

# 1. VANTIQ Simulator

## Live Objects Data Generator Project

This tutorial guides a developer to define a Vantiq application that simulates data from different temperature sensors and push them to Live Objects.

This Data Simulator project is defined to be used with the Control Temperature Project that monitors Temperature in Cold Storage Rooms.

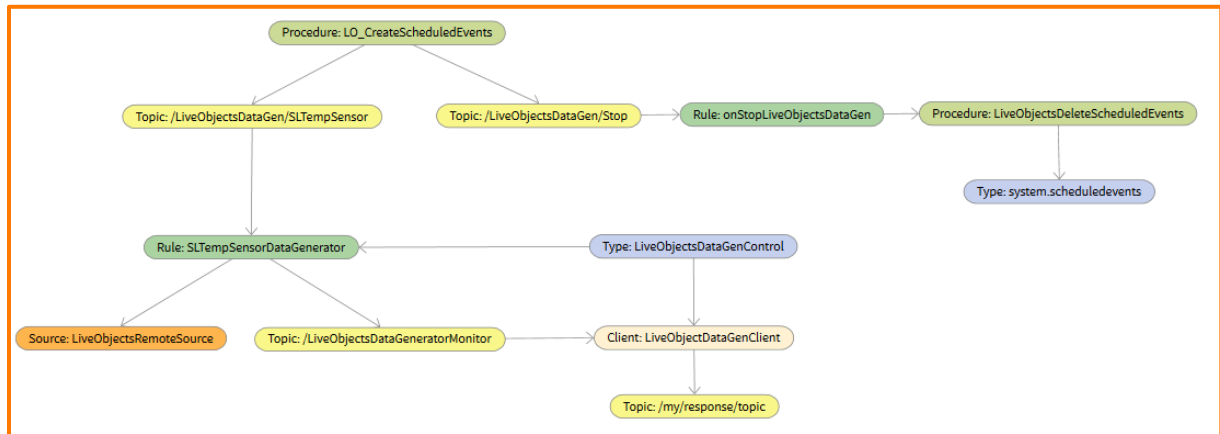
It is assumed that the developer has a working knowledge of the [VANTIQ IDE](#). It is recommended that a new developer completes the lessons in the [Introductory Tutorial](#) before starting this tutorial.

## Summary Table

1. Introduction.....	3
2. Create a new Project .....	4
3. Create Data Types.....	5
A. Create a new data Type LiveObjectsDataGenControl .....	5
B. Create Records to simulate 5 sensors.....	5
C. Create a new DataType LiveObjectsDataMessage.....	6
4. Define a procedure to generate triggers given frequency, for a given period .....	7
5. Create a source to push simulated data to Live Objects.....	8
6. Create a rule to generate data and push them to LiveObjects.....	9
7. Define a rule and a procedure to stop the scheduling .....	11
8. Create a client Web App to launch data simulation.....	12
A. Configure Properties:.....	12
B. Configure Data Objects.....	15
C. Configure Data Streams.....	15
D. Configure Page Properties (Start Page Properties) .....	17
E. Create Start Page with widget selection .....	18
F. Launch LiveObjectsDataGenerator .....	24

## 1. Introduction

### Project Modelo Graphical View of the project



## 2. Create a new Project

Use the **Projects** button, select **Create New Project** and title the project “LiveObjectsDataGen”.

### 3. Create Data Types

#### A. Create a new data Type LiveObjectsDataGenControl

This datatype is used to define the simulated sensors and their status.

For the purpose of the application, the status are defined as follows:

Normal Status for temperature below -19°C.

Risky Status for temperature comprised between -19°C and -15°C.

Critical Status for temperature above -15°C.

Use the **Add** button to select **Type...**

Use the **New Type** button to create the “LiveObjectsDataGenControl” datatype, and add the following properties.

Editing Type 'LiveObjectsDataGenControl'

Save (Ctrl-S) Reset Delete All Instances

Role: standard - a persistent type

Admin Access Level: ars\_readWrite Read write access to all records of this type

User Access Level: ars\_readOnly Read only access to all records of this type

Group By:

Natural Key: No natural keys defined

+ No indexes are defined

Properties

Name	Type	Required	Multi	Encrypted	Scale	--Actions--
SensorId	String	☒	☒	☒		
Status	Integer	☒	☒	☒		

Advanced

#### B. Create Records to simulate 5 sensors

Use the **Add New Record** button to record 5 sensors, with an init status.

Status can be 0 or Normal, 1 for Risky, 2 for Critical.

The user will be able to update the status later through the Web application.

Find Record Results: All Objects of type LiveObjectsDataGenControl

1 (1 - 5 of 5)

<input type="checkbox"/>	SensorId	Status	--Actions--
<input type="checkbox"/>	1	0	
<input type="checkbox"/>	2	2	
<input type="checkbox"/>	3	0	
<input type="checkbox"/>	4	0	
<input type="checkbox"/>	5	0	

### C. Create a new DataType LiveObjectsDataMessage

Use the **Add** button to select **Type...**

Use the **New Type** button to create the “LiveObjectsDataMessage” datatype and add the following properties.

