Visual Introduction to WebService
with
Tivoli Directory Integrator

(Part 1)

Document Last Updated

July 4th 2007

From TDI Support Team
ibmdi@us.ibm.com

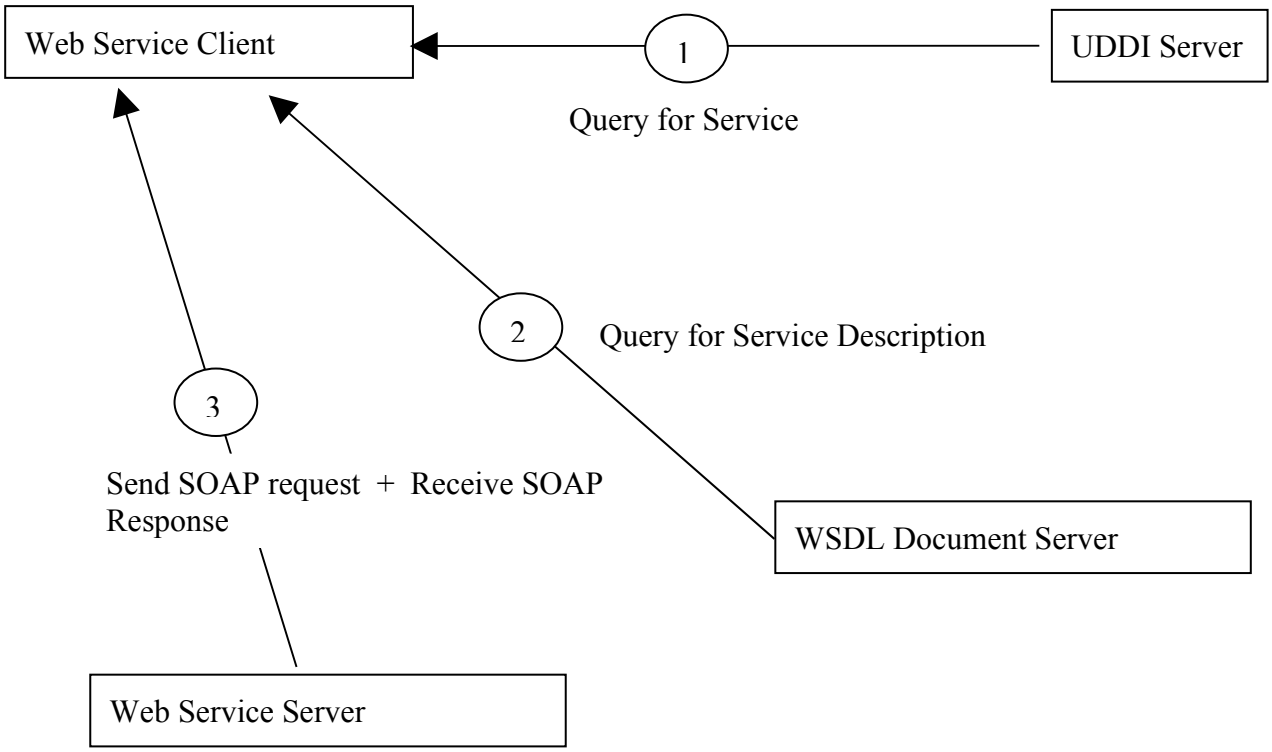
Visual Introduction to WebService
with
Tivoli Directory Integrator

Introduction to WebService

A Web Service (Server) is a software component that is described via WSDL and is capable of being accessed (from a web Service Client) via standard network protocols such as but not limited to SOAP over HTTP.

[adapted from http://www.oasis-open.org/committees/wsia/glossary/wsia-draft-glossary-03.htm#_Toc198211]

While a Web Service supports four types of patterns of communication between the Server and the Client (as defined in SOAP 1.2), we are going to consider only the Request-Response Message Exchange Pattern, which is depicted below.



As you can see from this diagram, UDDI and WSDL are really optional. If you know the location and the description of a Web service, you don't need to query the UDDI server and fetch the WSDL document. All you need to do is to build a request in SOAP format, and send it via HTTP to the provider.

Note: The Web Service Server is usually referred as Web Service Provider and Web Service Client as Web Service Consumer in standards literature. Here we will use Server and Client instead to make the tutorial compatible with naming conventions used by TDI Components.

Pre-requisites :

1. TDI version 6.1.1

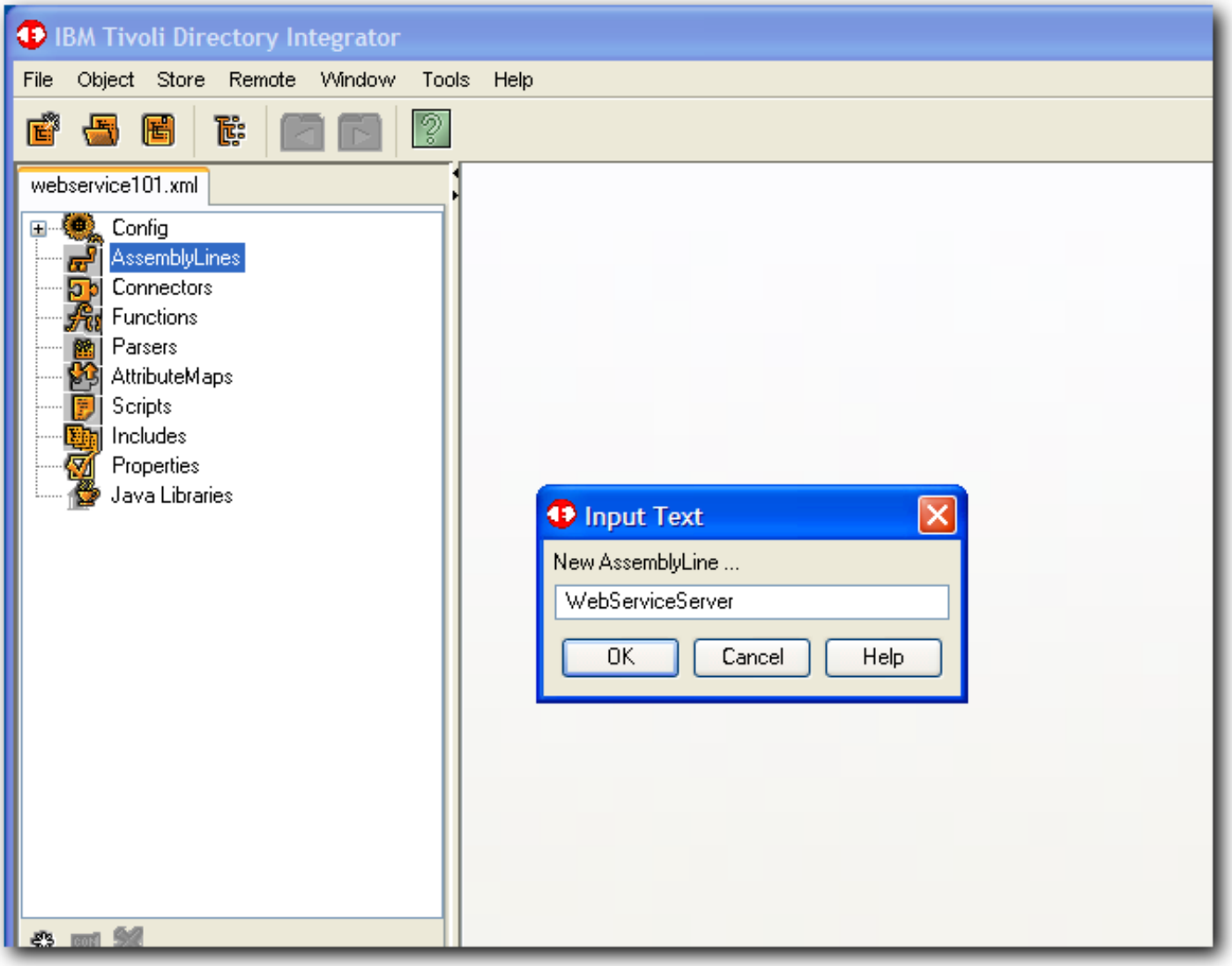
Additional Considerations:

Ensure that the port used in the tutorial for the Web Service Server is available for use.
Screen shots show Windows directory naming conventions, so please modify to suit your OS.

The Successful creation of a Web Service Server and Client requires successful completion of the following tasks (which are discussed in detail).

Section #1 – Creating a Web Service Server (provider)

