TIBCO ActiveMatrix BusinessWorks™

Samples

Version 6.10.0 | November 2023

Copyright © 2015-2023. Cloud Software Group, Inc. All Rights Reserved.

Contents

Contents **2**

Changing Help Preferences **8**

Accessing Samples **10**

Guided User Experience in TIBCO Business Studio for BusinessWorks **13**

Setting the Default Application Profile **16**

Creating a ActiveMatrix BusinessWorks Project from Template **18**

Unit Testing **21**

Maven Test Sample of Main Process 22

Maven Sample of Unit Test Demo Project 26

Maven Sample of Shared Module Unit Test Cases 31

Binding Features **34**

Using Basic REST Binding 34

Understanding the Configuration 36

Troubleshooting 38

Using REST to Manage Books for a Bookstore 38

Understanding the Configuration 44

Troubleshooting 48

Implementing a REST Service Using a JSON File with "allOf" Keyword 49

Implementing a REST Service Using a JSON File with "anyOf" Keyword 54

Implementing a REST Service Using a JSON File with "oneOf" Keyword 58

Implementing Multiple Response based REST Service using a JSON file 62

Using a SOAP Web Service to Make a Query to a Bookstore 67

Understanding the Configuration 68

Reconfiguring a Web Service by Changing the Transport 69
Converting the Transport from SOAP over JMS to SOAP over HTTP 71
Using a SOAP Connection to Publish Messages to Multiple Endpoints 72
Sending and Receiving a SOAP Bound Attachment 73
Understanding the Configuration 74
Interoperability of Using JAX-WS Client to Invoke Service 75
Understanding the Configuration 76
Invoking a SOAP over HTTP Web Service Using Different SOAP Versions 76
Understanding the Configuration 78
Invoking a SOAP over HTTP Web Service Using Different Binding Styles 79
Understanding the Configuration 81
Implementing a Service with a Service Binding and a Client with a Reference Binding 81
Understanding the Configuration 83
Implementing a Mortgage Loan Application with Fault 88
Understanding the Configuration 90
Sending and Receiving a SOAP Unbound Attachment 91
Understanding the Configuration 92
Managing a Purchase Order 93
Understanding the Configuration 94
Using SOAP Bound Headers 95
Understanding the Configuration 96
Using SOAP UnBound Headers 97
Understanding the Configuration 97
Using SOAP HTTP Transport Headers 98
Understanding the Configuration 100
Sending a SOAP Request and Receiving a SOAP Reply 100
Understanding the Configuration 102
Implementing a HTTP Service That Returns Information Based on ZIP Codes 103
Understanding the Configuration 104
Implementing a JMS Service That Writes Attachment to File 104

Understanding the Configuration ... 106
Sending a One Way Message over JMS ... 106
Understanding the Configuration ... 107
Using JMS Destination Bridges ... 108
Understanding the Configuration ... 109
Implementing a JMS Service that Returns Information Based on Zip Codes ... 109
Understanding the Configuration ... 110
Implementing a JMS Service That Returns an Attachment ... 111

Core Features ... 113

Using Process and Module Properties ... 113
Using a Process Conversation to Correlate JMS Message Jobs ... 114
Understanding the Configuration ... 116
Using Modularity with the Java Palette ... 116
Understanding the Configuration ... 118
Process ... 119
Troubleshooting ... 121
Configuring a Conversation to Join a Wait for File Change Activity ... 122
Understanding the Configuration ... 123
Using an Asynchronous Event Handler to Implement a Simple Mortgage Workflow ... 123
Understanding the Configuration ... 125
Using a Cross Process Conversation to Determine the Mortgage Rate ... 125
Understanding the Configuration ... 127
Catching Activity Faults with Scope and Fault Handlers ... 127
Modularity Among Applications with a Shared Module Project ... 129
Understanding the Configuration ... 131
Load Balancing an Application Using an AppSpace ... 131
Invoking Subprocesses ... 134
Understanding the Configuration ... 135
Dynamically Invoking Subprocesses Using SetEPR Activity ... 136
Troubleshooting ... 139

Calling a Direct Subprocess 139
Understanding the Configuration 140
Collecting Process, Activity, and Transition Statistics 140

Policy Feature 149
Enforcing Basic Authentication with LDAP Authentication 149
Understanding the Configuration 151
Enforcing Basic Authentication on a REST Service Binding 151
Understanding the Configuration 153
Enforcing Basic Credential Mapping on an Outbound Request 154
Understanding the Configuration 155
Enforcing SOAP Security to Enable Confidentiality and Integrity on Message Exchanges 155
Understanding the Configuration 157
Enforcing SOAP Security to Enable SAML Authentication and SAML Credential Mapping 157
Understanding the Configuration 158
Exposing Security Context 159
Understanding the Configuration 160

Palettes Features 163
Creating an Order Processing Service 163
Understanding the Configuration 165
Using FTP to Put and Get Files 165
Understanding the Configuration 167
Using HTTP to Send and Receive Dynamic Headers 168
Understanding the Configuration 170
HTTP Sending and Receiving MIMEAttachments 170
Understanding the Configuration 171
Using HTTP or SOAP to Get Information from MySQL 172
Understanding the Configuration 173
Using the HTTP Persistent Connection Feature 173

Sending a Request and Getting a Response from a Website 175
Sending a Secured Request and Getting a Secured Response from a Web Site 177
Understanding the Configuration 178
Invoking Custom Code to Complete a Purchase Order 179
Understanding the Configuration 181
Troubleshooting 181
Using Java Activities to Publish Information 182
Using Java Invoke that Makes a REST call to Search Twitter 184
Understanding the Configuration 185
Troubleshooting 186
Using a Custom Process Starter in a Project That Gets Information about a Checking Account 187
Understanding the Configuration 188
Troubleshooting 188
Using JDBC Query - Process in Subsets and JDBC Update - Batch Updates 189
Understanding the Configuration 191
Setting Up a JDBC Connection to Query and Update Tables 192
Understanding the Configuration 195
Using Oracle Objects and Collections in JDBC Call Procedure and JDBC Query Activities 195
Understanding the Configuration 197
Setting Up and Executing JDBC Activities with Eclipse Using DTP 197
Understanding the Configuration 201
Sending and Getting JMS Messages 202
Understanding the Configuration 203
Using JMS Message Selector 204
Understanding the Configuration 205
Receiving a JMS Message from a Queue and Responding to the Message 205
Understanding the Configuration 207
Sending a JMS Message and Receiving a Response in a Process 207
Sending a JMS Message to a Queue and Waiting for a Response 209

Understanding the Configuration ... 210
Sending a JMS Message and Receiving a Durable Response ... 211
Understanding the Configuration ... 212
Sending Mail Using HTML ... 212
Sending Mail Using MIME ... 213
Sending and Receiving Mail ... 217
Troubleshooting ... 219
Publishing and Subscribing to TIBCO Rendezvous Messages ... 219
Understanding the Configuration ... 221
Managing TCP Connections ... 221
Understanding the Configuration ... 223
Sharing a TCP Connection across Multiple Processes ... 224
Understanding the Configuration ... 225
Inter Process Communication using Wait for Notification and Notify Activities ... 226
Understanding the Configuration ... 227

Migrating Projects ... 228
HTTP Basic Sample ... 229
HTTP Persistent Connection ... 231
JAVA Method Project ... 232
JMS Message Selector Project ... 232
HTML Mail with Image ... 234
Mail with Simple Attachment ... 234

TIBCO Documentation and Support Services ... 236

Legal and Third-Party Notices ... 238

Changing Help Preferences

By default, documentation access from TIBCO Business Studio™ for BusinessWorks™ is online, through the TIBCO Product Documentation website that contains the latest version of the documentation. Check the website frequently for updates. To access the product documentation offline, download the documentation to a local directory or an internal web server and then change the help preferences in TIBCO Business Studio for BusinessWorks.

Before you begin

Before changing the help preferences to access documentation locally or from an internal web server, download the documentation.

1. Go to https://docs.tibco.com/
2. In the **Search** field, enter TIBCO ActiveMatrix BusinessWorks™ and press **Enter**.
3. Select the TIBCO ActiveMatrix BusinessWorks™ product from the list. This opens the product documentation page for the latest version.
4. Click **Download All**.
5. A compressed .zip file containing the latest documentation is downloaded to your web browser's default download location.
6. Copy the .zip file to a local directory or to an internal web server and unzip the file.

To point to a custom location:

Procedure

1. Perform one of the following steps in TIBCO Business Studio for BusinessWorks based on your operating system:
   - On Windows OS: Click **Window > Preferences**
   - On macOS: Click **TIBCO Business Studio > Preferences**.
2. In the Preferences dialog, click **BusinessWorks > Help**.
3. Click **Custom Location**, and then browse to the `html` directory in the folder where you extracted the documentation or provide the URL to the `html` directory on your

internal web server.

4. Click **Apply**, and then click **OK**.

Accessing Samples

You can access the samples in TIBCO Business Studio™ for BusinessWorks™ from the **File Explorer** view.