Type: LiveObjectsDataMessage

+ Add New Record

🔍 Show All Records

Properties

Name	Type	Required	Multi	Scale
metadata	Object	✖	✖	
model	String	✖	✖	
streamId	String	✖	✖	
tags	String	✖	✖	
timestamp	String	✖	✖	
value	Object	✖	✖	

#### 4. Define a procedure to generate triggers given frequency, for a given period

Use the **Add** button to select **Procedure**

The procedure triggers events on 2 topics

- 1<sup>st</sup> one : periodic events that will trigger data sending
- 2<sup>nd</sup> one : end of data simulation

Topics are automatically generated.

```
Procedure: LO_CreateScheduleEvents
1 PROCEDURE LO_CreateScheduleEvents (paramconfiguration)
2
3 PUBLISH { name: "TemperatureSE" } TO TOPIC "/LiveObjectDataGen
  /SLTempSensor"
4 SCHEDULE {name: "TemperatureSEEvent", periodic: true, interval:
  paramconfiguration.Frequency}
5
6
7 PUBLISH { name: "StopLiveObjectDataGen" } TO TOPIC "/LiveObjectDataGen
  /Stop" SCHEDULE {interval: paramconfiguration.ScheduleDuration}
8
```

Topic: /LiveObjectsDataGen/SLTempSensor	
Name	/LiveObjectsDataGen/SLTempSensor
Publish Message	<input type="text" value="{}"/>
<button>Publish</button>	

Topic: /LiveObjectsDataGen/Stop	
Name	/LiveObjectsDataGen/Stop
Publish Message	<input type="text" value="{}"/>
<button>Publish</button>	

## 5. Create a source to push simulated data to Live Objects

Use the **Add** button to select **Source** and name it LiveObjectsRemoteSource.

Select **REMOTE** as Source Type.

In Source Properties :

Set the **Server URI** as <https://liveobjects.orange-business.com>

In Request Default Properties :

Set the **Content Type** as application/json

Set the **Response Type** as application/json

Set the **Headers** as {"X-API-KEY": "your LiveObjects API Key"}

Set the **Method** to POST

Editing Source 'LiveObjectsRemoteSource'

Source Type: REMOTE Activation Constraint: >

Source Properties

Property	Value
Keep Active?	<input checked="" type="checkbox"/>
Server URI	<a href="https://liveobjects.orange-business.com">https://liveobjects.orange-business.com</a>
Message Type	-- Choose Type --

Polling Properties

Property	Value
Polling Interval (sec)	0

Authentication Properties

Basic Authentication

Property	Value
Username	solene.queillard@orange.com
Password	*****

Token Authentication

Property	Value
Access Token	
Realm	

Request Default Properties

Property	Value
Path	
Fragment	
Query Parameters	
Content Type	application/json
Response Type	application/json
Headers	{"X-API-KEY": "f50e618413954a199d7d82a"}
Body	
Method	

Source Properties

Property	Value
Keep Active?	<input checked="" type="checkbox"/>
Server URI	<a href="https://liveobjects.orange-business.com">https://liveobjects.orange-business.com</a>
Message Type	-- Choose Type --

Request Default Properties

Property	Value
Path	
Fragment	
Query Parameters	
Content Type	application/json
Response Type	application/json
Headers	{"X-API-KEY": "f50e618413954a199d7d82a"}
Body	
Method	POST



## 6. Create a rule to generate data and push them to LiveObjects

Use the **Add** button to select **Rule**, name it SLTempSensorDataGenerator.

The rule will

- generate temperature values considering the status of the sensor stored in the LiveObjectsDataGenControl datatype.
- format data as a SensingLabs Temperature Sensor
- send simulated data to LiveObjects
- send simulated data to an internal topic that will be used by the client web app to print the generated data

