Java Primitive Operator

If you have opted to write a Java primitive operator, make sure you set the JAVA_HOME environment variable as follows :

```
export JAVA_HOME=/opt/ibm/InfoSphere_Streams/4.0.0.0/java

export STREAMS_SPLPATH=/opt/ibm/InfoSphere_Streams/4.0.0.0/toolkits

export PATH=/opt/ibm/InfoSphere_Streams/4.0.0.0/java/bin:$PATH
```

You may also add the above lines to the ~/.bashrc file to avoid setting the variable every time you log in.

In a Java primitive operator, after the operator is written in Java, it is compiled and the compiled Java class file is manually copied to the namespace provided in the main SPL file.

The Java Operator API is in the InfoSphere Streams library `com.ibm.streams.operator.jar` , which is located in **$STREAMS_INSTALL**/lib. The Java Operator API depends on Apache Commons Math 2.1 library (`commons-math-2.2.jar`) which is located in **$STREAMS_INSTALL**/ext/lib. When developing Java operators using the Java Operator API, the `com.ibm.streams.operator.jar` and `commons-math-2.2.jar` libraries are required in the class path. Sample operators and patterns including source are supplied in the InfoSphere Streams library `com.ibm.streams.operator.samples.jar` , which is located in **$STREAMS_INSTALL**/lib.

Writing Java Primitive Operators is easier with InfoSphere Streams Studio (Eclipse). After creating SPL Application Project, You can right click on the project entry, following New > Java Primitive Operator, entering the name of the operator, and click on finish. Then the studio creates a sample code for you.

Example 1:

Create a folder "~/Example/Test", and include the main SPL file in the folder. Create subfolder called "my.op", and "my.op/Test"

1) Write the main SPL code:

```
// ****************************************************************
// namespace for the operator
use my.op::Test;

composite Main {
  graph
     stream<int32 count> Input = Beacon() {
     logic  state: mutable int32 n = 0;
     param  iterations : 10u;
     output Input          : count = n++;
    }
    // “Test” is the user-defined operator
    stream<int32 count> Output = Test(Input) {
    }

    () as Writer = FileSink(Output) {
      param  file : "result.csv";
    }
}
// *************************************************************
```

2) Change to the folder “my.op/Test”, and run the command “spl-make-operator --kind java”. An XML file “Test.xml” is generated. Change the class name to “Test” under the “execution settings” in Test.xml.

3) Write a Java operator.
The operator takes in a tuple of integers and prints out those whose value is greater than 3.
To fetch values from the input stream attributes, use getters as shown below.

```
//*************************************************************************

import com.ibm.streams.operator.AbstractOperator;
import com.ibm.streams.operator.OperatorContext;
import com.ibm.streams.operator.OutputTuple;
import com.ibm.streams.operator.StreamingInput;
import com.ibm.streams.operator.Tuple;
import com.ibm.streams.operator.StreamingOutput;

public class Test extends AbstractOperator {

  @Override
  public void initialize(OperatorContext context) throws Exception {
    super.initialize(context);
  }

   public void process(StreamingInput stream, Tuple tuple) throws Exception {

       final StreamingOutput<OutputTuple> output = getOutput(0);

        // Submit any tuple with count  greater than 3
        if( tuple.getInt("count") > 3 )
            output.submit(tuple);
   }
}

// *************************************************************************
```

4) Compile the code:

```
javac –cp
/opt/ibm/InfoSphere_Streams/4.0.0.0/lib/com.ibm.streams.operator.jar:/opt/ibm/InfoSphere_Streams/4.0.0.
0/ext/lib/commons-math-2.2.jar Test.java
```

Copy the "Test.class" to the folder "~/Example/Test/my.op/Test" if you wrote and compiled the Java in some other folder location.

5) Compile the SPL code using the command "sc –T –M Main" if you want it as a standalone application, else, "sc –M Main" for a distributed one.

