Implementing a Full Streaming WPS process

Following this tutorial you will be able to publish a Buffer process as a Full Streaming WPS process, so that it is possible to process continuous spatial data streams while obtaining intermediate results.

Prerequisites

- To follow this tutorial you need a 52°North WPS development environment. Have a look at A Primer on 52°North WPS in Eclipse Java "Indigo" or A Primer on 52°North WPS in Eclipse Java EE "Helios", depending on your preferred Eclipse version.
- Additionally, it is recommended to have a WPS client that supports streaming. As of now, the Quantum GIS WPS Client is the only one with this capability. Thus, if you want to visualize the result of the process in a neat way, consider installing such a client. Otherwise, you must read the response as XML.

Introduction

If you are not familiar with Streaming based WPS processes, please read this blog post to get an overview. A Full Streaming WPS process receives spatial data streams and meanwhile, starts processing them for publishing results. This enables your algorithms to process unbounded data streams receiving, processing, and publishing spatial data simultaneously.

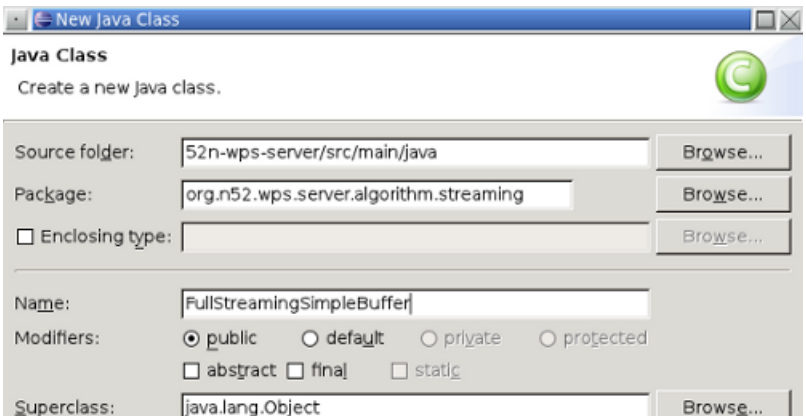
The Full Streaming Simple Buffer WPS will take a continuous data stream from an input `Playlist`, apply the buffer operation to available data, and, finally, send the result back to the client via an output `Playlist`. The following are the steps required to publish such a process.

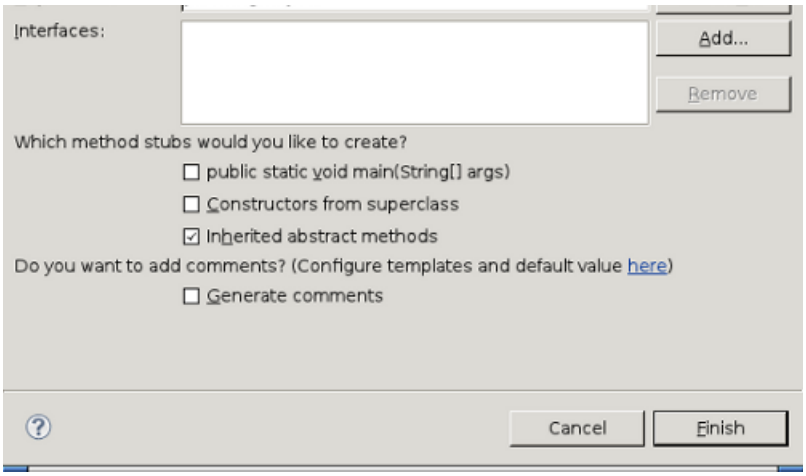
1. Create a base process

   Your Full Streaming WPS process will be based on a conventional WPS process. If necessary, create one following this tutorial. Fortunately, the Simple Buffer is already implemented in the 52°North WPS framework, so you can go to the step 2.

2. Creating a new class

   Create a new class in the `org.n52.wps.server.algorithm.streaming` package. You can call it `FullStreamingSimpleBuffer.java`





3. Extend the abstract class for Full Streaming

   Make your new class extend the abstract class called `AbstractVectorFullStreamingAlgorithm` and add the unimplemented methods, this way:

```
package org.n52.wps.server.algorithm.streaming;

public class FullStreamingSimpleBuffer extends AbstractVectorFullStreamingAlgorithm {

    @Override
    public String getBaseAlgorithmName() {
        // TODO Auto-generated method stub
        return null;
    }

    @Override
    public String getInputStreamableIdentifier() {
        // TODO Auto-generated method stub
        return null;
    }

    @Override
    public String getOutputIdentifier() {
        // TODO Auto-generated method stub
        return null;
    }

    @Override
    public int getTimeSlot() {
        // TODO Auto-generated method stub
        return 0;
    }

}
```

   You need to overwrite those four methods as indicated in the next steps.

4. Overwrite the `getBaseAlgorithmName` method

   Make the `getBaseAlgorithmName` method return the base process' full name. In this case, the full name of the Simple Buffer process is `org.n52.wps.server.algorithm.SimpleBufferAlgorithm`. So, just write

```
return "org.n52.wps.server.algorithm.SimpleBufferAlgorithm";
```

5. Overwrite the `getInputStreamableIdentifier` method

   Make the `getInputStreamableIdentifier` method return the name of the parameter that is a data stream, i.e., the one whose data will be available via a `Playlist`. For this, have a look at the `SimpleBufferAlgorithm` class and look for input identifiers.

   As you can see, there are two input identifiers: `data` and `width`. As Full Streaming WPS processes are intended to work with streams of spatial data, you must choose the identifier `data`. So, just write

   ```
   return "data";
   ```

   in the `getInputStreamableIdentifier` method of your `FullStreamingSimpleBuffer` class.