RULE SLTempSensorDataGenerator WHEN PUBLISH OCCURS ON "/LiveObjectsDataGen/SLTempSensor"

```
log.info("SLTempSensorDataGenerator Start ")
try
{
    SELECT FROM LiveObjectsDataGenControl AS dc
    {
        var temp
        if (dc.Status == 2)
        {
            temp = random(-14,0)
        }
        else
        {
            if (dc.Status == 1)
            {
                temp = random(-18,-16)
            }
            else
            {
                temp = random(-28,-20)
            }
        }
    }
    var data = {}
    var datavalue = {}
    var batterylevel = {}
    var tempinfo = {}
    var huminfo = {}
    var datanetwork = {}
    var loradatanetwork = {}
    var metadata = {}

    data.timestamp = now()
    data.model = "model_sensinglabs_senslab_v1"
    datavalue.payload = "034012131415440400"
    batterylevel.value = 10
    batterylevel.unit = "%"
    tempinfo.value = temp
    tempinfo.unit = "°C"
    huminfo.value = 3
    huminfo.unit = "%"
    datavalue.humidity = huminfo
    datavalue.temperature = tempinfo
    datavalue.batterylevel = batterylevel
    data.value = datavalue

    var loradeveui = "AAAA000000000000" + dc.SensorId
    metadata.source = "urn:lora:" + loradeveui
    loradatanetwork.devEUI = loradeveui
    datanetwork.lora = loradatanetwork
    metadata.network = datanetwork
    data.metadata = metadata
    data.tags = ["VantTests", "Temp"]

    var sourcepath = "/api/v0/data/streams/urn:lora:" + loradeveui + "!uplink"
    var mondata = {}
    var internaldatainfo = {}
    internaldatainfo.Time = data.timestamp
    internaldatainfo.Id = loradeveui
    internaldatainfo.Temperature = stringify(tempinfo.value) + tempinfo.unit

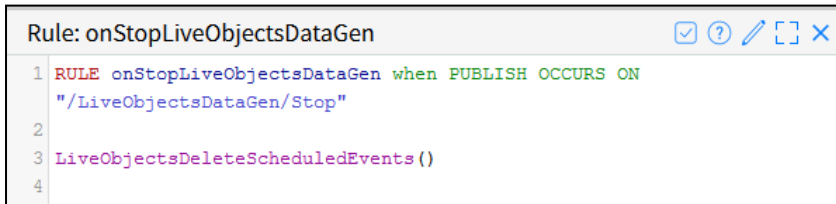
    mondata.data = internaldatainfo

    PUBLISH {message : mondata} to TOPIC "/LiveObjectsDataGeneratorMonitor"

    PUBLISH { body: data } TO SOURCE LiveObjectsRemoteSource USING {path:sourcepath}
}
}
catch (exception)
{
    log.info(stringify(exception))
}
```

## 7. Define a rule and a procedure to stop the scheduling

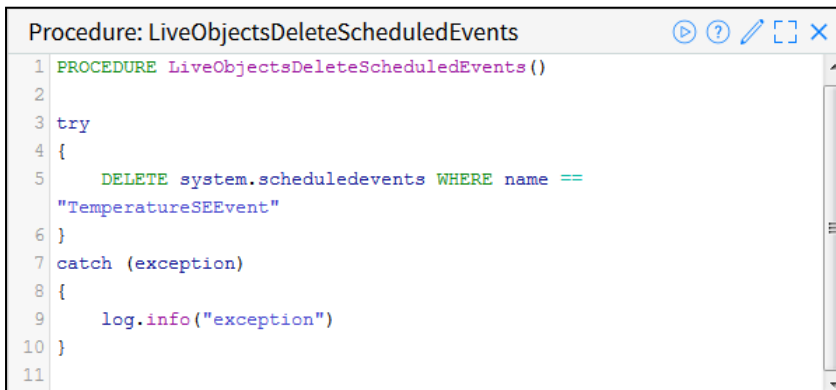
Use the **Add** button, select Rule, add a Rule and name it onStopLiveObjectsDataGen.



```
Rule: onStopLiveObjectsDataGen
1 RULE onStopLiveObjectsDataGen when PUBLISH OCCURS ON
2   "/LiveObjectsDataGen/Stop"
3 LiveObjectsDeleteScheduledEvents()
4
```

The Rule will be triggered on an event on the topic /LiveObjectsDataGen/Stop and calls a Procedure to stop the schedule events.

Use the **Add** button, select **Procedure**, add a Procedure and name it LiveObjectsDeleteScheduledEvents.



```
Procedure: LiveObjectsDeleteScheduledEvents
1 PROCEDURE LiveObjectsDeleteScheduledEvents()
2
3 try
4 {
5   DELETE system.scheduledevents WHERE name ==
6   "/TemperatureSEEvent"
7 }
8 catch (exception)
9 {
10   log.info("exception")
11 }
```

This procedure will also be called by the Client Web App.

## 8. Create a client Web App to launch data simulation

### A. Add document:

Add the Orange Logo in your workspace (the logo that will be used for the header of the web app)

In Show->Advanced-> Documents, add the OrangeLogo image with the following link  
<public/images/OrangeLogo.png>.

### B. Create Client:

Use the **Add** button, select **Client**, add a new one and name it LiveObjectsDataGenerator.  
Chose Design for browser Layout, and click OK.