Step#1 – Create an AL which will contain a WebService Server component



Step#2 – Create WebService Server Component

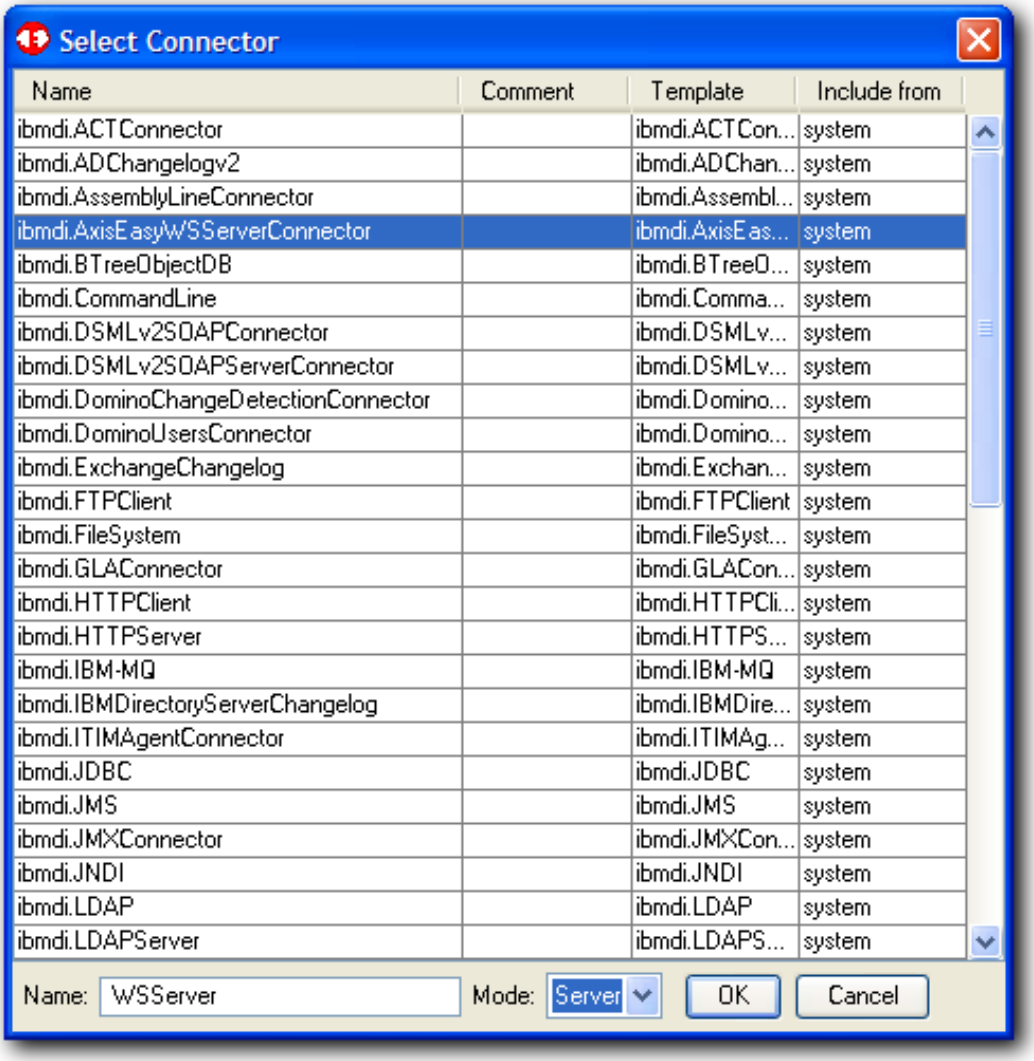
This example will use an AxisEasyWSServerConnector to provide the core WebService Server functionality.

Advantages of Connector :

The Axis Easy Web Service Server Connector is part of the TDI Web Services suite. It is a simplified version of the Web Service Receiver Server Connector in that it internally instantiates, configures and uses the AxisSoapToJava and AxisJavaToSoap FCs.

Create a connector which embeds a Webserver: AxisEasyWSServerConnector

Name: WSServer



The functionality provided is the same as if you chain and configure these FCs in an AssemblyLine which hosts the Web Service Receiver Server Connector. When using this Connector you forgo the possibility of hooking custom processing before parsing the SOAP request and after serializing the SOAP response. That is, you are tied to the processing and binding provided by Axis, but you gain simplicity of setup and use.

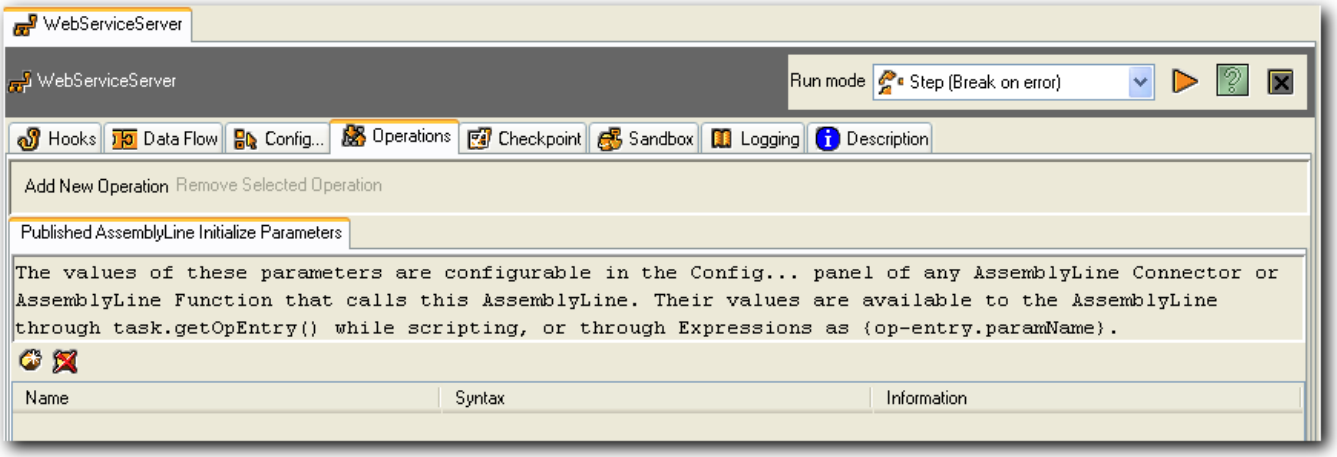
Another limitation of the *AxisEasyWSServerConnector* is that it can only be configured to handle one SOAP operation--it cannot service several SOAP operation requests. But apart from that, the customer is free to use whatever components he likes in the AssemblyLine that contains the Connector.

Limitations :
http://publib.boulder.ibm.com/infocenter/tivihelp/v2r1/topic/com.ibm.IBMDI.doc_6.1.1/referenceguide14.htm#axiswsrecvconn

http://publib.boulder.ibm.com/infocenter/tivihelp/v2r1/topic/com.ibm.IBMDI.doc_6.1.1/usersguide51.htm

Step#3 – Create & Expose WebService Functionality

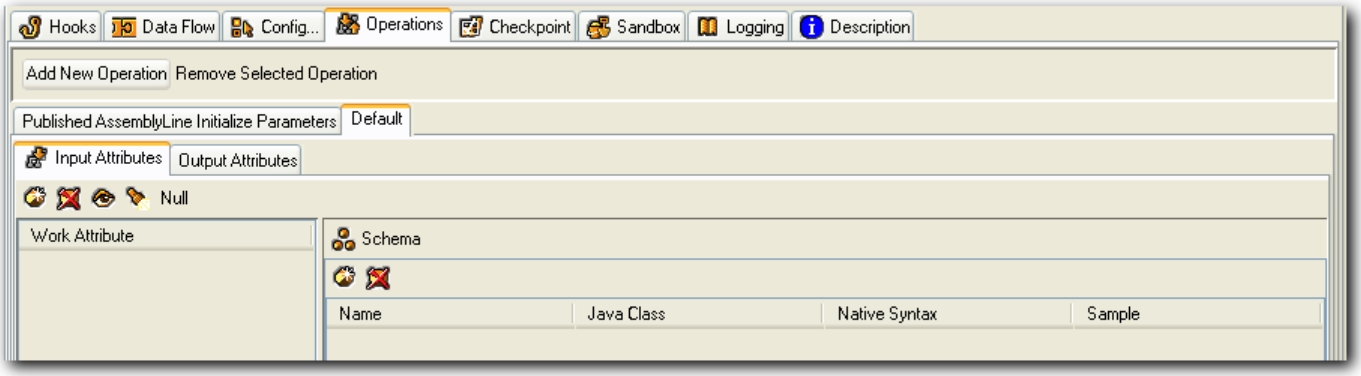
In TDI, AL operations are automatically exposed as the WebService functions and can be invoked remotely. In this example we will create 2 operations using the Operation Tab of the AL.



There are no Operations associated with the AL when the AL is created. Please NOTE that this behavior is different then AL's imported from TDIv6.0.

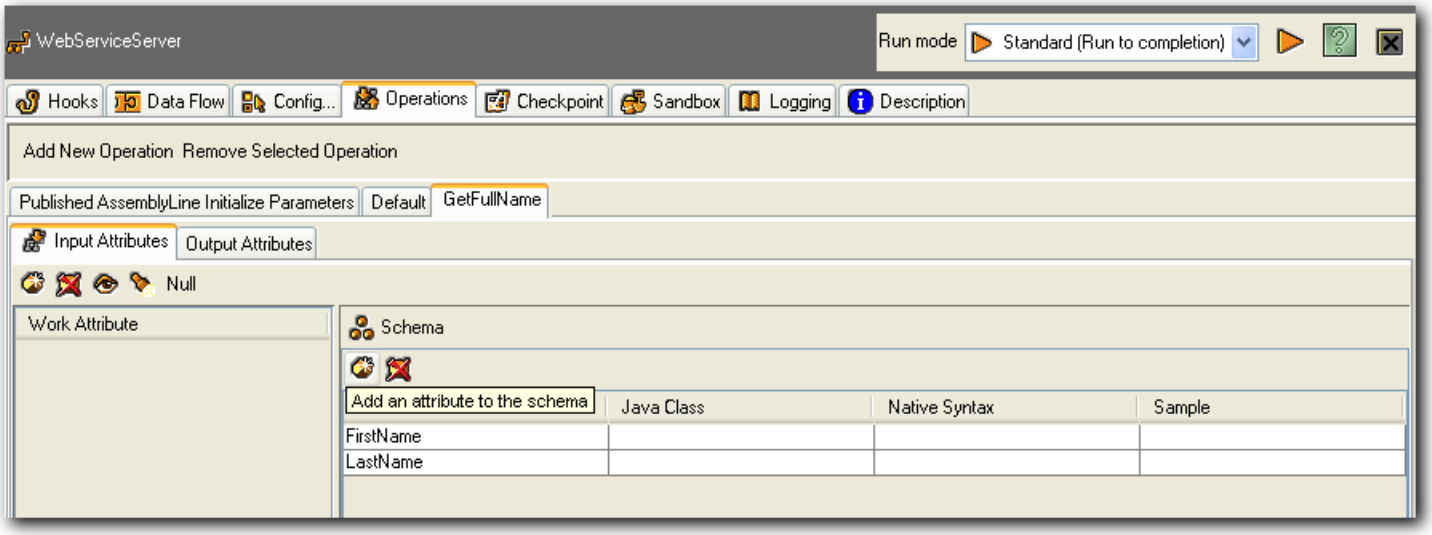
You need to create a "Default" operation for the AL to be able to run the AL.

And this particular operation will not have any Input or Output defined.