Example 2:

The main SPL code:

```
// ***************************************************
use my.op::Test;
composite Main {
  graph
    stream<int32 count> Input = Beacon() {
      logic  state: mutable int32 n = 0;
      param  iterations : 10u;
      output Input          : count = n++;
    }
    stream<int32 count, rstring name> Output = Test(Input) {
    }
    () as Writer = FileSink(Output) {
      param  file : "result.csv";
    }
}
// ***************************************************************
```

The Java primitive operator:

If the input and the output tuple schema are different, additional attributes can be of the output tuple can be assigned a value using the setters as shown below. This code takes in an input stream of the form <int32 count> and submits a tuple to the output port of the form <int32 count, rstring name>.

```
//***************************************************

import com.ibm.streams.operator.AbstractOperator;
import com.ibm.streams.operator.OperatorContext;
import com.ibm.streams.operator.OutputTuple;
import com.ibm.streams.operator.StreamingInput;
import com.ibm.streams.operator.Tuple;
import com.ibm.streams.operator.StreamingOutput;

public class Test extends AbstractOperator {

public int i;

  @Override
  public synchronized void initialize(OperatorContext context) throws Exception {
    super.initialize(context);
    i = 0;
  }

  public void process(StreamingInput stream, Tuple tuple) throws Exception {

      StreamingOutput<OutputTuple> output = getOutput(0);
      OutputTuple outputTuple = output.newTuple();

      // Submit any tuple with "count" greater than 3.
      if( tuple.getInt("count") > 3 )
      {
          outputTuple.setInt("count", tuple.getInt("count"));
          outputTuple.setString("name", "a"+i);
          output.submit(outputTuple);

          i++;
      }
  }
}

//**********************************************************
```

Example 3: Using user-defined parameter for primitive operator

1) The main SPL code

```
// **********************************************************
use my.op::Test;
composite Main {
  graph
    stream<int32 count> Input = Beacon() {
      logic  state: mutable int32 n = 0;
           param  iterations : 10u;
           output Input     : count = n++;

    }
    stream<int32 count, rstring name> Output = Test(Input) {
    param num : 3;
    }

    () as Writer = FileSink(Output) {
      param  file : "result.csv";
             flush  : 1u;
    }
}


// ***************************************************************
```

2) Edit the "Test.xml" file to include the parameter name. A snapshot of xml file is which needs to be edited:

```
  <parameters>
      <parameter>
        <name>num</name>
        <description>brief description of the parameter</description>
        <optional>true</optional>
        <type>int32</type>
        <cardinality>1</cardinality>
      </parameter>
    </parameters>
```

3) Java primitive operator

```
//***********************************************************

import com.ibm.streams.operator.AbstractOperator;
import com.ibm.streams.operator.OperatorContext;
import com.ibm.streams.operator.OutputTuple;
import com.ibm.streams.operator.StreamingInput;
import com.ibm.streams.operator.Tuple;
import com.ibm.streams.operator.StreamingOutput;

public class Test extends AbstractOperator {

private int i;
private int num;

  @Override
  public synchronized void initialize(OperatorContext context) throws Exception {
    super.initialize(context);
    i = 0;
    num = Integer.valueOf(context.getParameterValues("num").get(0));
  }

  public void process(StreamingInput stream, Tuple tuple) throws Exception {
      StreamingOutput<OutputTuple> output = getOutput(0);
      OutputTuple outputTuple = output.newTuple();

      // Submit any tuple with count  greater than "num"

      if( tuple.getInt("count") > num )
      {
          outputTuple.setInt("count", tuple.getInt("count"));
          outputTuple.setString("name", "a"+i);
          output.submit(outputTuple);
          i++;
      }
  }
}
```

4) The application is compiled just as in above examples.

//***

Example 4:
You can refer to the "SPL-Example-For-Beginners" and refer to the primitive java operator example "033_java_primitive_operator_at_work". However, the primitive java operator example does not have the java code and you will have to refer to another folder "RSS_Reader_Primitive" to refer to the Java code.
(Please note that "033_java_primitive_operator_at_work" example is the one you need to refer and not "Java_op_at_work". JavaOp is different and only permits a callout to another java operator)

Necessary Reading
"IBM SPL Toolkit Development Reference" and "IBM SPL Operator Model Reference" found in the Streas Information Center.