### C. Configure Properties:

Add **Custom Code** to define the Orange Header and the Sensor DropList

```

$('.navbarIcon').css("background-image", "url(https://dev.vantiq.com/ui/docs/NS/YourNamespace/images/OrangeLogo.png)");
$(".navbarIcon").css("width", "60px");
$(".navbarIcon").css("height", "60px");
$('.navbarIcon').css("background-repeat", "no-repeat");

//$("#rtcTitle").remove();
$(".main-header .logo").css({'height':'60px'}); // Hauteur du bandeau du haut
$(".navbar").css("background-color", "#000000"); // Couleur de la barre de navigation
$(".logo").css("background-color", "#000000"); // Couleur du fond sous le logo + hauteur du bandeau
$(".skin-blue .main-header .navbar .nav > li > a").css("color", "#444444"); // Theme de la bare du haut, couleur de police du nom du user et
Namespace VANTIQ, au format CSS,

function refreshLiveObjectsDataGenControl(client)
{ var http = new Http();
  http.setVantiqueUrlForResource("LiveObjectsDataGenControl");
  //
  // Add the Authorization header to the request
  //
  http.setVantiqueHeaders();
  var args = {};
  //
  // Execute the asynchronous server request. This expects 4 parameters:
  //
  // resourceId: The "_id" of the object being loaded
  // parameters: "null" or an object containing the parameters for this request
  // successCallback: A callback function that will be driven when the request completes
  //                  successfully (i.e. a status code of 2XX)
  // failureCallback: A callback function that will be driven when the request does not complete
  //                  successfully.
  //
  http.select(args,function(response)
  {
    //
    // At this point "response" is the object found
    //
    console.log("SUCCESS: " + JSON.stringify(response));
    client.sendClientEvent("CurrentLiveObjectsDataGenControlStream", response);

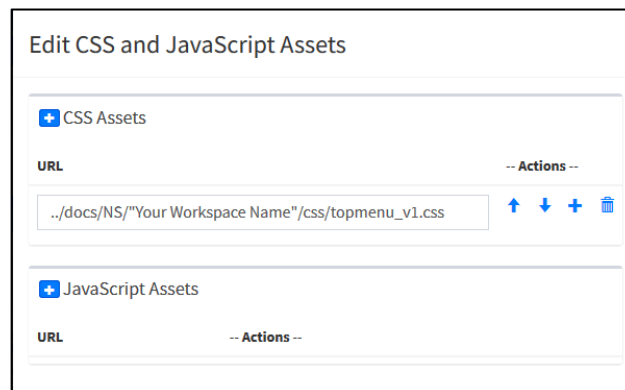
    var SensorDL = client.getWidget("SensorDropList");
    var enumList = [];

    var arrayLength = response.length;
    if (arrayLength > 0)
    {
      enumList.push({value:0, label: "Select a SensorId"});
      for (var i = 0; i < arrayLength; i++)
      {
        enumList.push({value:response[i]._id, label: "SensorId " + response[i].SensorId});
      }
    }
    else
    {
      enumList.push({value:0, label: "No Sensor defined"});
    }
    SensorDL.enumeratedList = enumList;
    Binding.applyChanges();
  },
  function(errors)
  {
    //
    // This call will format the error into a popup dialog
    //
    client.showHttpErrors(errors,"Doing a select on a single LiveObjectsDataGenControl");
  });
}

```

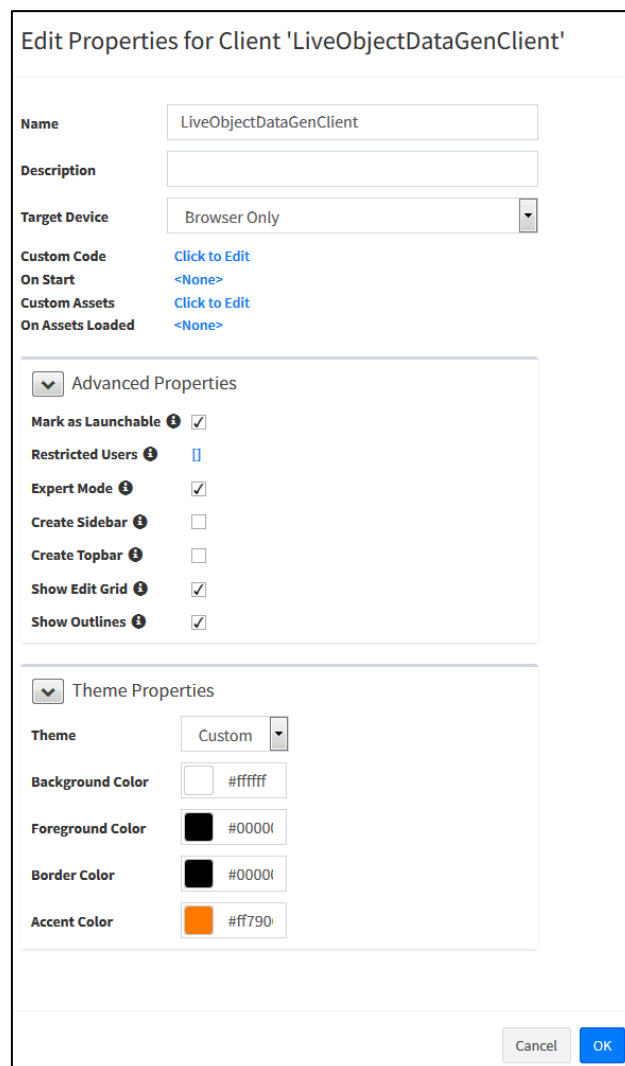
Add Custom Asset to add a specific .css if wanted .

The .css needs to be previously added in the documents of the workspace (Show -> Advanced -> Documents) with the following link public/css/topmenu\_v1.css.