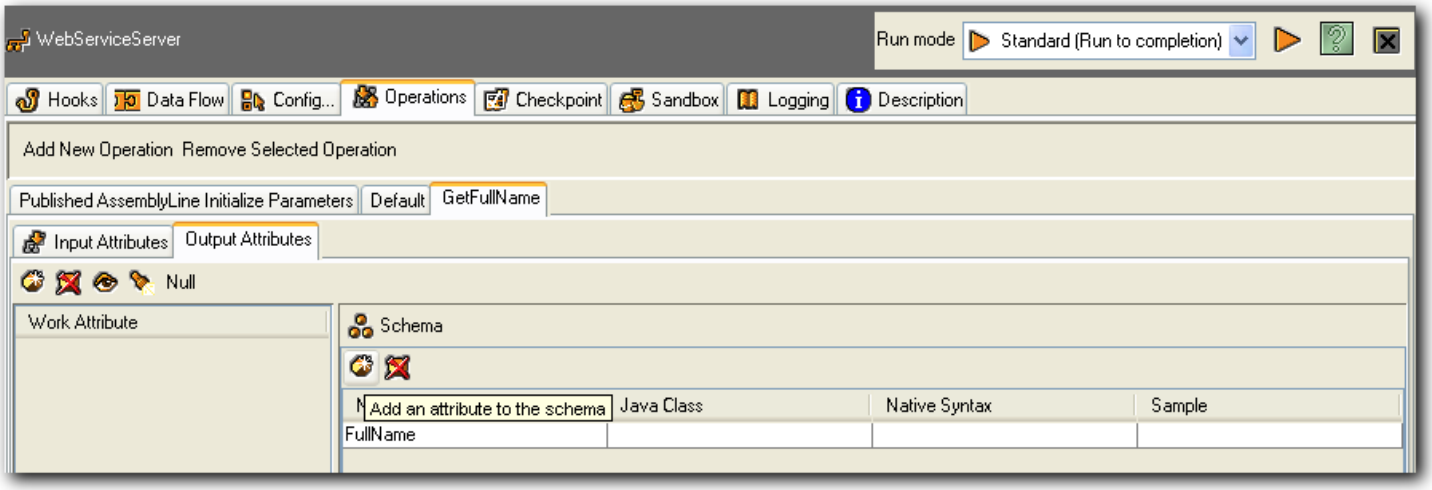


In addition to the Default Operation, we need to create an operation called "GetFullName".

Shown below are the two Input Attributes called “FirstName” & “LastName” which are the input parameters that are being passed in from the WebService Client into this WebService Server.

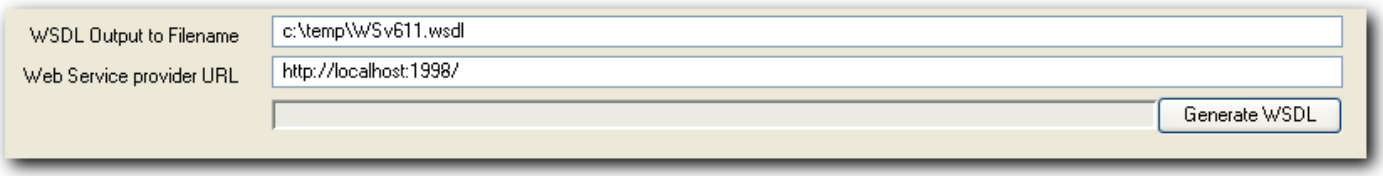


The next step in the process of defining the operation is to enumerate the list of attributes which will be passed back to the WebService Client. (This is defined in the “Output Attributes” Tab of the Operation).



Step#4 – Create the WSDL file

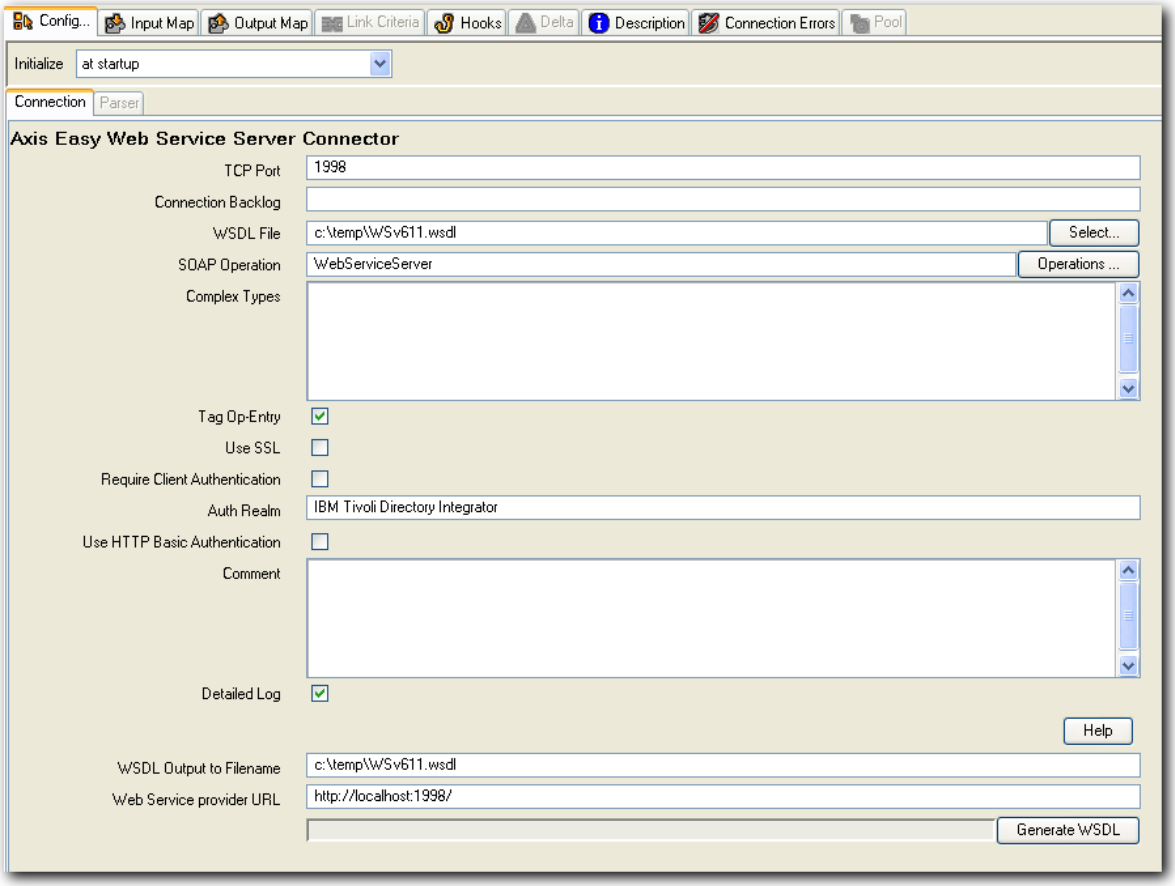
On the config Tab of the WebService Server Connector (AxisEasyWSServerConnector) enter the WSDL File name and the URL that the WebService Client will be communicating to. Example, if the WebService Server will be running on a machine called “tdihawk” and the port is 1998, you would define as http://tdihawk:1998/

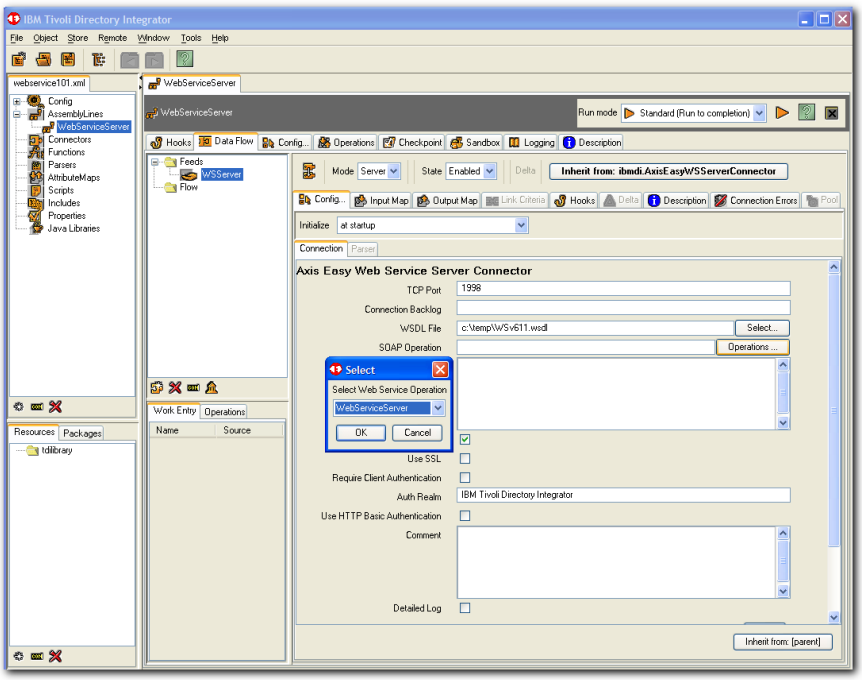


Ensure you press the “Generate WSDL”, and successfully generate the WSDL file.