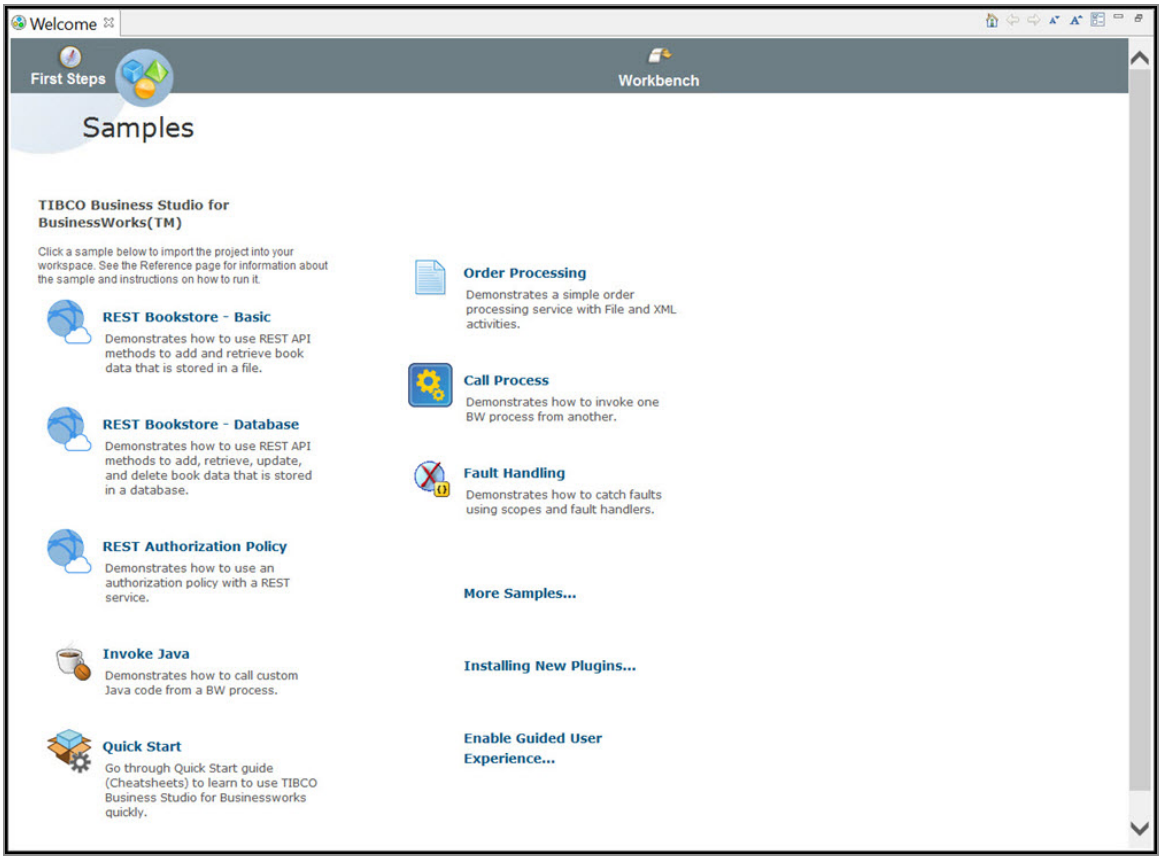
Procedure

1. Open TIBCO Business Studio™ for BusinessWorks™ from `$TIBCO_HOME\studio\<version>\eclipse\TIBCOBusinessStudio.exe`.

   By default, TIBCO Business Studio™ for BusinessWorks™ opens the Design perspective.

2. Click **Help > BusinessWorks Samples**.

   The **Welcome** screen is displayed with various available samples, for example, REST Bookstore - Basic, REST Bookstore - Database, Order Processing.

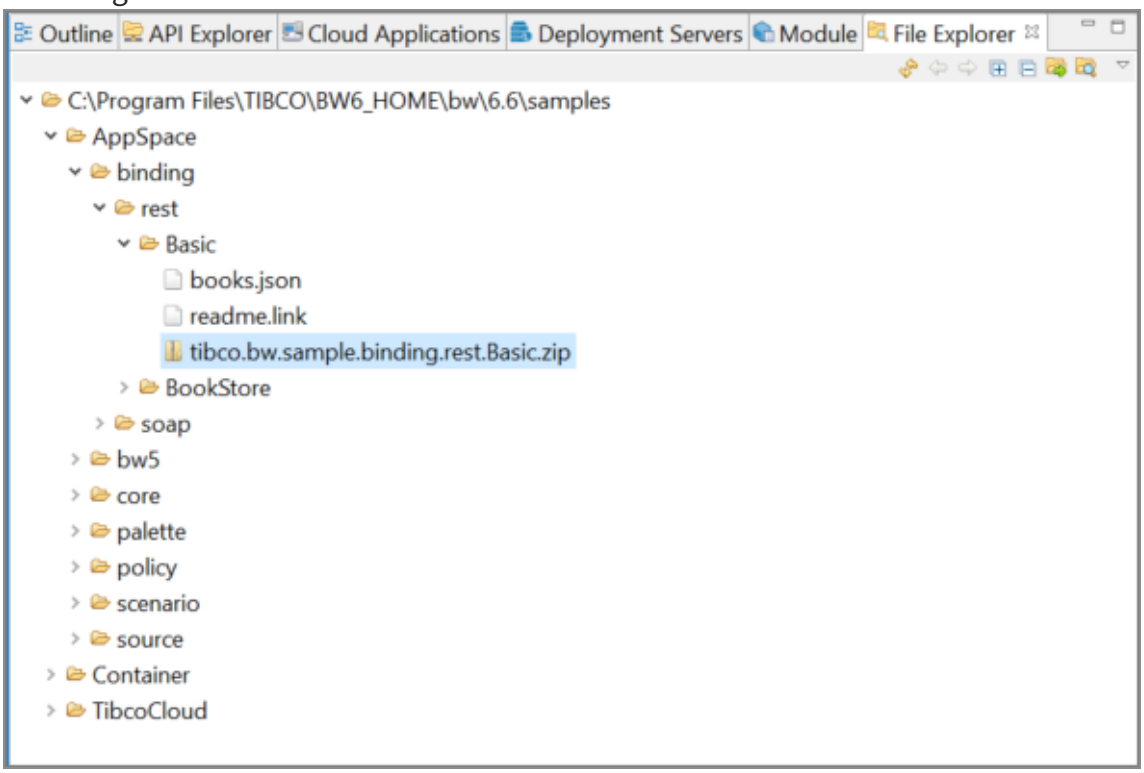


3. Select the required sample.

   The **File Explorer** view displays a top-level listing of all the samples.

   > **Note:** If the workspace is from an older version, you need to locate the samples manually using the **Open Directory to Browse** button available on the **File Explorer** view. TIBCO Business Studio™ for BusinessWorks™ stores the samples at `tibco-home/bw/6.6/samples`.

4. Navigate to the directory containing the sample that you want to access and do the following:



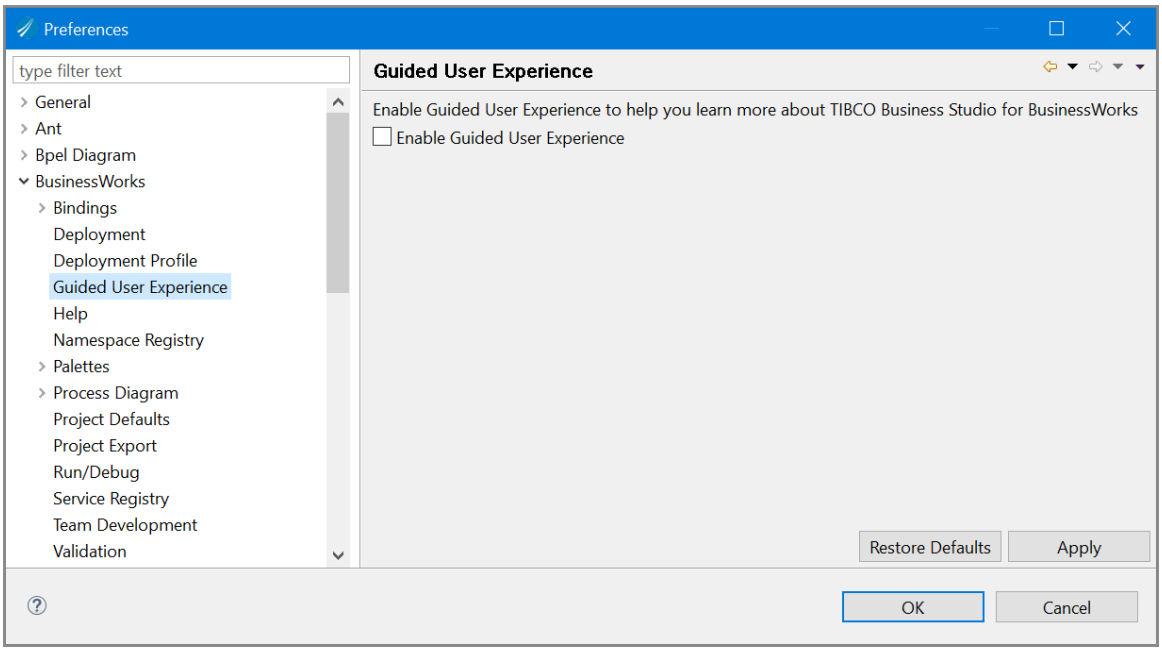
   - Double-click the archive file, `<sample_name>.zip`, to import the sample application to your workspace. For example, double-click `tibco.bw.sample.binding.rest.Basic.zip` to import the application and the application module to your workspace. Alternatively, you can right-click the required project folder and click **Import Selected Projects** option to import the required application.
   - Click `readme.link` to open the sample Help page in the **Reference - online** view. For example, by clicking **binding > rest > Basic > readme.link** you can

access the help page for a Basic REST Sample application. You can access the help page of all samples by following the same procedure.