The screenshot shows a dialog box titled "Edit CSS and JavaScript Assets". It contains two sections: "CSS Assets" and "JavaScript Assets". The "CSS Assets" section has a "URL" field with the text ".../docs/NS/"Your Workspace Name"/css/topmenu\_v1.css" and a "-- Actions --" button with icons for up, down, add, and delete. The "JavaScript Assets" section is currently empty with a "URL" field and a "-- Actions --" button.

Set **Mark as Launchable** and **Expert Mode** in Advanced Properties , and Theme Properties as defined below.



The screenshot shows a dialog box titled "Edit Properties for Client 'LiveObjectDataGenClient'". It contains several fields: "Name" (LiveObjectDataGenClient), "Description", "Target Device" (Browser Only), "Custom Code" (Click to Edit), "On Start" (<None>), "Custom Assets" (Click to Edit), and "On Assets Loaded" (<None>). Below these are two expandable sections: "Advanced Properties" and "Theme Properties". The "Advanced Properties" section has checkboxes for "Mark as Launchable" (checked), "Restricted Users" (empty), "Expert Mode" (checked), "Create Sidebar" (unchecked), "Create Topbar" (unchecked), "Show Edit Grid" (checked), and "Show Outlines" (checked). The "Theme Properties" section has a "Theme" dropdown (Custom), "Background Color" (#ffffff), "Foreground Color" (#000000), "Border Color" (#000000), and "Accent Color" (#ff7900). At the bottom right are "Cancel" and "OK" buttons.

## D. Configure Data Objects

Add a Property in **Data Objects / Client.data**

- a string LastMessageData for the print of the last generated data, with default value "Waiting..."

Editing Data Object: client.data

+Add a Property +Add a 'Typed Object' based on Type: -- Choose Type --

Name	Data Type	Default Label	Default Value	Is Array?	-- Actions --
stMessageData	String	LastMessageData	Waiting...	<input type="checkbox"/>	+ ⌂ ⚡ 🗑

Cancel Save Save and Exit

Add a Property in **Data Objects / page.data** for Page Start

- an integer Frequency , with default value 5 ( for 5 seconds)
- an integer ScheduleDuration, with default value 30 ( 30 seconds)
- a string SelectedSensorId
- an integer SelectedStatus

Editing Data Object: page.data for page 'Start'

+Add a Property +Add a 'Typed Object' based on Type: -- Choose Type --

Name	Data Type	Default Label	Default Value	Is Array?	-- Actions --
Frequency	Integer	Frequency	5	<input type="checkbox"/>	+ ⌂ ⚡ 🗑
ScheduleDuration	Integer	ScheduleDuration	60	<input type="checkbox"/>	+ ⌂ ⚡ 🗑
SelectedSensorId	String	SelectedSensorId	Default Value	<input type="checkbox"/>	+ ⌂ ⚡ 🗑
SelectedStatus	Integer	SelectedStatus	Default Value	<input type="checkbox"/>	+ ⌂ ⚡ 🗑

Cancel Save Save and Exit

## E. Configure Data Streams

Add in Data Streams

- a publish event **LiveObjectsDataGenMonitorStream** , on topic /LiveObjectsDataGeneratorMonitor

Edit Data Stream

Data Stream Name
LiveObjectDataGenMonitorStream

☐ On Data Changed
☒ On Publish Event
☐ On Source Event
☐ On Timed Query
☐ On Client Event
☐ On Event Stream

On Publish Event
Topic

/LiveObjectDataGeneratorMonitor

Group By (Optional)

Enter property

On Data Arrived
[Click to Edit](#)

Cancel
Save

with the following code to execute on data arrival:

Edit JavaScript for the 'onDataArrived' event on DataStream 'LiveObjectDataGenMonitorStream'

```

//
// This function is called on Data Stream 'LiveObjectDataGenMonitorStream'
// whenever new data arrives on the stream. (Note that 'this' points to the
// DataStream object.)
//
function DataStream_LiveObjectDataGenMonitorStream_onDataArrived(client,data)
{
1  var msg = {};
2
3  msg.Message = JSON.stringify(data.message.data);
4  client.sendClientEvent("LiveObjectPublishMessageClientStream", msg);
5  client.data.LastMessageData = JSON.stringify(data.message.data, null, 2);
6  Binding.applyChanges();
}

```

- a client event **LiveObjectsPublishMessageClientStream** to print the message

Edit Data Stream

Data Stream Name
LiveObjectPublishMessageClientStream

☐ On Data Changed
☐ On Publish Event
☐ On Source Event
☐ On Timed Query
☒ On Client Event
☐ On Event Stream

Client Event
☒ Get schema from Type
☐ Get schema from client.data object
Data Type

LiveObjectDataMessage

On Data Arrived
[Click to Edit](#)

Cancel
Save

with the following code to execute on data arrival:

'LiveObjectPublishMessageClientStream'

```

//
// This function is called on Data Stream 'LiveObjectPublishMessageClientStream'
// whenever new data arrives on the stream. (Note that 'this' points to the
// DataStream object.)
//
function DataStream_LiveObjectPublishMessageClientStream_onDataArrived(client,data)
{
1  console.log(JSON.stringify(data));
}