Step#5 – Configure WebService Server parameters

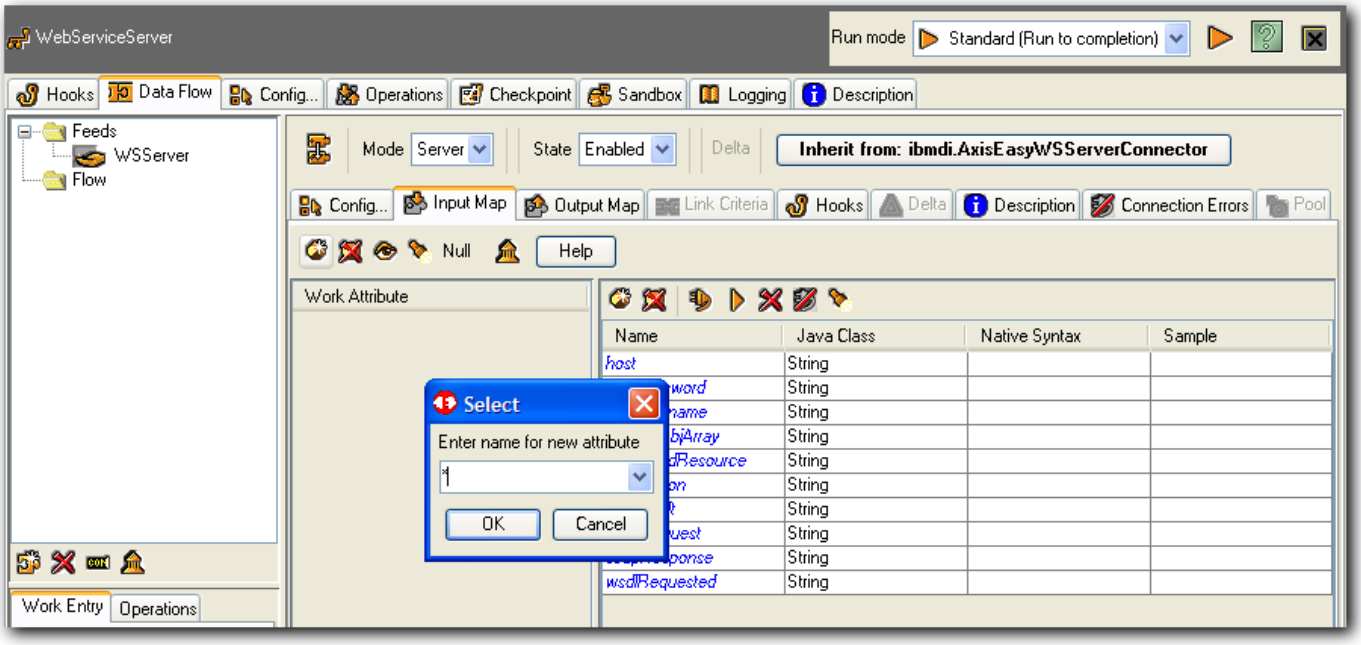
a. Configure Port
b. Enable “Tag Op-Entry”
c. Define the path to the WSDL file that you previously generated in Step4.
d. Choose a SOAP Operation



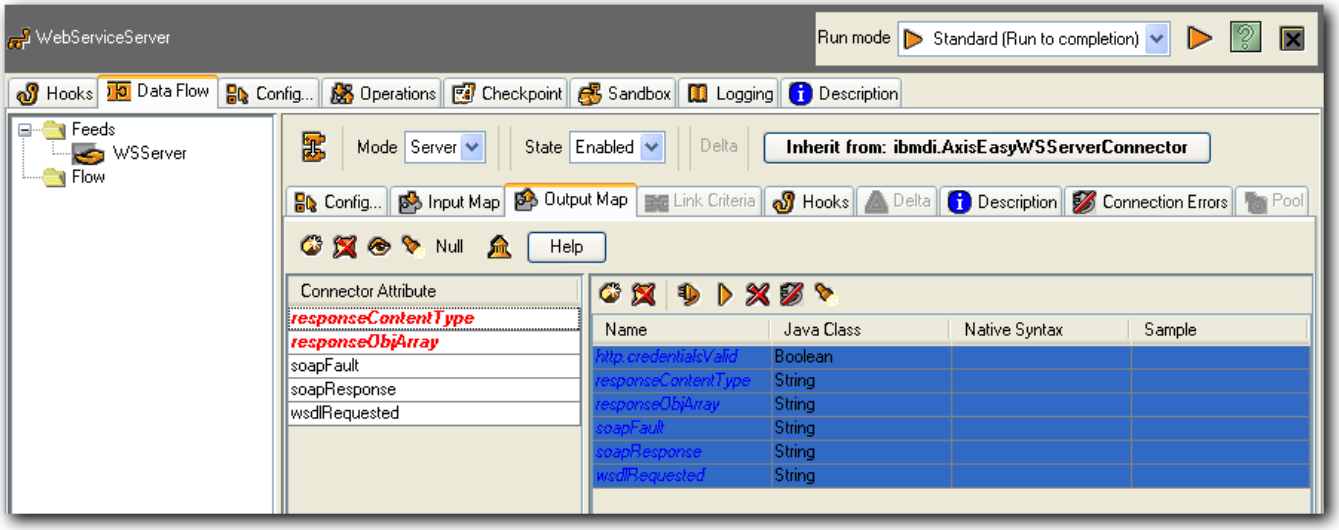


Step#6 – Define Input & Output Maps for WebService Server

Input Map: add "*"

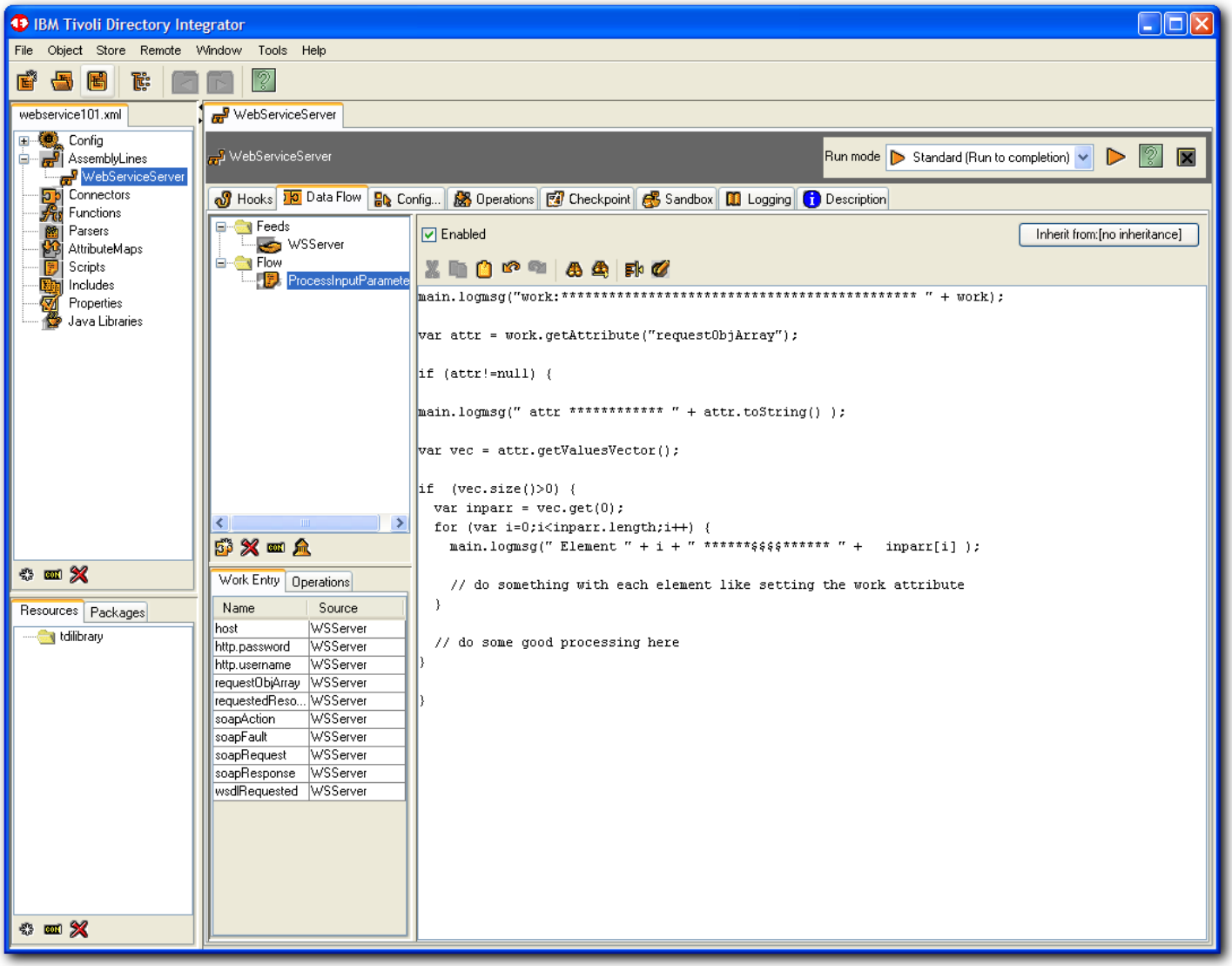


Output Map: The attributes responseContentType and responseObjArray are used to define the contents which will be sent back to the WebService Client. The additional three attributes, soapFault, soapResponse, and wsdlRequested, provide default responses needed for the WebService Server