6. Overwrite the `getOutputIdentifier` method

   Make the `getOutputIdentifier` method return the output identifier of the base process. Proceed as in the previous step to get the output identifier of the `SimpleBufferAlgorithm` class.

   Yeap, it is `result`, so write this in the `getOutputIdentifier` method:

   ```
   return "result";
   ```

7. Overwrite the `getTimeSlot` method

   Make the `getTimeSlot` method return how often the input `Playlist` must be read in milliseconds. There is no strict rule for this parameter. Just as an example, if the input `Playlist` is updated every 5 seconds, you could set the time slot as 2000 (2 seconds). The smaller the time slot, the sooner the process will get available data; but also the more worthless requests will be triggered by the server. It is up to you.

   Write this in the `getTimeSlot` method:

   ```
   return 2000; // 2 seconds
   ```

   You are almost done, the name of the new process must be added to a 52°North WPS framework config file, see next step.

8. Add the process to the config file

Open the `wps_config.xml` file in `52n-wps-app/src/main/webapp/config/` and look for the `AlgorithmRepositoryList` tag in the middle of the file. You need to add a new `Property` element to the repository called `LocalAlgorithmRepository`. The new `Property` element is:

```
<Property name="Algorithm" active="true">org.n52.wps.server.algor
```

After adding it, the `wps_config.xml` file must look like this:

```
<Repository name="LocalAlgorithmRepository"
    className="org.n52.wps.server.LocalAlgorithmRepository" active="true">
    <Property name="Algorithm" active="true">org.n52.wps.server.algorithm.SimpleBufferAlgorithm</Property>
    <Property name="Algorithm" active="true">org.n52.wps.server.algorithm.coordinatetransform.CoordinateTransformAlgorithm</Property>
    <Property name="Algorithm" active="true">org.n52.wps.server.algorithm.simplify.DouglasPeuckerAlgorithm</Property>
    <Property name="Algorithm" active="true">org.n52.wps.server.algorithm.intersection.IntersectionAlgorithm</Property>
    <Property name="Algorithm" active="true">org.n52.wps.server.algorithm.convexhull.ConvexHullAlgorithm</Property>
    <Property name="Algorithm" active="true">org.n52.wps.server.algorithm.raster.AddRasterValues</Property>
    <Property name="Algorithm" active="true">org.n52.wps.server.algorithm.test.DummyTestClass</Property>
    <Property name="Algorithm" active="true">org.n52.wps.server.algorithm.spatialquery.IntersectsAlgorithm</Property>
    <Property name="Algorithm" active="true">org.n52.wps.server.algorithm.spatialquery.TouchesAlgorithm</Property>
    <Property name="Algorithm" active="true">org.n52.wps.server.algorithm.SnapPointsToLinesAlgorithm</Property>
    <Property name="Algorithm" active="true">org.n52.wps.server.algorithm.raster.ndwi</Property>
    <Property name="Algorithm" active="true">org.n52.wps.server.algorithm.streaming.OutputStreamingSimplifyDouglasPeucker</Property>
    <Property name="Algorithm" active="true">org.n52.wps.server.algorithm.streaming.OutputStreamingNDWI</Property>
    <Property name="Algorithm" active="true">org.n52.wps.server.algorithm.streaming.FullStreamingSnapPointsToLines</Property>
    <Property name="Algorithm" active="true">org.n52.wps.server.algorithm.streaming.FullStreamingSimpleBuffer</Property>
</Repository>
```

9. Compile and deploy

   Compile and deploy the application. A new process must appear once you call the `GetCapabilities` request.

Simple, isn't it?

For testing the Full Streaming WPS process you've just created, go to the 52°North WPS Test Client and copy the content of this document into the input box. Before sending it, you can choose whether you want to provide a static `Playlist` or a dynamic one. For the former, you are done; the URL of such a `Playlist` is already in the text you've just copied. For the latter, i.e., a dynamic `Playlist`, go to this page and copy the returned URL to the `href` attribute of the `data` input in the request, so that it looks like this:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<wps:Execute service="WPS" version="1.0.0" xmlns:wps="http://www.opengis.net/wps/1.0.0"
xmlns:ows="http://www.opengis.net/ows/1.1" xmlns:xlink="http://www.w3.org/1999/xlink"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.opengis.net/wps/1.0.0 http://schemas.opengis.net/wps/1.0.0
/wpsExecute_request.xsd">
<ows:Identifier>org.n52.wps.server.algorithm.streaming.FullStreamingSimpleBuffer</ows:Ident
  <wps:DataInputs>
    <wps:Input>
      <ows:Identifier>data</ows:Identifier>
        <ows:Title>data</ows:Title>
          <wps:Reference mimeType="application/x-ogc-playlist+text/xml"
schema="http://schemas.opengis.net/gml/2.1.2.1/feature.xsd"
xlink:href="http://geotux.pythonanywhere.com/playlists/playlist1346840098805.txt"/>
    </wps:Input>
    <wps:Input>
      <ows:Identifier>width</ows:Identifier>
      <ows:Title>width</ows:Title>
      <wps:Data>
        <wps:LiteralData>400</wps:LiteralData>
      </wps:Data>
```

Once you send the request, you will get an XML response with the URL of the output `Playlist`, which will be updated by the server every time intermediate results are available.

As stated before, it is recommended to use the Quantum GIS WPS Client for visualizing incoming intermediate results.

You will notice that there is a new parameter compared to the base process. It is called

`MaxTimeIdle` and you can use it to set the maximum time (in milliseconds) the server has to wait for new data from the input `Playlist`. If that time is exceeded, an exception will be thrown and the process will end, so make sure you choose a proper value that allows the input `Playlist` to be updated.