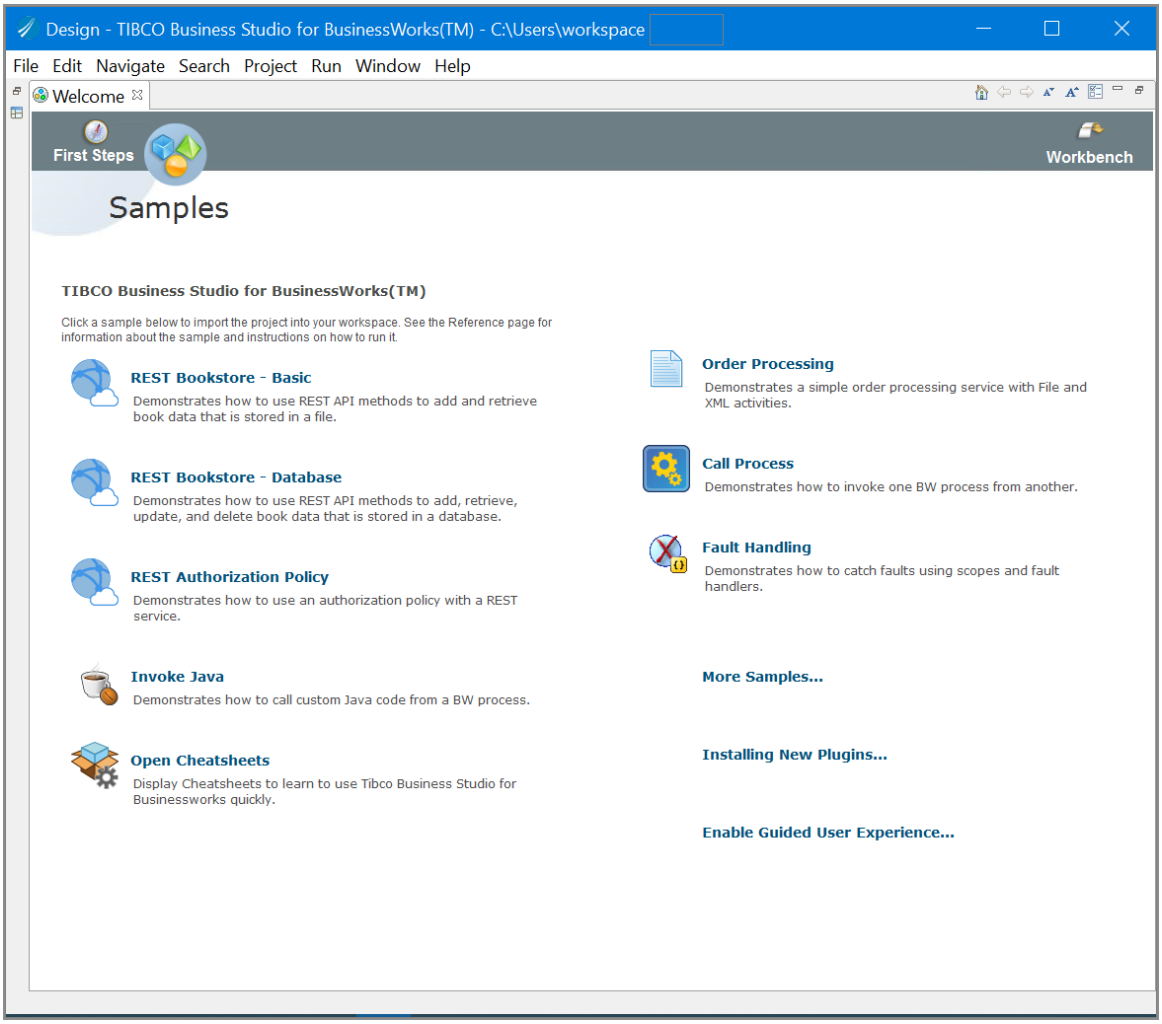
Guided User Experience in TIBCO Business Studio for BusinessWorks

TIBCO Business Studio for BusinessWorks introduces a new feature, guided user experience, wherein you can see notifications, pop-up messages, tooltips, or suggestions while performing an operation.

To enable the guided user experience option, navigate to **Window > Preferences > BusinessWorks > Guided User Experience** and select the **Enable Guided User Experience** checkbox on the Preferences window.

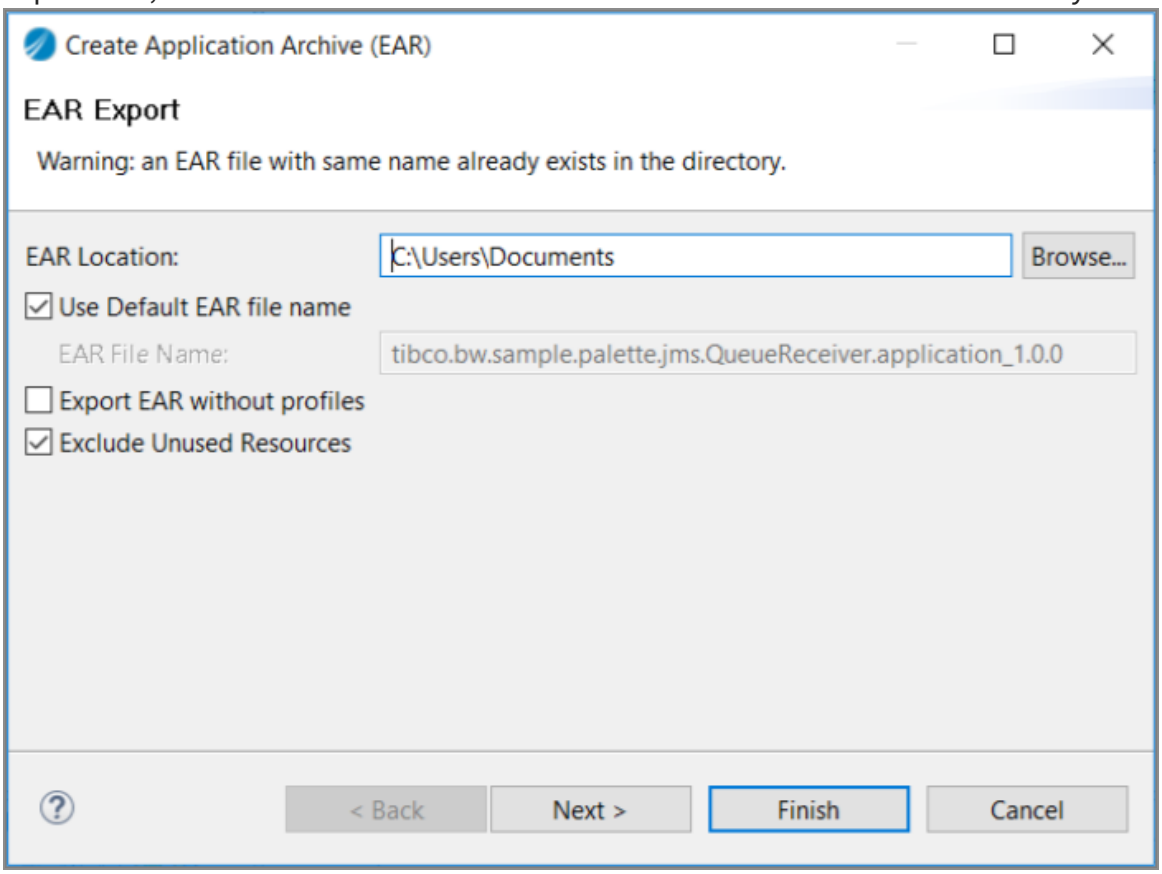


Or click **Enable Guided User Experience...** on the Welcome screen and select the **Enable Guided User Experience** checkbox on the Preferences window.



For example, while developing an application, when you click **Generating EAR**, the **Create Application Archive (EAR)** window is displayed. If you have already enabled guided user

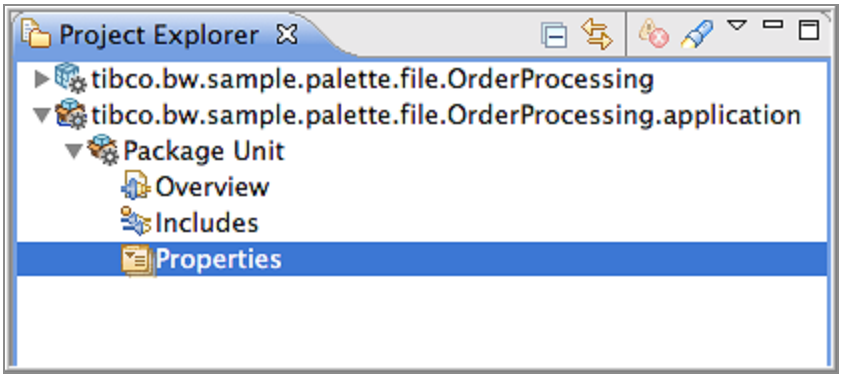
experience, the **Exclude Unused Resources** checkbox is visible and selected by default.