```



**Edit Data Stream**

**Data Stream Name**  
CurrentLiveObjectDataGenControlStream

☐ On Data Changed  
☐ On Publish Event  
☐ On Source Event  
☐ On Timed Query  
☒ On Client Event  
☐ On Event Stream

**Client Event**

☒ Get schema from Type  
☐ Get schema from client.data object

**Data Type**  
LiveObjectDataGenControl

**On Data Arrived** <None>

Cancel Save

- an on Timed Query event **LiveObjectsDataGenStream** to get the information from LiveObjectsDataGenControl DataType (Query every 10 seconds)

**Edit Data Stream**

**Data Stream Name**  
LiveObjectDataGenStream

☐ On Data Changed  
☐ On Publish Event  
☐ On Source Event  
☒ On Timed Query  
☐ On Client Event  
☐ On Event Stream

**Timed Query**

**Data Type**  
LiveObjectDataGenControl

**Update Interval (seconds)**  
10

**Group By (Optional)**  
-- Choose Property --

☐ Limit maximum number of records to return  
☐ Use Advanced Query

**Basic Query Constraints**

SensorId (String) Any  
 Status (Integer) Any

**On Data Arrived** <None>

Cancel Save

- a Client Event DataStream CurrentLiveObjectsDataGenControlStream used in the custom code for handling the sensor drop list.

**Edit Data Stream**

**Data Stream Name**  
CurrentLiveObjectsDataGenControlStream

☐ On Data Changed  
☐ On Publish Event  
☐ On Source Event  
☐ On Timed Query  
☒ On Client Event  
☐ On Event Stream

**Client Event**

☒ Get schema from Type  
☐ Get schema from client.data object

**Data Type**  
LiveObjectsDataGenControl

**On Data Arrived** <None>

Cancel Save

## F. Configure Page Properties (Start Page Properties)

Edit **Properties** on **Page Start** and set on On Start

Edit JavaScript for the 'onStart' event on page 'Start'

```
// This function is called when page 'Start' first begins executing. (Note  
// that 'this' points to the Page object. 'parameters' is passed by calling  
// client.gotoPage or client.returnToCallingPage.)  
//  
function Client_Start_onStart(client,parameters)  
{  
1 refreshLiveObjectsDataGenControl(client);  
}
```

## G. Create Start Page with widget selection

### ▪ Data Generation

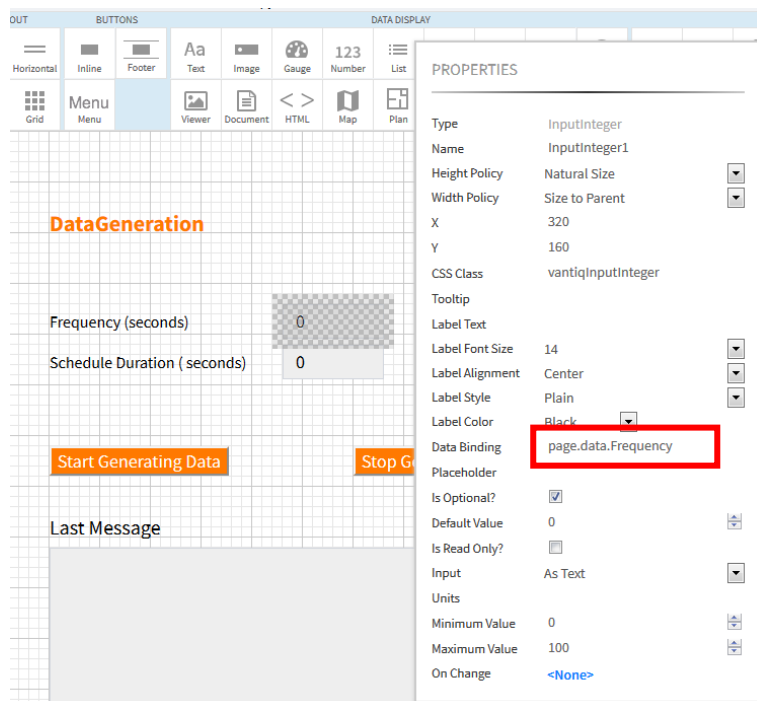
Add **Static Text** box , with Text “DataGeneration”, Font Size 22, Font Weight Bold , Font Color Custom #ff7900

Add **Static Text** box with Text “Frequency (seconds)” , Font Size 16.

Add on its right an **Integer Data Input** with DataBinding to “page.data.Frequency”

Add a **Static Text** box with Text “Schedule Duration (seconds)”, Font Size 16.

Add on its right an **integer Data Input** with DataBinding to “page.data.ScheduleDuration”



This enables to set the frequency and duration of data generation.