Step#7 – Define WebService Server Logic Flow

Click on Flow, create Script Component.
Name: ProcessInputParameters



And Add the Following Script to the Script Component

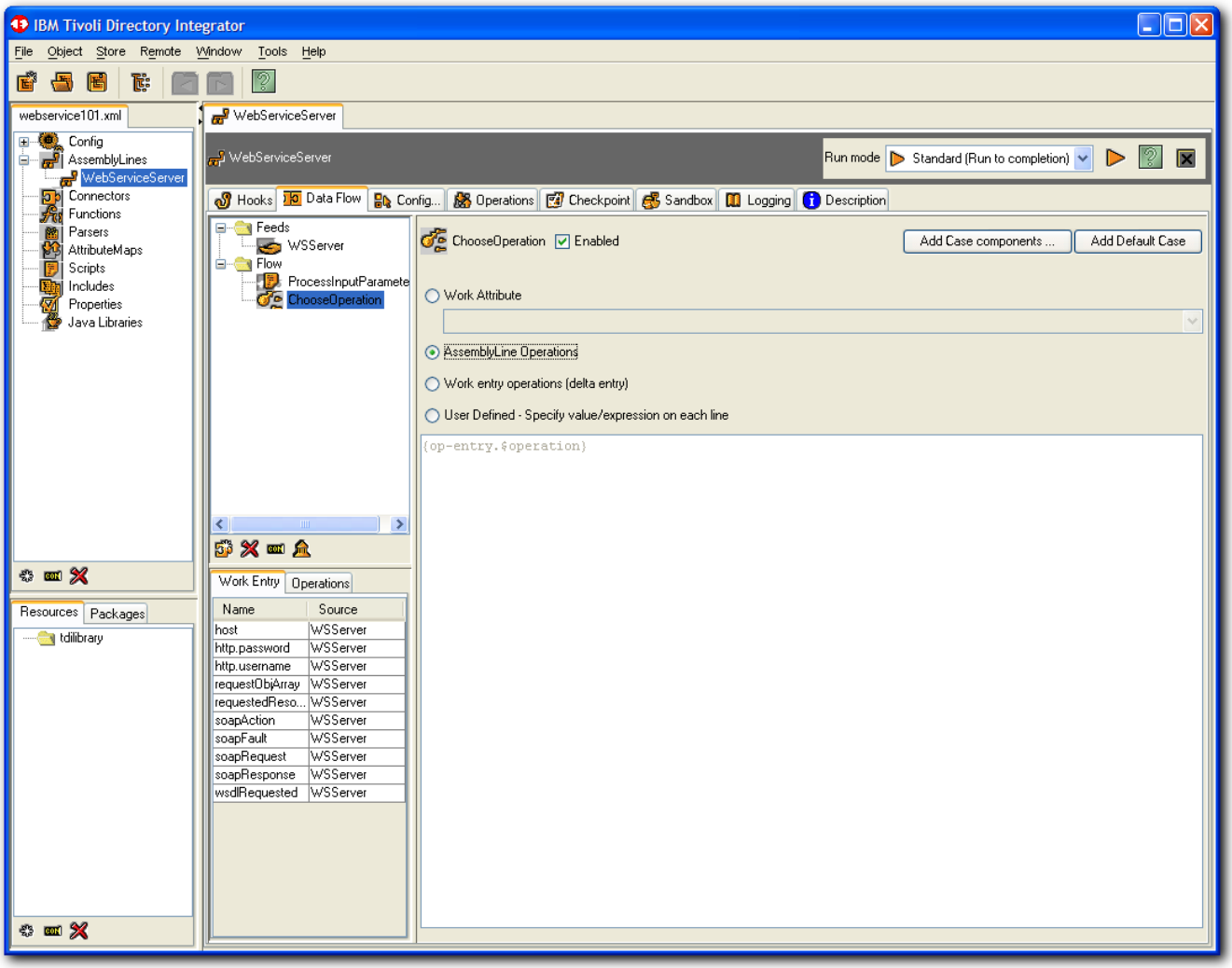
```
main.logmsg("work:********** " + work);
var attr = work.getAttribute("requestObjArray");
if (attr!=null) {
        main.logmsg(" attr ********** " + attr.toString() );
        var vec = attr.getValuesVector();
        if  (vec.size()>0) {
                var inparr = vec.get(0);
                for (var i=0;i<inparr.length;i++) {
                        main.logmsg(" Element " + i + " ******$$$$****** " +  inparr[i] );
                         // do something with each element like setting the work attribute
                }
                // do some good processing here
        }
}
```

Add a Switch Component, name: ChooseOperation

Choose “AssemblyLine Operations”

Choose “Add Case Components” - will choose the 2 operations created.

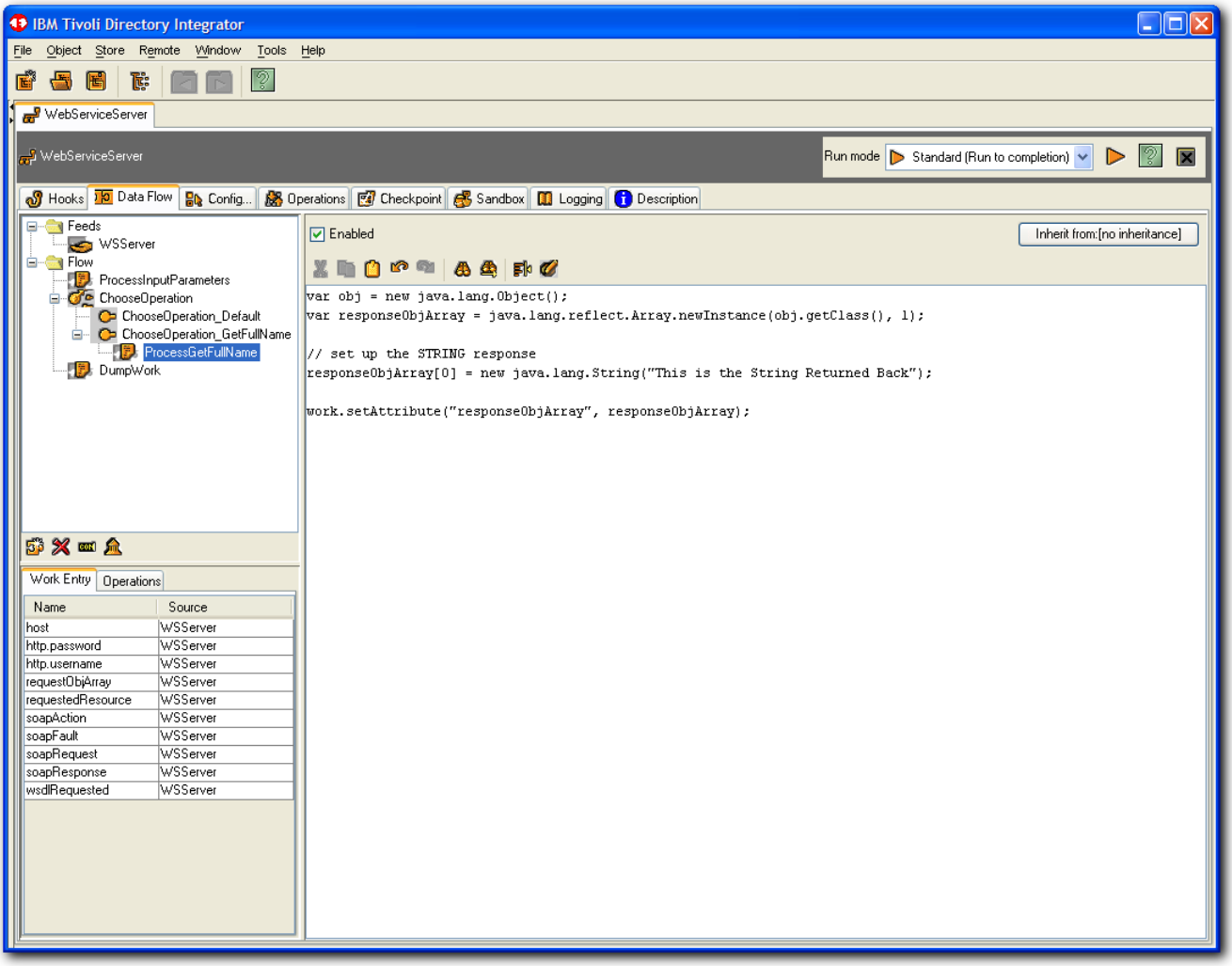
Note: if click again, get a message already added.



Note – You can add additional Case Components by clicking on “Add Case components” button during solution development.

In the ChooseOperation_GetFullname, click to add a script component.

Add script to setup the responseObjArray.



The Script that needs to be added to the response Object is ..

```
var obj = new java.lang.Object();
var responseObjArray = java.lang.reflect.Array.newInstance(obj.getClass(), 1);

// set up the STRING response
responseObjArray[0] = new java.lang.String("This is the String Returned Back");

work.setAttribute("responseObjArray", responseObjArray);
```

And Lastly…

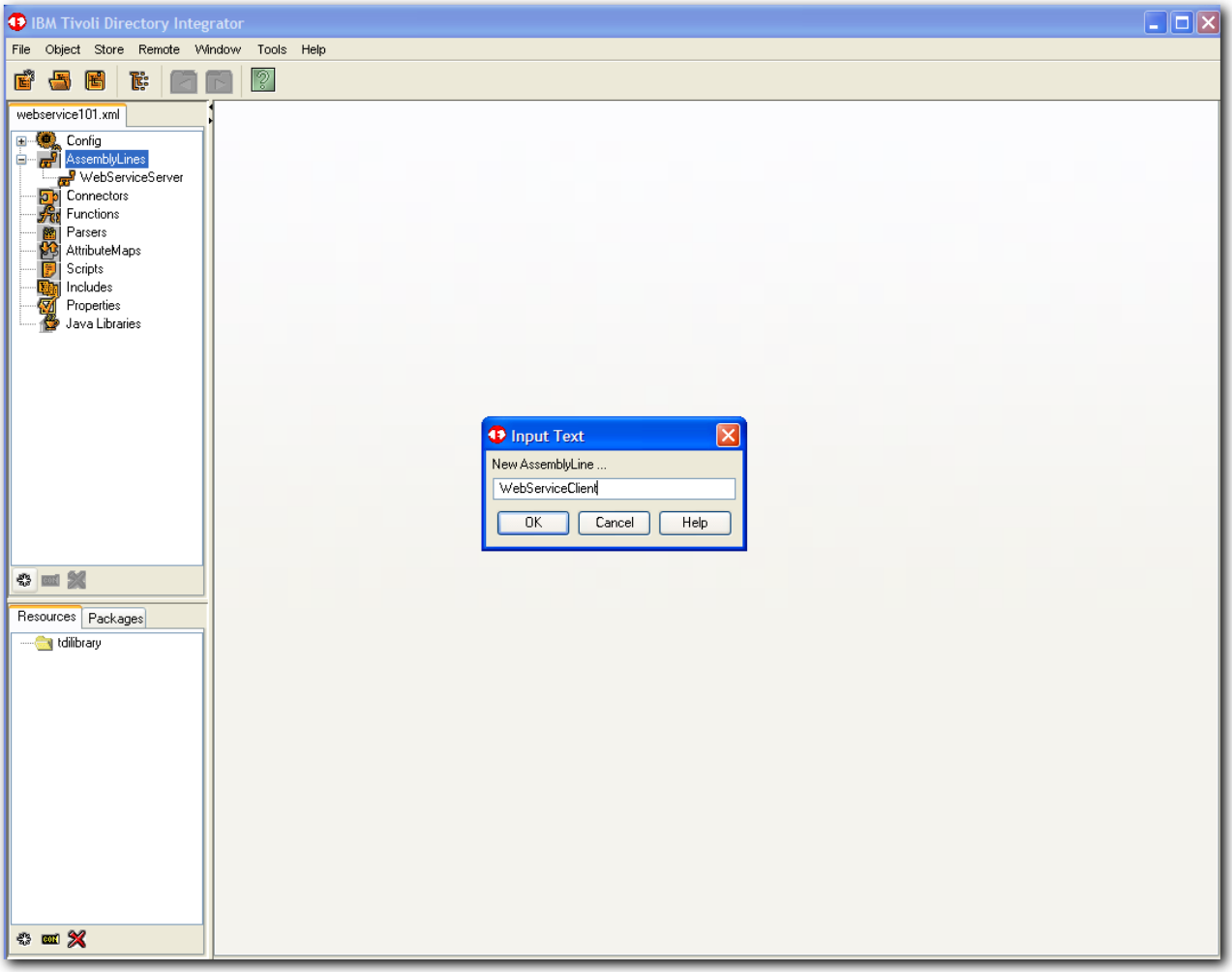
Add script component to end of flow, name: DumpWork

```
main.logmsg(“ *** “ + work);
```