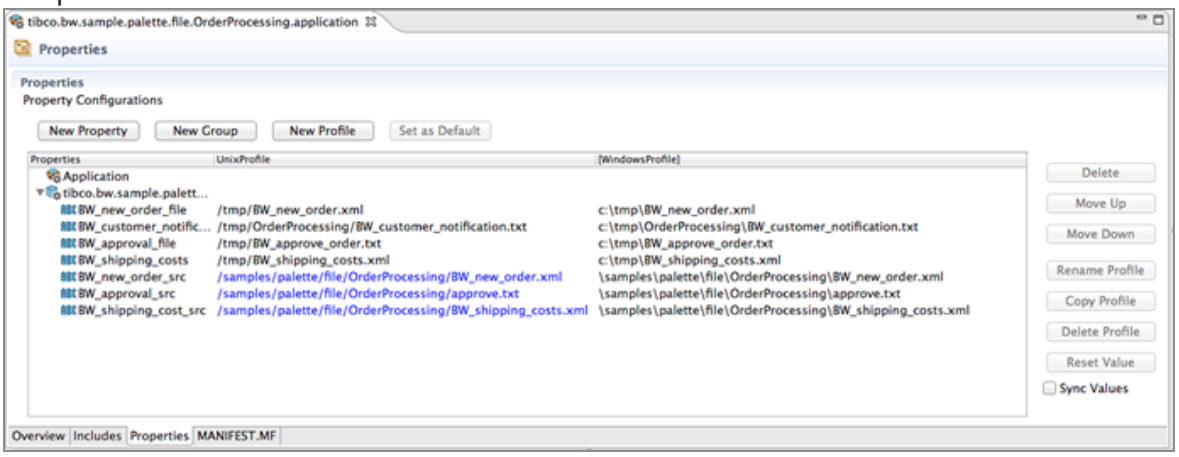
Setting the Default Application Profile

Procedure

1. Before executing the sample, ensure to update the default application profile according to the OS Platform you are using.
2. Click the**<*Project*>.application**, expand **Package Unit** folder and open the **Properties** file.



3. If you are using Unix based system, then select **UnixProfile** column header to enable **Set as Default** button. Click the **Set as Default** button to set it as the default application profile. The default profile is indicated by the straight brackets [ ] around the profile name.



If you are using Windows system, then ensure the **WindowsProfile** is set as default before running or debugging this application in TIBCO Business Studio™ for

BusinessWorks™.

4. Save your project.

Creating a ActiveMatrix BusinessWorks Project from Template

You can use an existing ActiveMatrix BusinessWorks project template to create a new project.

Before you begin

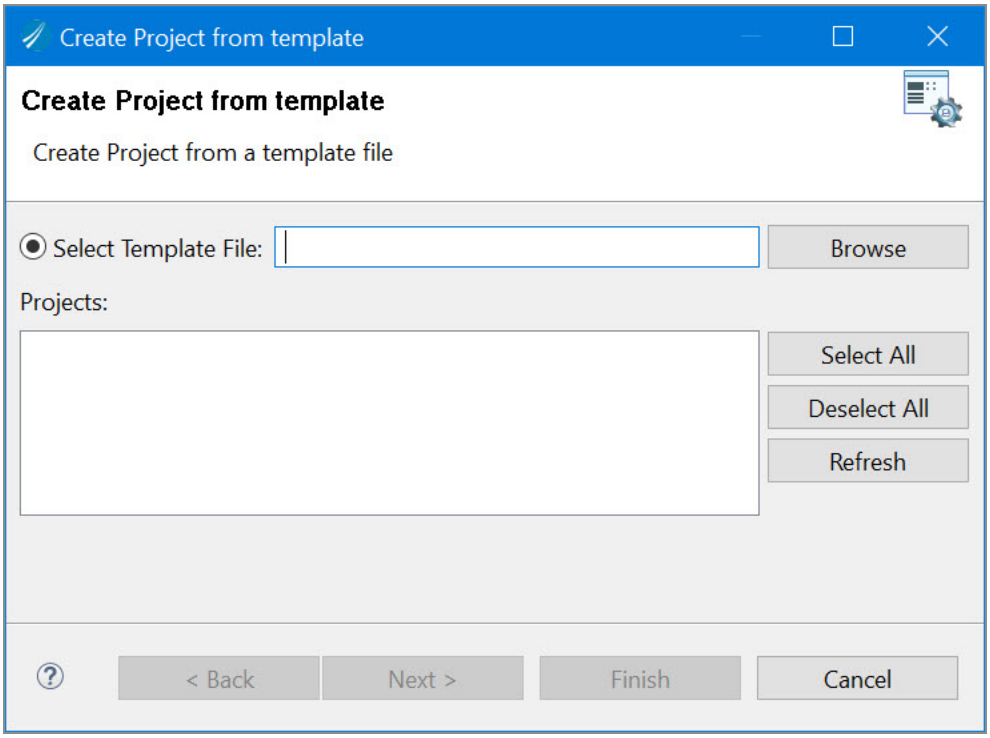
TIBCO Enterprise Message Service must be running.

Procedure

1. Navigate to **File > New > Project > BusinessWorks** and click the **BusinessWorks Project from Template** option. Click **Next**.

   Or click the icon on the toolbar.

   The *Create Project from template* dialog is displayed.



2. In the Select Template File field, provide the template file location from the samples directory, **bwTemplate > tibco.bw.sample.Template.TemplateSample.1.0.0.bwtemplate**. The project in the template is displayed.
3. To see information about template, click **Next**.
4. To rename the project, click **Next**. In the Existing Name column, select the project and provide the name in the New Name column.
5. Click **Finish**. On successful project creation, you can see the template project in the Project Explorer pane.
6. In the Project Explorer view, expand the **tibco.bw.sample.Template.TemplateSample** project.
7. Verify your TIBCO Enterprise Message Service connection.
    a. Fully expand the **Resources** directory.
    b. Double-click **JMSConnectionResource.jmsConnResource**.
    c. In the Basic Configuration pane, click the **Test Connection** button to verify the connection.
8. Fully expand the **Processes** directory and double-click **Process.bwp**.
9. Click **Run > Debug Configurations**.
10. At the left of the Debug Configurations wizard, expand **BusinessWorks Application** and select **BWApplication**.
11. In the **Applications** tab, click the **Deselect All** button if you have multiple applications. Select the checkbox next to the **tibco.bw.sample.Template.TemplateSample.application** option.
12. Click **Debug**. This runs the sample in the Debug mode.
13. Click the ■ (Terminate) icon to stop the process.

Result

The Console displays messages similar to the following:

```
13:11:42.211 INFO  [bwEngThread:In-Memory Process Worker-1]
com.tibco.bw.palette.generalactivities.Log.tibco.bw.sample.Template.Temp
lateSample.Log - Sending JMS Message
```

```
13:11:42.243 INFO  [bwEngThread:In-Memory Process Worker-1]
com.tibco.bw.palette.generalactivities.Log.tibco.bw.sample.Template.Temp
lateSample.Log1 - JMS Message sent successfully

13:11:42.337 INFO  [bwEngThread:In-Memory Process Worker-2]
com.tibco.bw.palette.generalactivities.Log.tibco.bw.sample.Template.Temp
lateSample.Log2 - Message received JMS Message sent
```

Understanding the Configuration

The JMS Send Message activity sends the messages with the message style set to Queue. The Get JMS Queue Message activity receives the JMS messages from a Queue.

Unit Testing

Unit testing in ActiveMatrix BusinessWorks consists of verifying whether individual activities in a process are working as expected. While you can run unit tests on processes at any time during the development cycle, testing the processes before you push the application to the production environment might help you to identify issues earlier.

This section explains how to use the Maven plug-in and its features to achieve unit testing of ActiveMatrix BusinessWorks projects.

In the ActiveMatrix BusinessWorks application, the following sample projects are provided:

- **MainProcessWithUnitTestDemo.zip:** Sample to show the assertion or mocking support of main processes, mocking support of process starters, and REST and SOAP service binding.
- **UnitTestDemoProject.zip:** Sample to show the assertion or mocking support of subprocesses.
- **tibco.bw.UT.sample.Calculator.module.zip:** Sample shows the use case of implementing unit test cases in the Shared Module of your application.

Following are the details of the processes in the sample projects:

MainProcessWithUnitTestDemo:

- MainProcessWithProcessStarterMocked: Shows assertion support in the main process along with mocking support on the process starter.
- RESTServicBindingeMocked: Shows mocking support on the REST Service binding.
- SOAPServiceBindingMocked: Shows mocking support on the SOAP Service binding.

UnitTestDemoProject:

- SubProcesswithActivityAssertion.bwp: Shows mocking support and Activity assertion in a subprocess.
- SubProcessWithPrimitiveAssertion.bwp: Shows mocking support and Primitive assertion in a subprocess.

tibco.bw.UT.sample.Calculator.module:

- MainProcess: This process invokes calculation operations based on input received from the HTTPReceiver. It demonstrates support for mocking the HTTPReceiver and Primitive assertion on the "cube" (a call process activity).

  In the Shared Module, you can perform the following processes:

  - Addition: This process provides inputs (number 2 and number 3) for addition and verifies the result of the "end" activity using a primitive assertion.
  - Cube: This process provides input (number 8) for the cube operation and verifies the result of the "end" activity using an activity assertion.
  - Multiply: This process provides inputs (number 2 and number 3) using an "input XML file" for multiplication and verifies the result of the "end" activity using an activity assertion with a reference to a gold input file on the "end" activity.
  - Square: This process provides input (number 2) using an "input XML file" for squaring and verifies the result of the "end" activity using a primitive assertion.