Add a **inline button** with button label “Start Generating Data” , Button Background Color set to Custom #ff7900 , Button Label Color set to White, and on Click function that calls the LO\_CreateScheduledEvents procedure (code below).

```
//
// Create an instance of the Http class to execute our server request
//
var http = new Http();
//
// Build the URL needed to do an "execute" of a Procedure
//
http.setVantiqUrlForSystemResource("procedures");
//
// Add the Authorization header to the request
//
http.setVantiqHeaders();
//
// Set the Procedure arguments
//
var args= [
    { "Frequency": page.data.Frequency*1000,
      "ScheduleDuration": page.data.ScheduleDuration*1000
    }
];
//
// Execute the asynchronous server request. This expects 4 parameters:
//
// procedureArguments: The procedure arguments.
// procedureName: The fully-qualified name of the Procedure.
// successCallback: A callback function that will be driven when the request completes
//                  successfully (i.e. a status code of 2XX)
// failureCallback: A callback function that will be driven when the request does not complete
//                  successfully.
//
http.execute(args,"LO_CreateScheduledEvents",function(response)
{
    //
    // At this point "response" is results of the Procedure call
    //
    console.log("SUCCESS: " + JSON.stringify(response));
},
function(errors)
{
    //
    // This call will format the error into a popup dialog
    //
    client.showHttpErrors(errors,"Executing 'LO_CreateScheduledEvents'");
});
```

Add a **inline button** with Button Label “Stop GeneratingData”, Button Background Color set to Custom #ff7900 , Button Label Color set to White, and on Click function that calls the LiveObjectsDeleteScheduledEvents procedure (code below).

```
//
// Create an instance of the Http class to execute our server request
//
var http = new Http();
//
// Build the URL needed to do an "execute" of a Procedure
//
http.setVantiqUrlForSystemResource("procedures");
//
// Add the Authorization header to the request
//
http.setVantiqHeaders();

//
// Set the Procedure arguments
//
var args = {};

//
// Execute the asynchronous server request. This expects 4 parameters:
//
// procedureArguments: The procedure arguments.
// procedureName: The fully-qualified name of the Procedure.
// successCallback: A callback function that will be driven when the request completes
//                  successfully (i.e. a status code of 2XX)
// failureCallback: A callback function that will be driven when the request does not complete
//                  successfully.
//
http.execute(args,"LiveObjectsDeleteScheduledEvents",function(response)
{
    //
    // At this point "response" is results of the Procedure call
    //
    console.log("SUCCESS: " + JSON.stringify(response));
    client.data.LastMessageData = "Waiting...";
},
function(errors)
{
    //
    // This call will format the error into a popup dialog
    //
    client.showHttpErrors(errors,"Executing 'LiveObjectsDeleteScheduledEvents'");
});
```

Add a **Static Text** box with Text “Last Message”, Font Size 20.

Add a **vantiqMultilineInput**, with Width Policy set to Explicit, and Width to 540 , with DataBinding to “client.data.LastMessageData” to print the data generated.

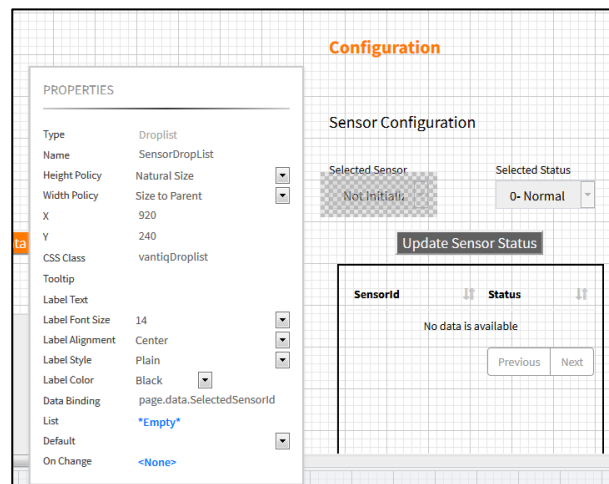
### ■ Sensors Configuration

Add **Static Text**, with Text “Configuration”, Font Size 22, Font Weight Bold , Font Color Custom #ff7900

Add **Static Text**, with Text “Sensor Configuration”.

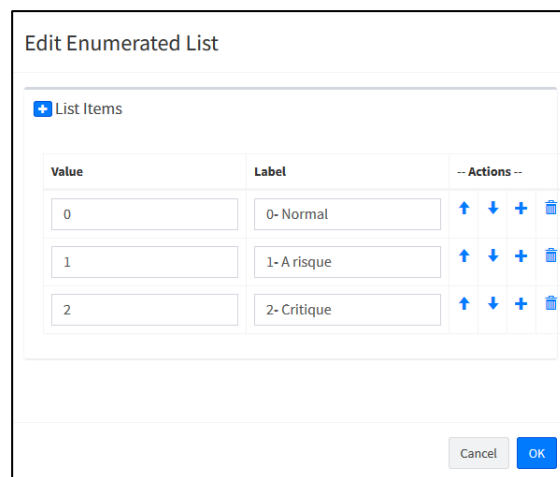
Add a **Label**, with Label Text “Selected Sensor”, Font Size 14.