Section #2 – Creating a Web Service Client (consumer)

Step#8 – Create an AL which will contain a WebService Client component

Add a new AL named: WebServiceClient



Step#9 – Create WebService Client Component

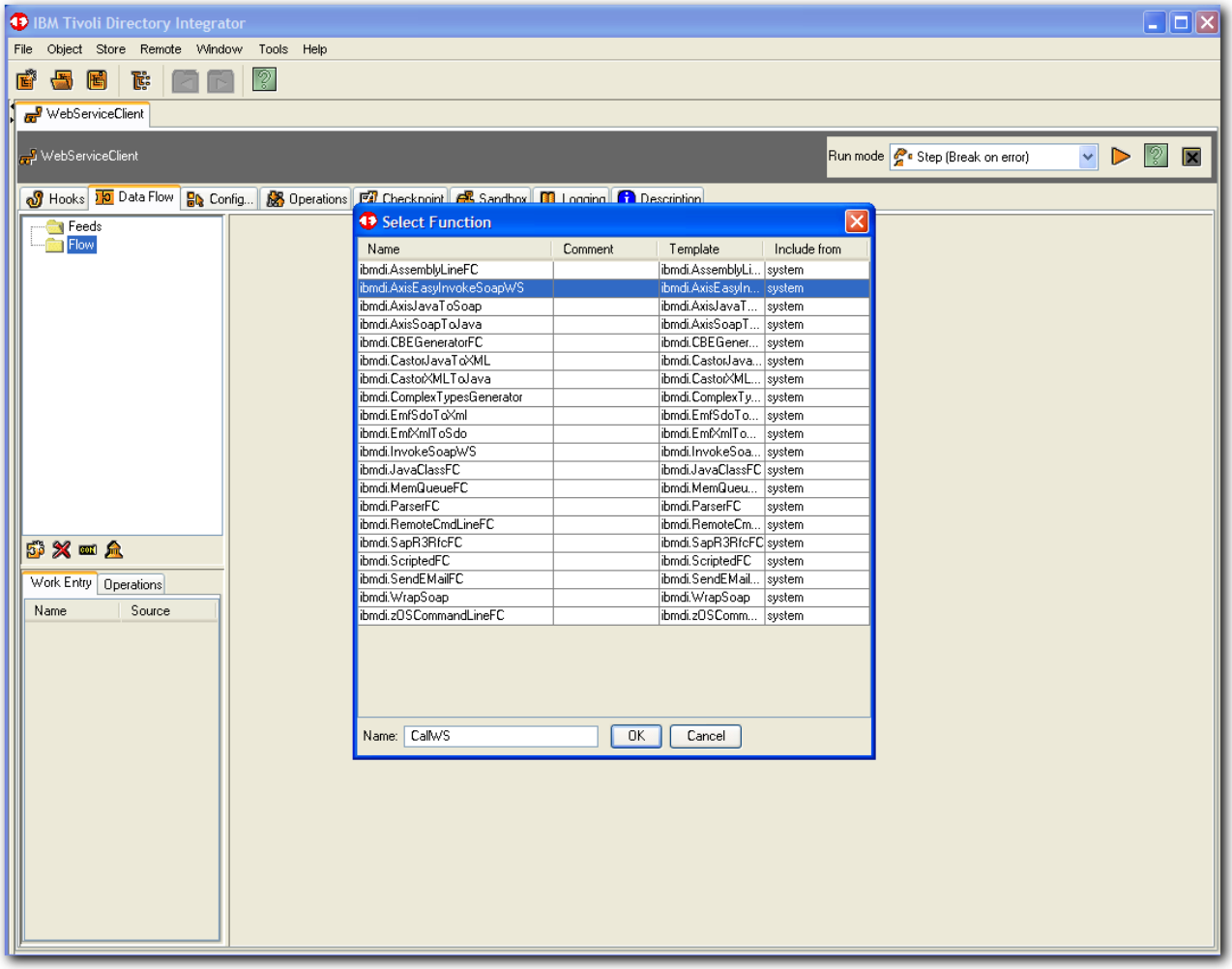
Add a Function Component of type AxisEasyInvokeSoapWS

The Axis EasyInvokeSoapWS Function Component (FC) is part of the TDI Web Services suite.

This is a "simplified" web service invocation component: it is a stand-alone FC with its own Config screen, but internally instantiates, configures and uses the following three FCs: AxisJavaToSoap, InvokeSoapWS and AxisSoapToJava.

The functionality provided is the same as if you chain and configure these three FCs in an AssemblyLine. When using this FC you lose the possibility of hook custom processing, that is, you are tied to the processing and binding provided by Axis. However, you gain simplicity of setup and use.

http://publib.boulder.ibm.com/infocenter/tivihelp/v2r1/topic/com.ibm.IBMDI.doc_6.1.1/referenceguide119.htm#axiseasyinvokesoapwsfc



This Function Component (FC) provides a relatively simple way of invoking SOAP over HTTP web services.

This is how the communication flows:

Web service client <-> AxisEasyInvokeSoapWS FC <-> org.apache.axis.client.Call <-> Web service

The Function Component takes either an Object Array or an Entry Type:

Object[] -> AxisEasyInvokeSoapWS FC -> Object[]

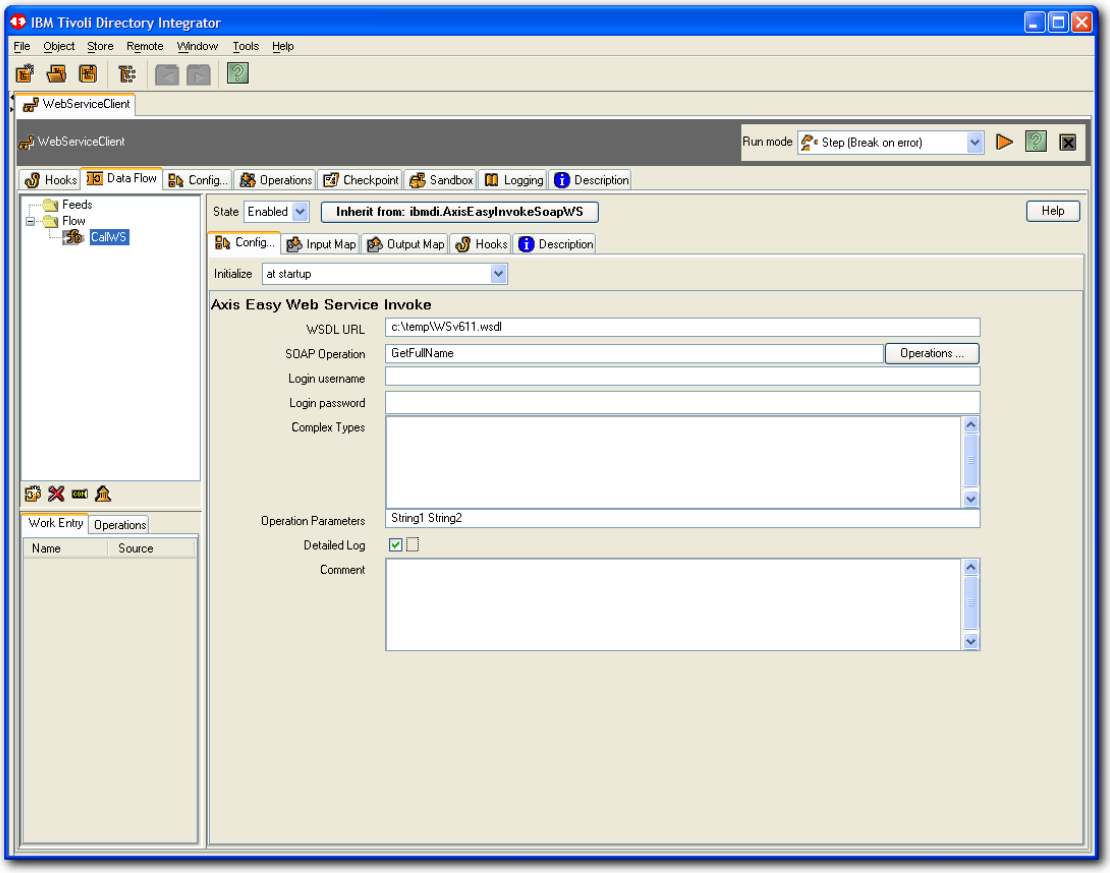
or

Entry -> AxisEasyInvokeSoapWS FC -> Entry

In our Example, we will use the Entry Type to define the Output Attributes of the Function Component. Note the Output of the Function Component is what is SENT to the Web Service Server, and the returned values come in the Input Map.