Maven Test Sample of Main Process

This sample covers mocking for activities in process, service binding, starter activity and types of activity assertions.

Before you begin

- Ensure that Maven is installed on your system.
- Ensure that the Maven plugin is installed correctly in your bw studio by verifying the Maven folder present under `tibco_home\bw\6.8` directory.

Procedure

1. In the samples directory, select **AppSpace > UnitTesting > MainProcessWithUnitTestcases** and double-click **MainProcessWithUnitTestDemo.zip**. For more information, see Accessing Samples.
2. Mavenize the project by right-clicking **MainProcessWithUnitTestDemo.application** and selecting the **Generate POM for Application** option. The POM Generation

window is displayed.

3. Enter the Parent POM details such as Group Id, Parent Artifact Id, and so on, and click **Finish**.

> **Note:** The workspace is indexed after generating the POM files for the first time and may take some time. You can continue with the following steps and allow the indexing to run in the background.

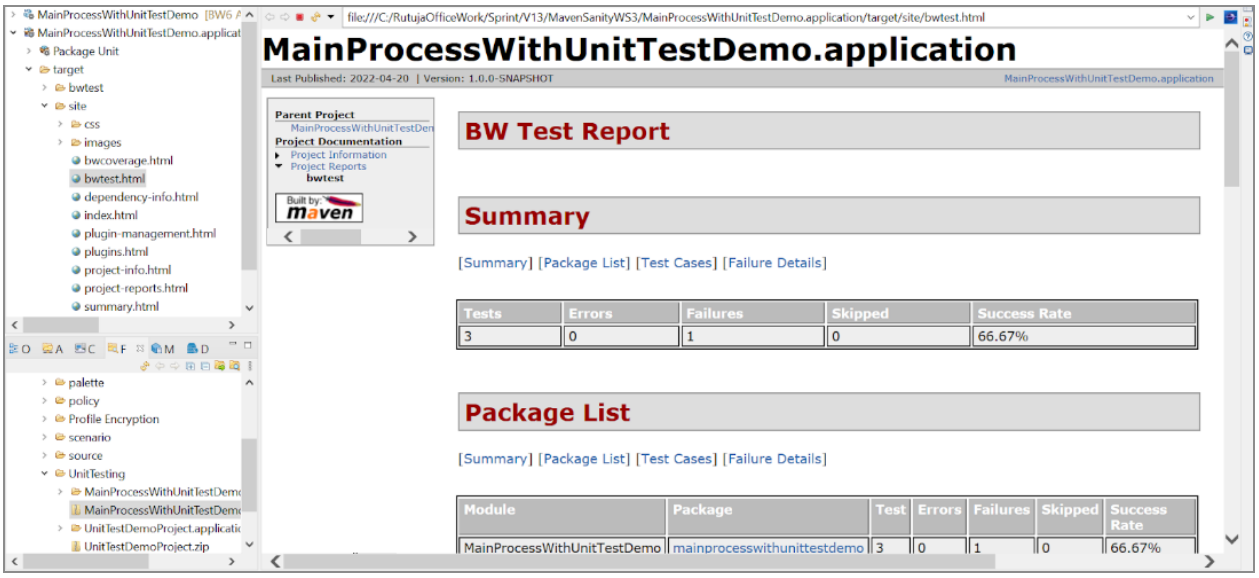
4. **MainProcessWithUnitTestDemo.application.parent** is generated and the project is converted to Maven (Eclipse project) nature.
5. Right-click **MainProcessWithUnitTestDemo.application.parent** and select **Run as > Maven test**.
6. Check the console to see the result.

Result

The console displays messages similar to the following:

```
[INFO]
Tests for mainprocesswithunittestdemo.MainProcessWithProcessStarterMocked
Tests run : 1    Success : 1    Failure : 2     Errors : 0
Tests for mainprocesswithunittestdemo.RESTServicBindingeMocked
Tests run : 1    Success : 1    Failure : 0     Errors : 0
Tests for mainprocesswithunittestdemo.SOAPServiceBindingMocked
Tests run : 1    Success : 1    Failure : 0     Errors : 0

Results
Success : 3    Failure : 2    Errors : 0
```

> **Note:** Here we have covered 3 positive scenarios and 2 negative scenarios by giving incorrect value to assertions. If you want to see the result in the HTML form, use Maven “site” goal, while running the application.



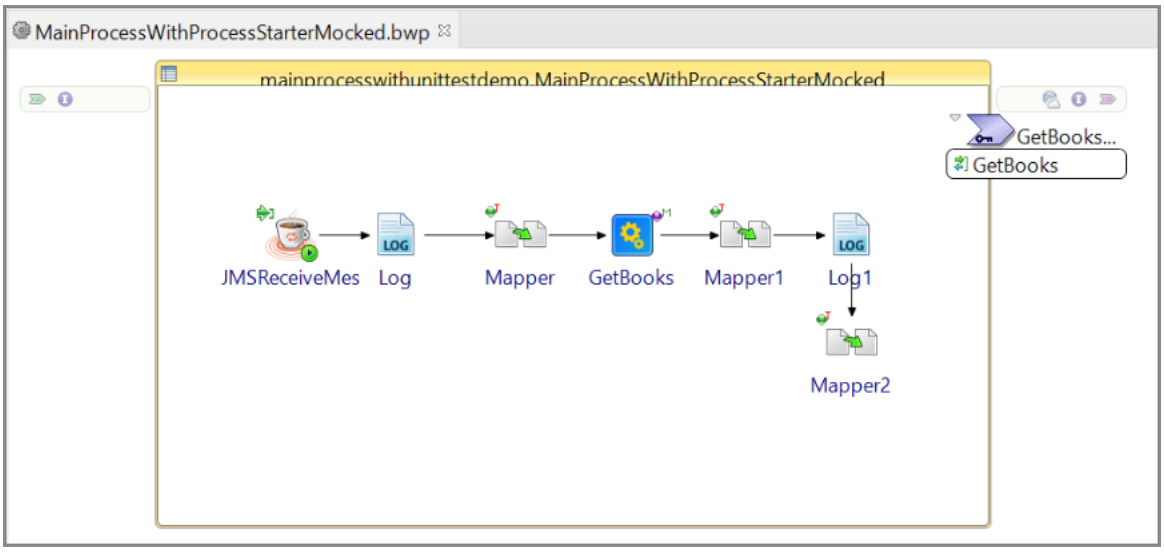
Understanding the Configuration

The project contains the following processes.

- **MainProcessWithProcessStarterMocked.bwp:**

  This process has JMSReceiver as a starter activity. Whenever a message is published on JMS queue, this process gets triggered. From the JMS message content, input for “GetBooks By author” SOAP call is constructed using mapper activity. Later in the process,“GetBooks By Author” SOAP call is made, which returns corresponding book details, and then using a mapper, the output response is converted in required format.

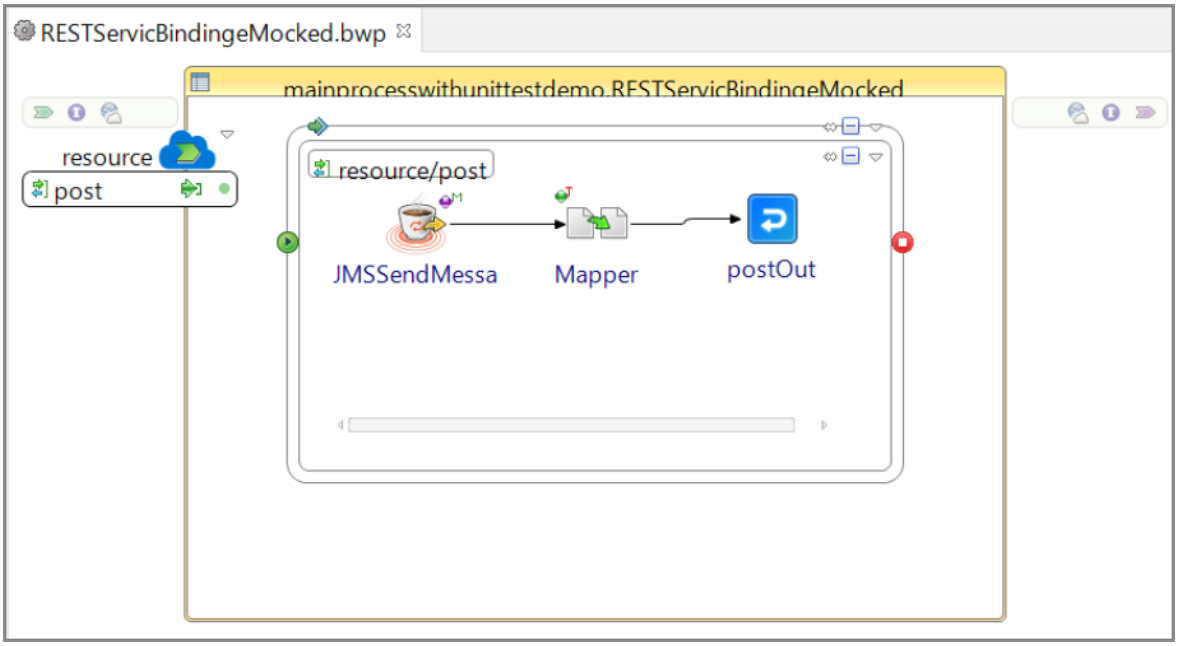
  This process shows assertion support in the main process along with mocking support on process starter.