Add a **dropList** , just below, with Name “SensorDropList”, with DataBinding to “page.data.SelectedSensorId”



Add a **Label** on the right, with Label Text “Status”, Font Size 14.

Add a **dropList** just below with DataBinding to “page.data.SelectedStatus”, and Enumerated list set to 0-Normal,1-Risky,2-Critical :



Add a **Inline Button** , with Button Label “Update SensorStatus” to set the status of Selected Sensor, with onclick function below.

```
//
// Create an instance of the Http class to execute our server request
//
var http = new Http();
//
// Build the URL needed to do an "update" on our Type
//
http.setVantiqUrlForResource("LiveObjectsDataGenControl");
//
// Add the Authorization header to the request
//
http.setVantiqHeaders();
    var updateData = {};
    updateData.Status = page.data.SelectedStatus;
//
// Execute the asynchronous server request. This expects 4 parameters:
//
// data: The object being inserted.
// parameters: "null" or an object containing the parameters for this request
// successCallback: A callback function that will be driven when the request completes
//                successfully (i.e. a status code of 2XX)
// failureCallback: A callback function that will be driven when the request does not complete
//                successfully.
//
http.update(updateData,page.data.SelectedSensorId,function(response)
{
    //
    // At this point "response" is the updated object
    //
    console.log("SUCCESS: " + JSON.stringify(response));
},
function(errors)
{
    //
    // This call will format the error into a popup dialog
    //
    client.showHttpErrors(errors,"Doing an update of sensor status 2");
});

ds = client.getDataStreamByName("LiveObjectsDataGenStream");
ds.restart();
```

Add a **Data Table** to see the current Status of the different Sensors,  
with DataStream set to “LiveObjectsDataGenStream”  
with Columns : SensorId, Status

PROPERTIES

Type

DataTable

Name

DataTable27

Height Policy

Natural Size

Width Policy

Explicit

X

950

Y

397

Width

500

CSS Class

vantiqDataTable

Tooltip

Data Stream

LiveObjectsDataGenStream

Columns

SensorId,Status

Border Thickness

2

Rows Per Page

5

Header Font Face

inherit

Header Font Size

14

Header Font Style

Plain

Header Font Color

Black

Cell Font Face

inherit

Cell Font Size

14

Cell Font Style

Plain

Cell Font Color

Black

On Select

<None>

On Deselect

<None>

## H. Launch LiveObjectsDataGenerator

Go to <https://dev.vantiq.com/ui/rtc/index.html#/home>.

Launch LiveObjectsDataGenerator Client.

Configure the sensors status, and press the button Start Generating Data.

orange

LiveObjectsDataGenerator

DataGeneration

Frequency(seconds)

5

Schedule Duration (seconds)

30

Start Generating Data

Stop Generating Data

Last Message

```
{
  "Time": "2018-10-23T13:00:27.007Z",
  "Id": "AAAA000000000001",
  "Temperature": "-11°C"
}
```

Configuration

Sensor Configuration

Selected Sensor

Select a Se

Status

2- Critical

Update Sensor Status

Sensorid	Status
1	2
2	0
3	0
4	0
5	0

Previous

1

Next

In your Live Objects account, in Data section, you should see the incoming stream from your devices.

Date	Source	Value	Fcnt	Port	Rssi, Snr, Sf	Tags	Network signal
11/05/2018 11:37:15 AM	<a href="#">urn:lora:AAAA0000000000000005</a>	034012131415440400	Q	-	-	Temp VantTests	-
11/05/2018 11:37:15 AM	<a href="#">urn:lora:AAAA0000000000000004</a>	034012131415440400	Q	-	-	Temp VantTests	-
11/05/2018 11:37:15 AM	<a href="#">urn:lora:AAAA0000000000000003</a>	034012131415440400	Q	-	-	Temp VantTests	-
11/05/2018 11:37:13 AM	<a href="#">urn:lora:AAAA0000000000000002</a>	034012131415440400	Q	-	-	Temp VantTests	-
11/05/2018 11:37:13 AM	<a href="#">urn:lora:AAAA0000000000000001</a>	034012131415440400	Q	-	-	Temp VantTests	-



The details of the data are as shown below.

```
{
  "metadata": {
    "connector": "http",
    "source": "urn:lora:AAAA000000000005"
  },
  "streamId": "urn:lora:AAAA000000000005!uplink",
  "created": "2018-11-05T10:37:16.426Z",
  "extra": null,
  "location": null,
  "model": "model_sensinglabs_senslab_v1",
  "id": "5be01d5c7676a73dde1710ad",
  "value": {
    "payload": "034012131415440400",
    "temperature": {
      "unit": "°C",
      "value": -24
    },
    "humidity": {
      "unit": "%",
      "value": 3
    },
    "batterylevel": {
      "unit": "%",
      "value": 10
    }
  },
  "timestamp": "2018-11-05T10:37:15.743Z",
  "tags": [
    "VantTests",
    "Temp"
  ]
}
```