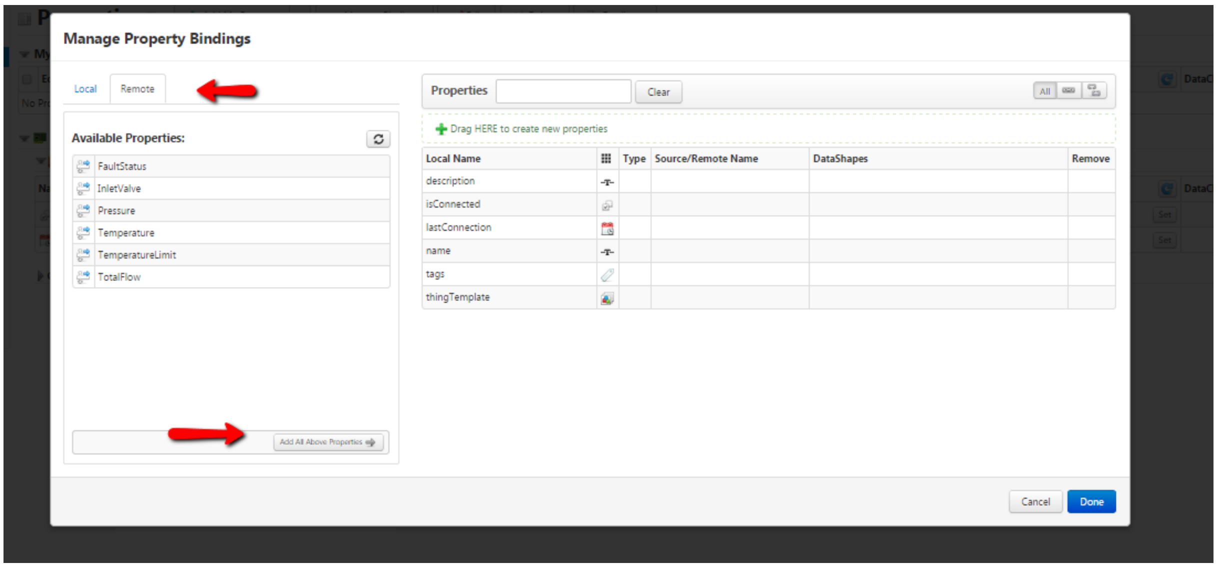
Name: SteamSensor
Description:
Tags: Search ModelTags or + to create a new one
Thing Template: RemoteThing
Implemented Shapes: Search ThingShapes or + to create a new one
Active: ✓
Home Mashup: Search Mashups or + to create a new one
Avatar: Change
Identifier: SN0002 Browse...
Last Modified Date: No date and time selected
Value Stream: Search Things or + to create a new one
Documentation:

- Save the Thing and go to the Properties section. You should see the property isConnected of your Thing appearing as True.



Name	Type	Alerts	Additional Info	Default Value	Value	DataChange
isConnected		0 Alerts		false	true	
lastConnection		0 Alerts			2015-03-09 16:06:52	

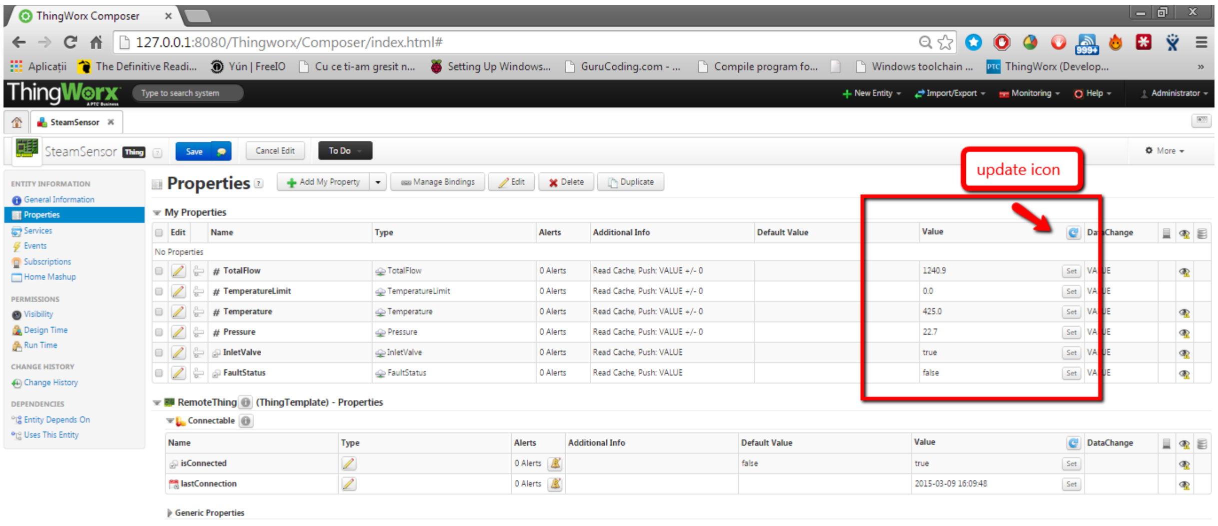
- To bring in the properties that this remote SteamSensor is sending to the Thingworx. Click Edit and go to Manage Bindings → Remote Tab.



Available Properties:
- FaultStatus
- InletValve
- Pressure
- Temperature
- TemperatureLimit
- TotalFlow

Local Name	Type	Source/Remote Name	DataShapes	Remove
description				
isConnected				
lastConnection				
name				
tags				
thingTemplate				

- Click Done and Save the Thing. You will now be able to see all the properties that this SteamSensor is sending from your java code.
- Click on the update icon for the properties. You should now be able to see also the values that are being sent together with the properties names.



My Properties

Edit	Name	Type	Alerts	Additional Info	Default Value	Value	DataChange
No Properties							
	TotalFlow	TotalFlow	0 Alerts	Read Cache, Push: VALUE +/- 0		1240.9	VALUE
	TemperatureLimit	TemperatureLimit	0 Alerts	Read Cache, Push: VALUE +/- 0		0.0	VALUE
	Temperature	Temperature	0 Alerts	Read Cache, Push: VALUE +/- 0		425.0	VALUE
	Pressure	Pressure	0 Alerts	Read Cache, Push: VALUE +/- 0		22.7	VALUE
	InletValve	InletValve	0 Alerts	Read Cache, Push: VALUE		true	VALUE
	FaultStatus	FaultStatus	0 Alerts	Read Cache, Push: VALUE		false	VALUE

RemoteThing (ThingTemplate) - Properties

Connectable

Name	Type	Alerts	Additional Info	Default Value	Value	DataChange
isConnected		0 Alerts		false	true	
lastConnection		0 Alerts			2015-03-09 16:09:48	

Generic Properties

Also, if you’d like to learn how to bind these Remote Things automatically to the Thingworx platform, check out the video How to programmatically create a given number of things and assign to them a specific identifier on the Learning Exchange.

Congratulations you now know how ThingWorx Java Edge SDK is used to send data read from different sensors into the ThingWorx Platform and how to import this data manually on the Platform.

©2015 PTC Inc. The information contained herein is provided for informational use and is subject to change without notice. The only warranties for PTC products and services are set forth in the express warranty statements accompanying such products and services and nothing herein should be construed as constituting an additional warranty. PTC shall not be liable for technical or editorial errors or omissions contained herein. Important Copyright, Trademark, Patent, and Licensing Information: See the About Box, or copyright notice, of your PTC software. 01012015