- **RESTServicBindingeMocked.bwp:**

  This process is triggered by the POST endpoint call, which requires book details like isbn, name, description, and author name in the post body. First this process publishes the book name, which it received in the POST body, as JMS message body and author name as dynamic property to a JMS queue. Later using mapper activity output for the REST endpoint is constructed and then using mapper the output is mapped to the postOut (reply activity).

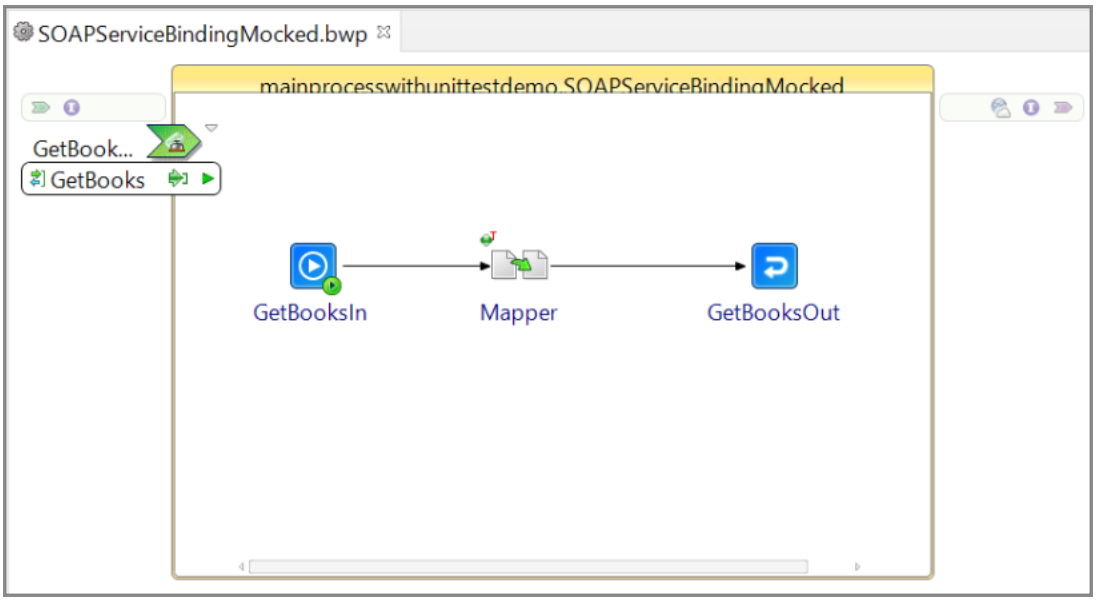
  This process shows mocking support on the REST Service binding.



- **SOAPServiceBindingMocked.bwp:**

  This process is triggered by the Get Books SOAP call, which receives 'author' as an input. After accepting author as input, depending on the author, the required output is constructed using mapper activity and then using GetBooksOut (Reply activity) the response like title, author, date isbn, and publisher is returned in the response.

  This process shows mocking support on the SOAP Service binding.



Troubleshooting

If you find the `application = null` error in the console then check the Maven version by navigating to **Window > Preferences > Maven Defaults**. The BW6 Maven Plugin Version should be the same as the plugin version you installed while installing TIBCO Business Studio for BusinessWorks. Similarly, check Maven version in the `pom.xml` file.

Refer the Troubleshooting guide for any issues.

Maven Sample of Unit Test Demo Project

This sample covers mocking for activities in process, subprocess, service binding, starter-end activity, and types of activity assertion.

Before you begin

- Ensure that Maven is installed in your system.
- Ensure that the Maven plugin is installed correctly in your bw studio by verifying the Maven folder present under `tibco_home\bw\6.8` directory.

Procedure

1. In the samples directory, select **AppSpace > UnitTesting > SubProcessWithUnitTestcases** and double-click **UnitTestDemoProject.zip**. For more information, see Accessing Samples.
2. In the Project Explorer, right-click **UnitTestDemoProject.application** and select **Generate POM for Application**. The POM Generation window is displayed.
3. Enter the Parent POM details such as Group Id, Parent Artifact Id, and so on, and click **Finish**.

   Note: The workspace is indexed after generating the POM files for the first time and may take some time. You can continue with the following steps and allow the indexing to run in the background.

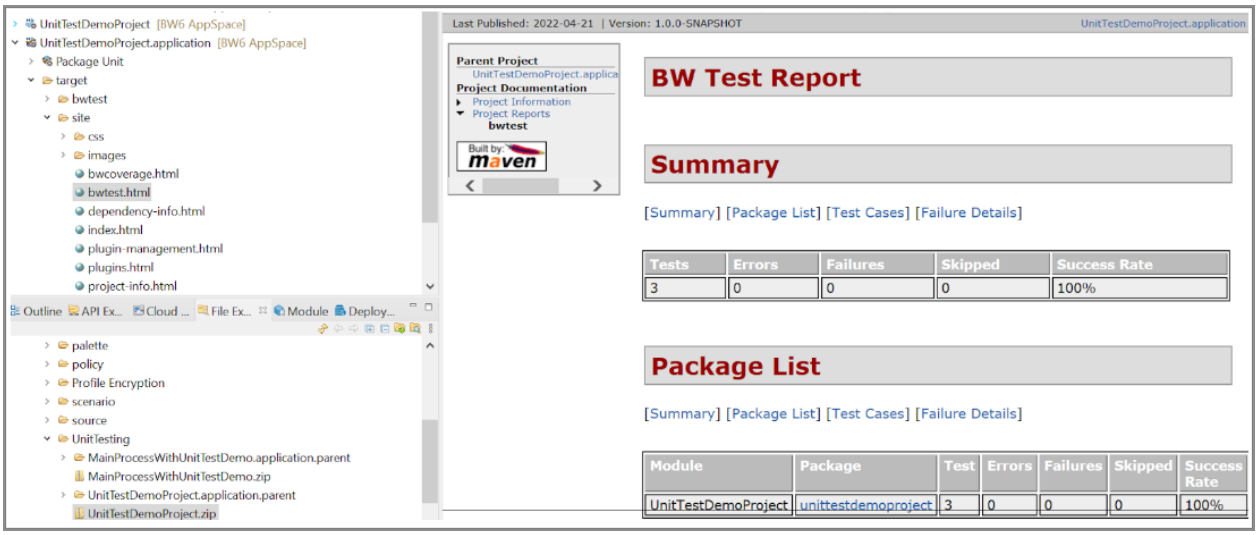
4. **UnitTestDemoProject.application.parent** is generated and the project is converted to Maven (Eclipse project) nature.
5. Right-click **UnitTestDemoProject.application.parent** and select **Run as > Maven test**.
6. Check the console to see the result.

Result

The console displays messages similar to the following:

```
Tests for unittestdemoproject.SubProcesswithActivityAssertion
Tests run : 2    Success : 2    Failure : 0     Errors : 0
Tests for unittestdemoproject.SubProcessWithPrimitiveAssertion
Tests run : 1    Success : 1    Failure : 0     Errors : 0

Results
Success : 3    Failure : 0    Errors : 0
```

Note: Here we have covered 3 positive scenarios. If you want to see the result in the HTML form, use maven “site” goal, while running the application.

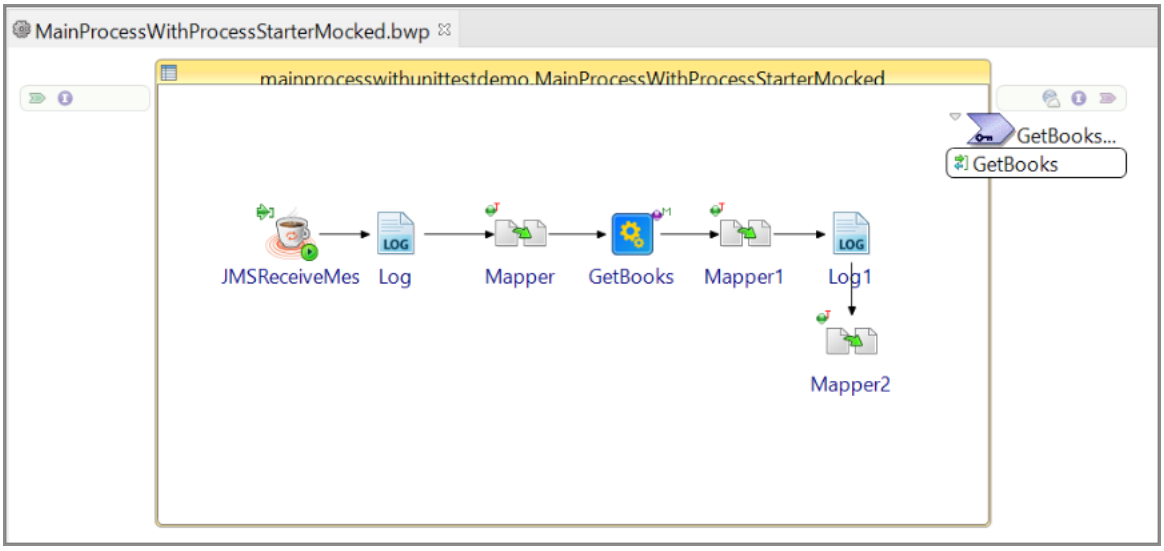


Understanding the Configuration

The project contains the following processes.

