Writing A Virtual Sensor For GSN

with examples

Written by Ali Salehi

In GSN, a virtual sensor is a piece of *Java* code which acts in the final stage of data processing. A virtual sensor simply sits between the wrapper and the data publishing engine. The virtual sensor is actually performs the last processing on the data. The inputs of a virtual sensor are specified in the Virtual Sensor Descriptor (VSD) file. Once a virtual sensor is loaded, GSN automatically prepares a pool for each virtual sensor. The number of the virtual sensor instances inside a pool can be specified in the virtual sensor bases inside the VSD file of each virtual sensor [xml element]. The virtual sensor simply extends the **`gsn.vsensor.AbstractVirtualSensor`** class which in turn implements (partially) the **`gsn.vsensor.VirtualSensor`** Interface. Any virtual sensor needs to implement/ override 3 or 4 methods from the **`AbstractVirtualSensor`** :

```
public class AnyVS extends AbstractVirtualSensor {
    public void initialize ( HashMap map ) {...}
    public void dataAvailable ( String inputStreamName,
                                StreamElement streamElement ) {...}
    public void dataFromWeb ( String data ) {...}
    public void finalize ( HashMap map ) {...}
}
```

In the aforementioned methods, just the `dataAvailable` method is mandatory and the rest have default implementations in the `AbstractVirtualSensor` class. Lets start by showing the code for the **`BridgeVirtualSensor`** :

```
public void initialize ( HashMap map ) { super.initialize ( map ); }
public void dataAvailable (String inputStreamName, StreamElement data){
  dataProduced ( data );
}
public void dataFromWeb ( String data ) { /* empty */}
public void finalize ( HashMap map ) {super.finalize(map);}
```

As it can be seen, the only thing the **`BridgeVirtualSensor`** does is to forward whatever it receives [in the `dataAvailable` method] to a method called `dataProduced` which is actually a helper method in `AbstractVirtualSensor`.

Going in more details, while GSN loading a virtual sensor and just before using it, GSN prepares a pool of instances of the processing class of the virtual sensor [specified in the element of the XML configuration file].

This pool is useful for two reasons :

1. Performance.

2. Limiting the resources used by a virtual sensor, so that a virtual sensor can't kill the GSN container.

In order to put an instance of the specified virtual sensor processing class into the pool, GSN first calls the initialize method with a context variable containing the following data :

Variable Name	Type
VirtualSensorPool.CONTAINER	Container
VirtualSensorPool.VSENSORCONFIG	VSensorConfig

In addition to above variables, the virtual sensor can have initialization parameters (e.g., ChartVirtualSensor). The initialization paramters of a virtual sensor is specified in the VSD file like below :

```
<processing-class>
   <class-name>gsn.vsensor.ChartVirtualSensor</class-name>
       <init-params>
             <param name="InputStream">DATA</param>
             <param name="Title">Light-Temperature Status</param>
        </init-params>
</processing-class>
```

And accessed in the virtual sensor class like below :

```
Collection<KeyValue> params = virtualSensorConfiguration.
                                      getMainClassInitialParams ();
for ( KeyValue param : params ) {
      if (( (String) param.getKey() ).trim().equalsIgnoreCase ("param1") ) {...}
      ....
}
```

After discussing how to access the initialization parameters we come to the actual implementation of the initialize and finalize methods. In the initialize method, the virtual sensor is responsible in acquiring all the resources it needs in order to start processing the input streams. This method as showed above can use VSD to configure itself. The implementation of these methods [initialize and finalize] should always start by calling the parent implementation [e.g., calling the same method from the parent using the super reference].

In the implementation of the finalize method, the virtual sensor should release all the resources it acquired during its initialization process. Note that as I said before, you should start the implementation of the finalize method by calling the finalize method from the super class. The finalize method is going to be called whenever GSN wants to remove a virtual sensor which typically happens when :

1. User shuts down the GSN container.

2. User removes the virtual sensor which GSN is running.

Implementing the `dataAvailable` method :

As mentioned in other documents, this method is called by GSN whenever an input stream has a data. The name of the input stream which has the data is provided in the first argument while the actual data is specified in the second argument. For example, if we have the following configuration fragment in a VSD file :

```
<input-stream name="DATA" ...>
   ...
   select NAME from StreamSource1
</input-stream>
```

the `dataAvailable` method is going to be called by GSN with the input stream name `DATA` and the data item containing one attributed called `NAME`.

Note : if several data items produced from one or more input streams used by a virtual sensor and if there is already an instance of the processing class is actively processing some other data, GSN uses another instance from the pool [the first goal of the pool] and if the pool is full [all the instances are busy with processing the data items] GSN decides depending on the current memory and usage status on whether buffer the data or drop it[second goal of the pool].

Once a data time received by a virtual sensor [the `dataAvailable` method called], it is the responsibility of the virtual sensor's processing class [specified by the `<processing-class>` element of VSD file] to determine when to publish data [calling the `dataProduced` method]. In the `BridgeVirtualSensor` example, the `dataProduced` method is called for each stream element received by the virtual sensor.

IMPORTANT : The output of a virtual sensor MUST confirm exactly with the defined output in the VSD file associated with the virtual sensor.

IN THE CASE OF THE `BridgeVirtualSensor`, THIS CLASS JUST FORWARDS THE DATA THUS IT IS THE RESPONSIBILITY OF THE INPUT-STREAM TO PRODUCE THE DATA IN THE CORRECT STRUCTURE AS SPECIFIED IN THE VSD FILE.