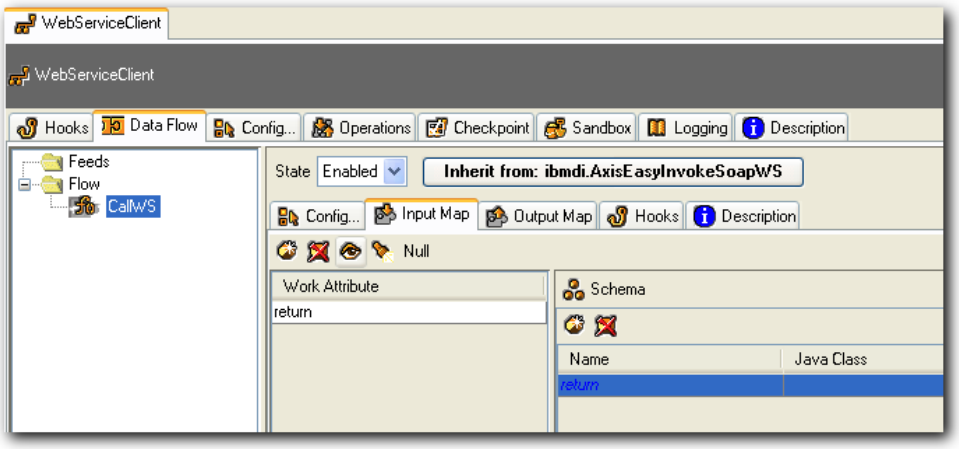
Step#10 – Configure WebService Client Component

Set the WSDL File, SOAP Operation, and the Operation Parameters (as shown in the example below)



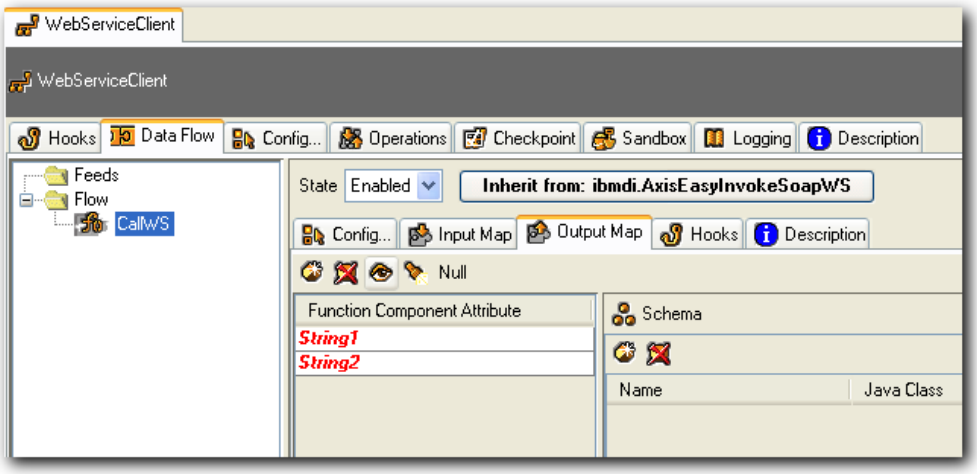
Define the Input Map

(Basically drag and drop the available Attribute called “return” on the Work side)



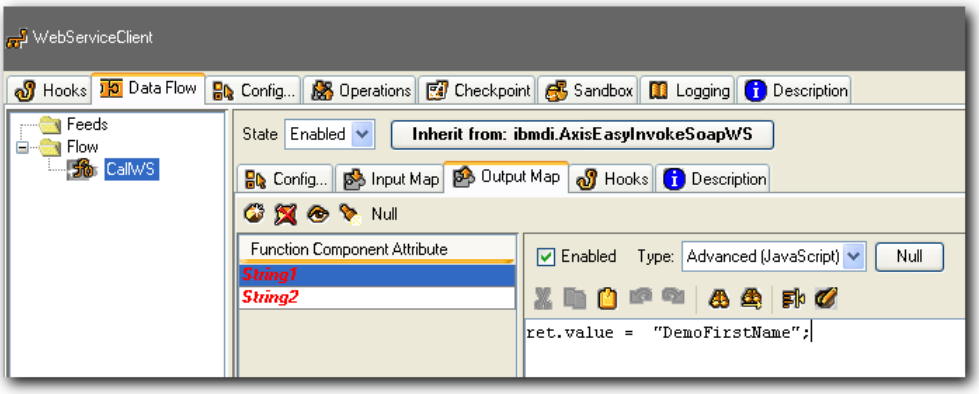
Define the Values for the Output Map

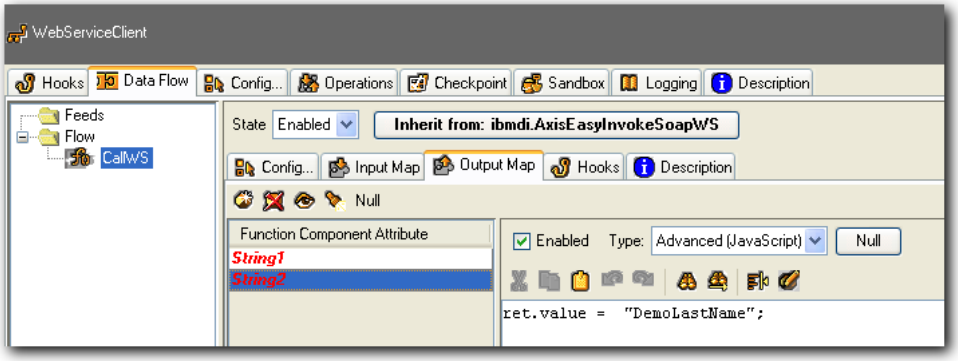
(The Output Map should have the two Attributes that are expected by the Web Service Server)



And set the Value for each the Output Map Attributes

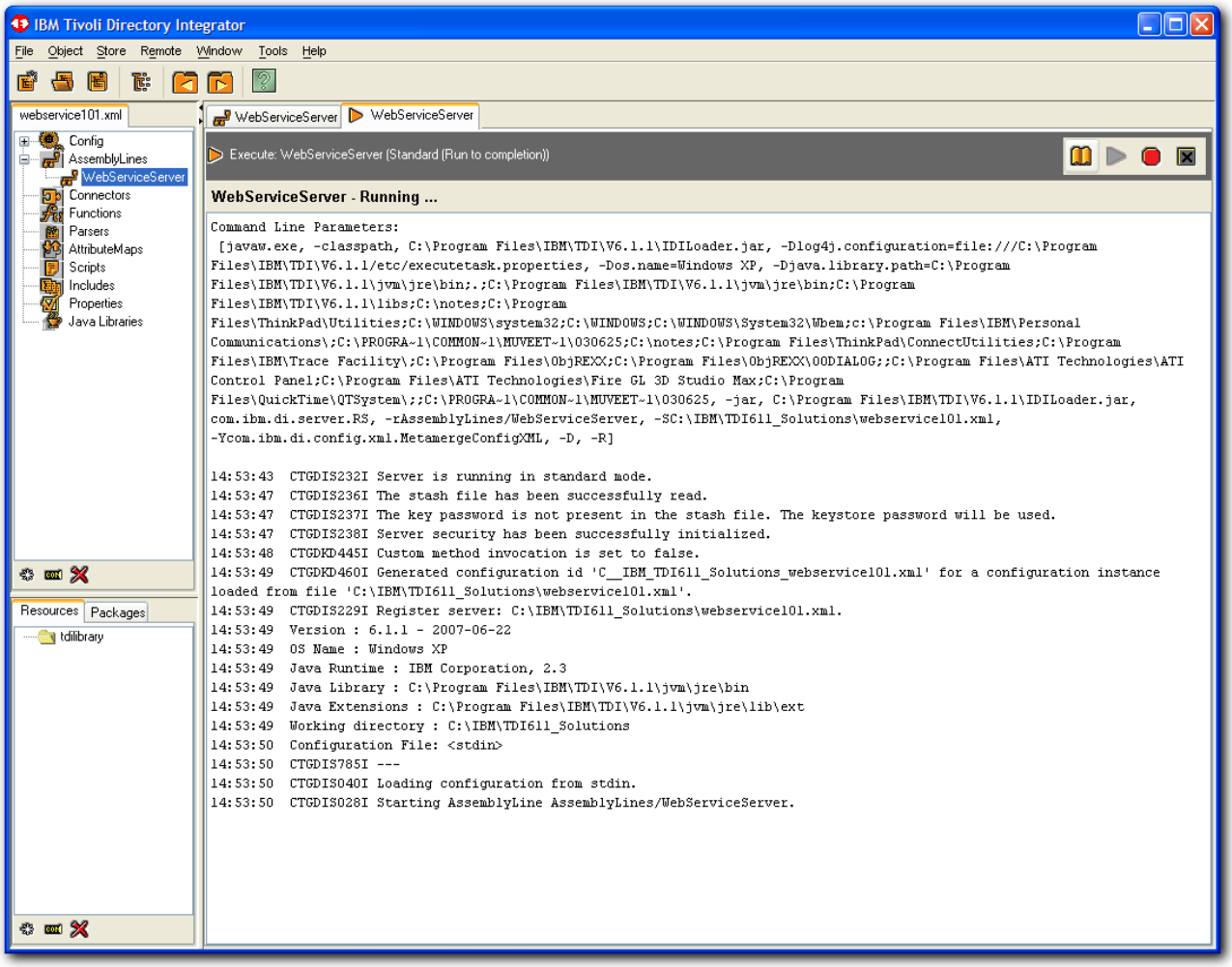
(The example shown below uses literal values)



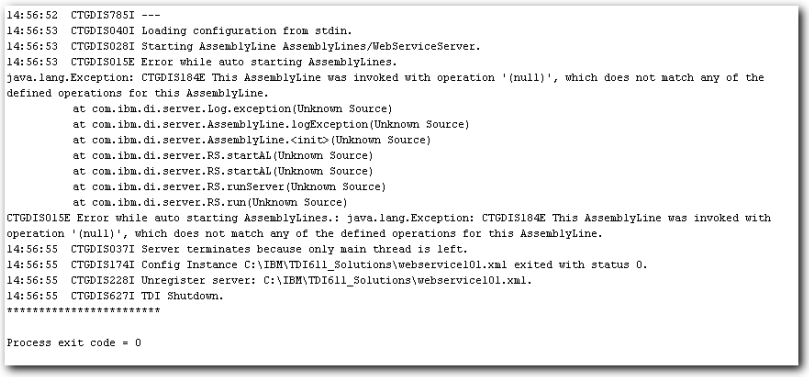


Section #3 – Test the Web Services Server and Client

Step#11 – Start the Web Services Server (Provider)