- **MainProcess.bwp:**

  This process has a timer as a starter activity, which is executed only once. Later this process makes a subprocess call to another subprocess, *SubProcessWithPrimitiveAssertion.bwp*.

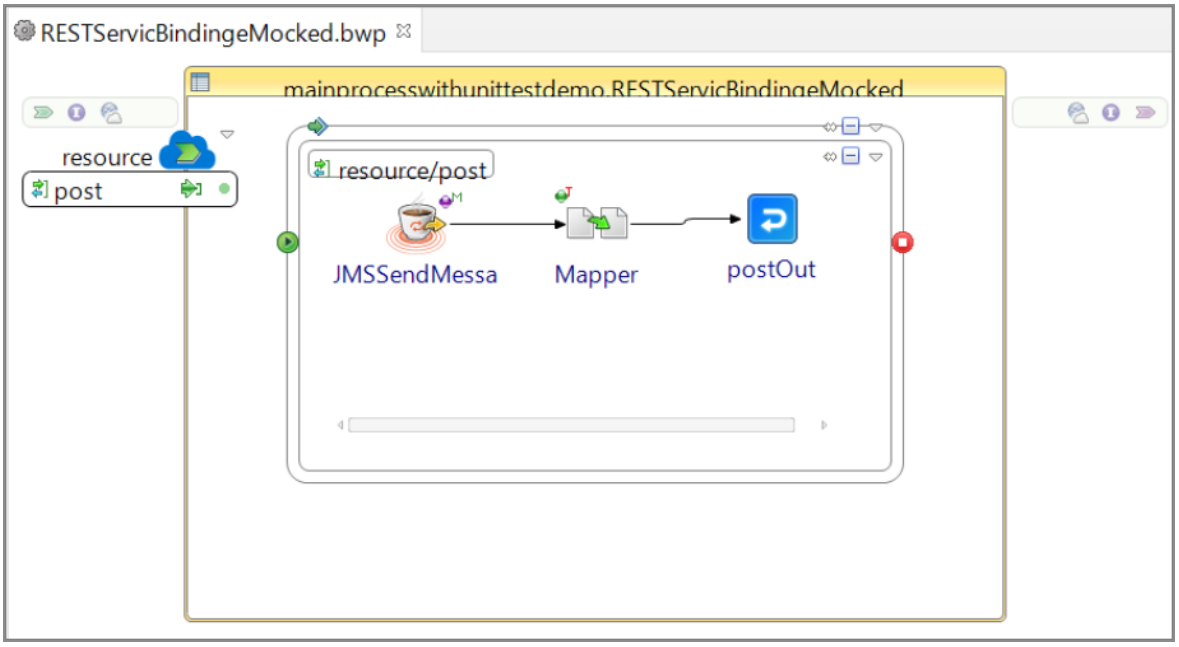


- **SubProcessWithPrimitiveAssertion.bwp:**

  This process does not accept any input. First, it sends a request to a JMS destination, queue in this case, using the JMS Request Reply activity. Later, the JMS Request

Reply activity body is logged and the reply message is mapped to the required format using the mapper activity. The output of the mapper is mapped with the end activity.

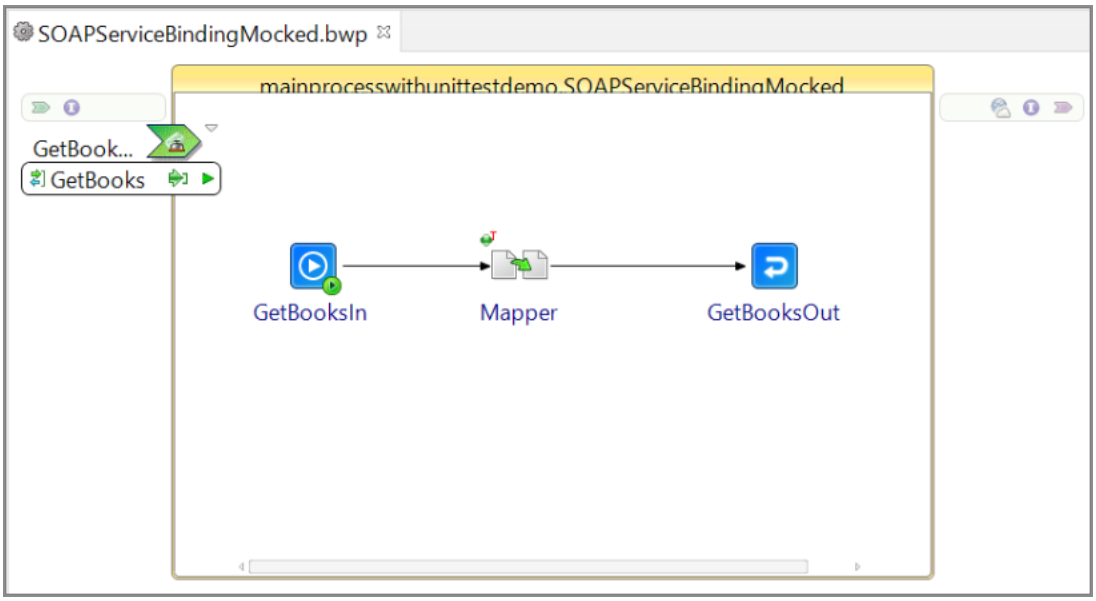
This process shows mocking support and Primitive Assertion in a subprocess.



- **SubProcesswithActivityAssertion.bwp:**

  This process accepts authorID as an input of the Start activity. The book details corresponding to the the authorID are displayed in the mapper. The GetBooks SOAP call is invoked by the author as an input. The GetBooks response is logged using the log activity.

  This process shows mocking support and Activity Assertion in a subprocess.



- **JMSReceiver.bwp:**

  This process has a JMS Message receiver as a starter activity which gets triggered when a message is received on a queue (queue.sample). Later this process replies to the JMS message received using the Reply to JMS Message activity and then log the message.

- **SOAPService.bwp:**

  This process implements SOAP Getbooks by author operation, which accepts author as an input and replies the corresponding book details of that author in the response. The response is hard-coded.

Troubleshooting

If you find the `application = null` error in the console then check the Maven version by navigating to **Window > Preferences > Maven Defaults**. The BW6 Maven Plugin Version should be the same as the plugin version you installed while installing TIBCO Business Studio for BusinessWorks. Similarly, check Maven version in the `pom.xml` file.

Refer Troubleshooting guide for any issue

Maven Sample of Shared Module Unit Test Cases

In this sample, a REST endpoint is exposed by using the `HTTPReceiver` activity. This activity is responsible for executing fundamental calculator operations.

Here, the behavior of the HTTPReceiver activity is simulated. After the operations are outlined in the mock `HTTPReceiver-mockOutput.xml` file, the corresponding actions are invoked and the obtained results are verified through assertions.

This particular sample also portrays the Test Suite feature that can encompass test cases for an application module.

Procedure

1. From the samples directory, select **AppSpace > UnitTesting > SharedModuleUnitTestcases** and double-click **tibco.bw.UT.sample.Calculator.OperationsSM.zip**. Wait for the shared module to be imported into the workspace. Then double-click **tibco.bw.UT.sample.Calculator.module.zip**.

   For more information, see Accessing Samples.

2. In the Project Explorer, right-click **tibco.bw.UT.sample.Calculator.module.application** and select the **Generate pom.xml** option. The POM Generation window is displayed.

   A project **tibco.bw.UT.sample.Calculator.module.application.parent** is generated if it does not exist. If it is already present, it then regenerates with updated JAR values.

3. To run the test cases irrespective of the test suites, right-click **tibco.bw.UT.sample.Calculator.module.application.parent** and select **Run as > Maven test**.