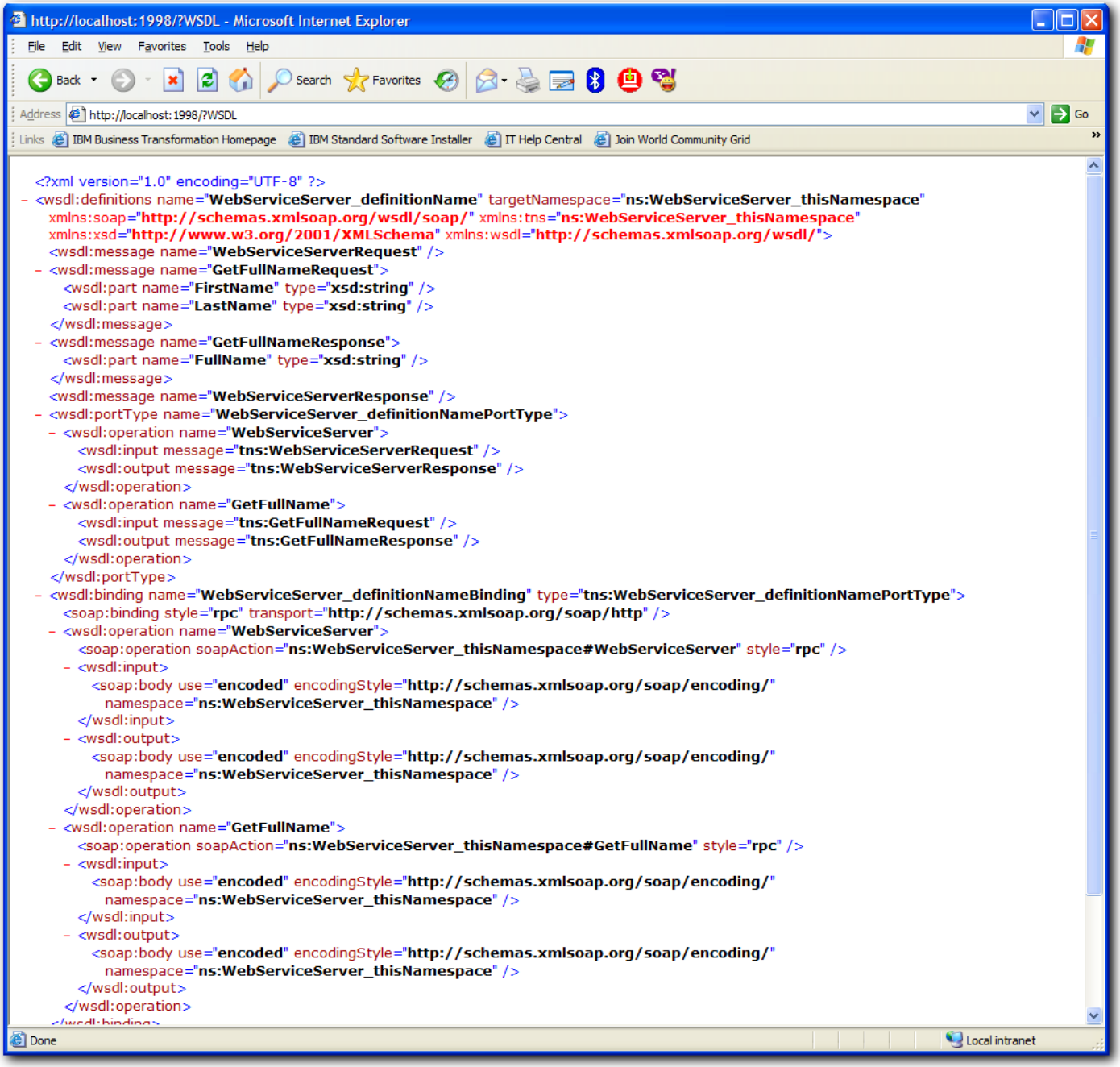


Please Note: If you get any Error in starting the Server Assembline Line, you need to fix this before proceeding further. If you get an error like the one shown below, ensure that you have an AssemblyLine Operation called Default (not “default”).



Step#12 – Test WSDL file from a Browser

Use a browser to go to http:\\webservice_server_url?WSDL (check the webservice_server_url defined in Section 1 of Step#4) In this example - Bring up a browser, with url: http://localhost:1998/?WSDL



Step#13 – Invoke the Web Services Client (to consume the Web Service)

It should provide a log like this..as it runs to completion..

```
15:46:36  [CallWS] CTGDIS504I *Result of attribute mapping*
15:46:36  [CallWS] CTGDIS505I The 'conn' object
15:46:36  [CallWS] CTGDIS003I *** Start dumping Entry
15:46:36  Operation: generic
15:46:36  Entry attributes:
15:46:36            String2 (replace):   'DemoLastName'
15:46:36            String1 (replace):   'DemoFirstName'
15:46:36  [CallWS] CTGDIS004I *** Finished dumping Entry
15:46:36  [CallWS] CTGDIS506I The 'work' object
15:46:36  [CallWS] CTGDIS003I *** Start dumping Entry
15:46:36  Operation: generic
15:46:36  Entry attributes:
15:46:36  [CallWS] CTGDIS004I *** Finished dumping Entry
15:46:36  [CallWS] CTGDIZ613I About to call web service...
15:46:38  [CallWS] CTGDIZ601I Web service called successfully.
15:46:38  [CallWS] CTGDIZ614I SOAP response Java Object: This is the String Returned Back.
15:46:38  [CallWS] CTGDIS504I *Result of attribute mapping*
15:46:38  [CallWS] CTGDIS505I The 'conn' object
15:46:38  [CallWS] CTGDIS003I *** Start dumping Entry
15:46:38  Operation: generic
15:46:38  Entry attributes:
15:46:38            return (replace):    'This is the String Returned Back'
15:46:38  [CallWS] CTGDIS004I *** Finished dumping Entry
15:46:38  [CallWS] CTGDIS506I The 'work' object
15:46:38  [CallWS] CTGDIS003I *** Start dumping Entry
15:46:38  Operation: generic
15:46:38  Entry attributes:
15:46:38            return (replace):    'This is the String Returned Back'
15:46:38  [CallWS] CTGDIS004I *** Finished dumping Entry
15:46:38   *** [return:This is the String Returned Back]
15:46:38  CTGDIS088I Finished iterating.
15:46:38  CTGDIS100I Printing the Connector statistics.
15:46:38   [CallWS] CallReply:1
15:46:38   [DumpWork] (No statistics for script component.)
15:46:38  CTGDIS104I Total: CallReply:1.
```

And on the Server Side,

```
15:46:38   [WSServer] Get:1, reply:1
15:46:38   [ProcessInputParameters] (No statistics for script component.)
15:46:38   [ChooseOperation] Switches:1
15:46:38   [ChooseOperation_Default] Match:0, NoMatch:1
15:46:38   [ChooseOperation_GetFullName] Match:1, NoMatch:0
15:46:38   [ProcessGetFullName] (No statistics for script component.)
15:46:38   [DumpWork] (No statistics for script component.)
15:46:38  CTGDIS104I Total: Get:1, reply:1, Switches:1.
15:46:38  CTGDIS101I Finished printing the Connector statistics.
15:46:38  CTGDIS080I Terminated successfully (0 errors).
15:46:38  CTGDIS079I AssemblyLine AssemblyLines/WebServiceServer terminated successfully.
```

Step#14 – Check the Server logs to see if the results are as expected

Enable Detailed Long on the STEP 5 – for the Web Services Server Connector, then the Server logs would be similar to this..

```
15:46:38  [WSServer] CTGDIS506I The 'work' object
15:46:38  [WSServer] CTGDIS003I *** Start dumping Entry
15:46:38   Operation: generic
15:46:38   Entry attributes:
15:46:38             soapAction (replace):          'ns:WebServiceServer_thisNamespace#GetFullName'
15:46:38             host (replace):      'localhost:1998'
15:46:38             requestedResource (replace):   ''
15:46:38             soapRequest (replace):         '<?xml version="1.0" encoding="UTF-8"?><soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"><soapenv:Body><nsl:GetFullName
soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" xmlns:nsl="ns:WebServiceServer_thisNamespace"><FirstName
xsi:type="xsd:string">DemoFirstName</FirstName><LastName
xsi:type="xsd:string">DemoLastName</LastName></nsl:GetFullName></soapenv:Body></soapenv:Envelope>'
15:46:38             requestObjArray (replace):     '[Ljava.lang.Object;@2040204'
15:46:38             responseObjArray (replace):    '[Ljava.lang.Object;@ea20ea2'
15:46:38             wsdlRequested (replace):       'false'
15:46:38  [WSServer] CTGDIS004I *** Finished dumping Entry
15:46:38  [WSServer] CTGDIZ008I Response Object values: [This is the String Returned Back].
15:46:38  [WSServer] CTGDIZ127I EventHandler response HTTP status code set to '200 OK'.
15:46:38  [WSServer] CTGDIZ128I EventHandler response content type set to 'text/xml;charset=UTF-8'.
15:46:38  [WSServer] CTGDIZ129I EventHandler response content length set to '510'.
15:46:38  [WSServer] CTGDIZ130I EventHandler response set to '<?xml version="1.0" encoding="UTF-8"?><soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <soapenv:Body>
  <nsl:GetFullNameResponse xmlns:nsl="ns:WebServiceServer_thisNamespace">
   <nsl:arg0 xsi:type="soapenc:string" xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/">This is the String Returned Back</nsl:arg0>
  </nsl:GetFullNameResponse>
 </soapenv:Body>
</soapenv:Envelope>'.
15:46:38  CTGDIS088I Finished iterating.
15:46:38  CTGDIS315I AssemblyLine worker thread: AssemblyLines/WebServiceServer.WSServer.751578316 is stopped.
15:46:38  CTGDIS100I Printing the Connector statistics.
15:46:38   [WSServer] Get:1, reply:1
```

The End.

For any question about this document, contact ibmdi@us.ibm.com

Useful TDI Links

On-line Documentation Fix Packs ITDI Wiki News Group

Remote Assistance Tool Internal Site Internal News Group