4. To see the result, check the console.

Result

The console displays messages similar to the following:

```
[INFO]
[INFO] ## Running Test for addition.bwt ##
[INFO] ## Running Test for cube.bwt ##
[INFO] ## Running Test for multiply.bwt ##
[INFO] ## Running Test for square.bwt ##
[INFO] Starting Tests in Module : tibco.bw.UT.sample.Calculator.OperationsSM
[INFO] Uploading tests for Processes in Module :  tibco.bw.UT.sample.Calculator.OperationsSM
 tibco.bw.ut.sample.calculator.operationssm.Addition
 tibco.bw.ut.sample.calculator.operationssm.Cube
 tibco.bw.ut.sample.calculator.operationssm.Multiply
 tibco.bw.ut.sample.calculator.operationssm.Square
[INFO] <tns:CalculatorOutput xmlns:tns="http://www.example.com/namespaces/tns/1693209694314"
xmlns:tns1="http://www.example.com/namespaces/tns/1693209556450"><tns:Result>5</tns:Result></tns:CalculatorOutput>
[INFO] <tns:CalculatorOutput xmlns:tns="http://www.example.com/namespaces/tns/1693209694314"
xmlns:tns1="http://www.example.com/namespaces/tns/1693209556450"><tns:Result>512</tns:Result></tns:CalculatorOutput>
[INFO] <tns:CalculatorOutput xmlns:tns="http://www.example.com/namespaces/tns/1693209694314"
xmlns:tns1="http://www.example.com/namespaces/tns/1693209556450"><tns:Result>6</tns:Result></tns:CalculatorOutput>
[INFO] <tns:CalculatorOutput xmlns:tns="http://www.example.com/namespaces/tns/1693209694314"
xmlns:tns1="http://www.example.com/namespaces/tns/1693209556450"><tns:Result>4</tns:Result></tns:CalculatorOutput>
[INFO]
Tests for tibco.bw.ut.sample.calculator.operationssm.Addition
Tests run : 1    Success : 1    Failure : 0     Errors : 0
Tests for tibco.bw.ut.sample.calculator.operationssm.Cube
Tests run : 1    Success : 1    Failure : 0     Errors : 0
Tests for tibco.bw.ut.sample.calculator.operationssm.Multiply
Tests run : 1    Success : 1    Failure : 0     Errors : 0
Tests for tibco.bw.ut.sample.calculator.operationssm.Square
Tests run : 1    Success : 1    Failure : 0     Errors : 0

Results
Success : 4    Failure : 0    Errors : 0
```

```
[INFO] ## Running Test for calculator.bwt ##
[INFO] Starting Tests in Module : tibco.bw.UT.sample.Calculator.module
[INFO] Uploading tests for Processes in Module :  tibco.bw.UT.sample.Calculator.module
 tibco.bw.UT.sample.Calculator.module.MainProcess
[INFO] 2023-10-03T11:59:01,042 INFO  [bwEngThread:In-Memory Process Worker-5] com.tibco.bw.palette.generalactivities.Log.tibco.bw.UT.sample.Calculator.module.Log - Got
calculation Request
[INFO] <tns:CalculatorOutput xmlns:tns="http://www.example.com/namespaces/tns/1693209694314"
xmlns:tns1="http://www.example.com/namespaces/tns/1693209556450"><tns:Result>8</tns:Result></tns:CalculatorOutput>
[INFO] 2023-10-03T11:59:01,136 INFO  [bwEngThread:In-Memory Process Worker-5] com.tibco.bw.palette.generalactivities.Log.tibco.bw.UT.sample.Calculator.module.Log1 - 8
[INFO]
Tests for tibco.bw.UT.sample.Calculator.module.MainProcess
Tests run : 1    Success : 1    Failure : 0     Errors : 0

Results
Success : 1    Failure : 0    Errors : 0
[INFO]
```

You can also configure the Maven goal to run specific test suites. For example, in the **Run As -> Maven build..** field for "Goals", you can enter the following command:

```
test -
DtestSuiteName=TestSuites\CommutativeOperations.bwts;TestSuite\Calculati
onSuite.bwts
```

Result

When running test suites with the Maven goal, the console displays messages similar to the following:

```
Tests for TestSuite TestSuites\CommutativeOperations.bwts
Tests run : 2    Success : 2    Failure : 0     Errors : 0

Results
Success : 2    Failure : 0    Errors : 0
[INFO]
[INFO]
[INFO]  ## Running Test Suite TestSuite\CalculationSuite.bwts ##
[INFO]      Running Test for calculator.bwt
[INFO] Starting Tests in Module : tibco.bw.UT.sample.Calculator.module
[INFO] Uploading tests for Processes in Module :  tibco.bw.UT.sample.Calculator.module
 tibco.bw.UT.sample.Calculator.module.MainProcess
[INFO] 2023-10-03T12:04:54,790 INFO  [bwEngThread:In-Memory Process Worker-3] com.tibco.bw.palette.generalactivities.Log.tibco.bw.UT.sample.Calculator.module.Log - Got
calculation Request
[INFO] <tns:CalculatorOutput xmlns:tns="http://www.example.com/namespaces/tns/1693209694314"
xmlns:tns1="http://www.example.com/namespaces/tns/1693209556450"><tns:Result>8</tns:Result></tns:CalculatorOutput>
[INFO] 2023-10-03T12:04:54,879 INFO  [bwEngThread:In-Memory Process Worker-3] com.tibco.bw.palette.generalactivities.Log.tibco.bw.UT.sample.Calculator.module.Log1 - 8
[INFO]
Tests for TestSuite TestSuite\CalculationSuite.bwts
Tests run : 1    Success : 1    Failure : 0     Errors : 0

Results
Success : 1    Failure : 0    Errors : 0
```

Binding Features

Endpoints expose the location of the service to the service partner. An endpoint binding specifies the transport to use for the endpointany transport-specific properties. This section includes samples that show how to use bindings correctly.

Using Basic REST Binding

In this **REST Bookstore** sample you can use the RESTful service to add, and retrieve books from a Bookstore (in this sample, Bookstore is files as a datastore). This sample uses two REST methods: POST and GET.

Before you begin

Install the latest Google Chrome version in the standard default location.

> **Note:** If you are accessing the sample from the **Welcome** page of TIBCO Business Studio™ for BusinessWorks™, go directly to **Step 3**.

Procedure

1. In the **samples** directory, select **binding > rest > Basic** and double-click **tibco.bw.sample.binding.rest.Basic**. For more information, see the Accessing Samples section.
2. In the **Project Explorer**, expand the **tibco.bw.sample.binding.rest.Basic** project.
3. Set the default application profile to match the OS you are running on. For more information, see Setting the Default Application Profile.
4. Fully expand the **Processes** directory and double-click **books.bwp**.
5. Click **Run > Debug Configurations**.
6. At the left hand tree of the Debug Configuration wizard, expand **BusinessWorks Application** and select **BWApplication**.

7. Click the **Applications** tab, and then click the **Deselect All** button, if you have multiple applications. Select the checkbox next to **tibco.bw.sample.binding.rest.Basic.application**.
8. Click **Debug**. This runs the sample in Debug mode.

   The console window shows engine messages similar to: Started BW Application [tibco.bw.sample.binding.rest.Basic.application:1.0].

   The OSGI command to list REST and Swagger URLs is `lrestdoc`, which lists the following discovery URL:

   [Application Name]: tibco.bw.sample.binding.rest.Basic.application [Discovery Url]: http://localhost:7777/tibco.bw.sample.binding.rest.Basic.application.
9. Launch Google Chrome browser and open http://localhost:7777/tibco.bw.sample.binding.rest.Basic.application.

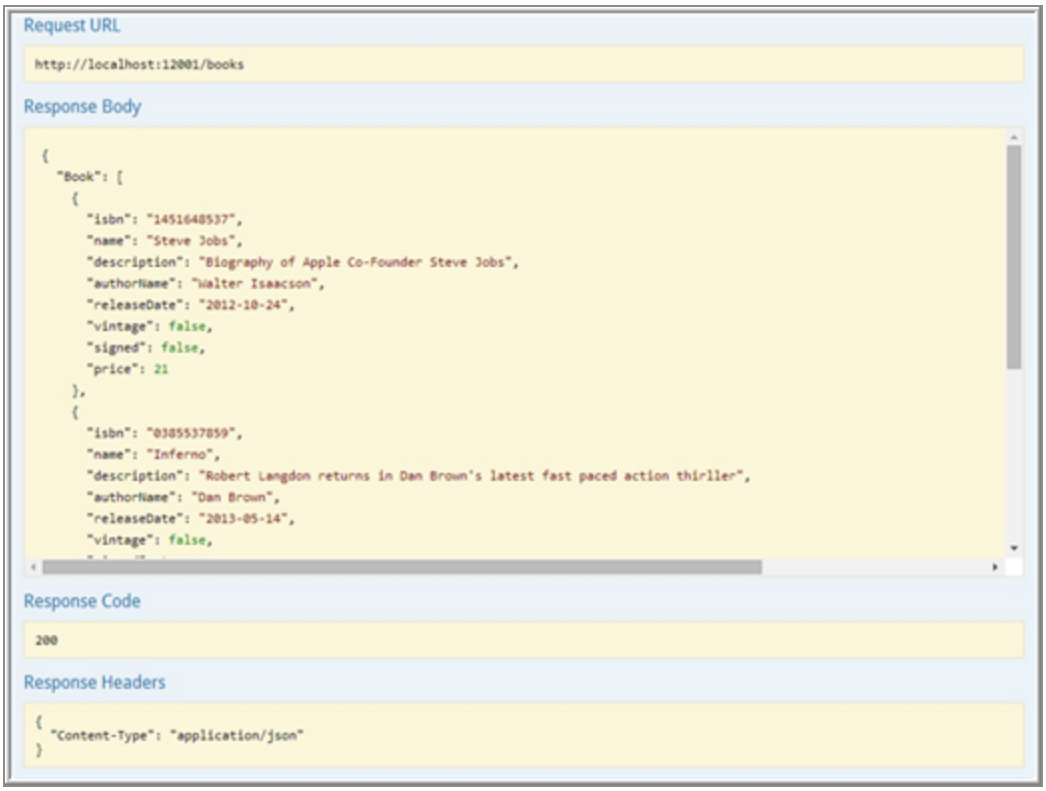
Result

The Swagger API Page displays:

Click **POST** and provide the JSON payload.

You can use the `books.json` payload provided with the sample present at `BW_HOME\samples\binding\rest\Basic\books.json`. On successful POST of Data, the `books.log` file is created in the `c:/tmp/RestBasic` folder (or `/tmp/RestBasic` on Unix Platform), with the data you just posted.

Click **Get > Try it out** to view all the books that you have just posted. You can see the data similar as the following:



You can also change the output file location.

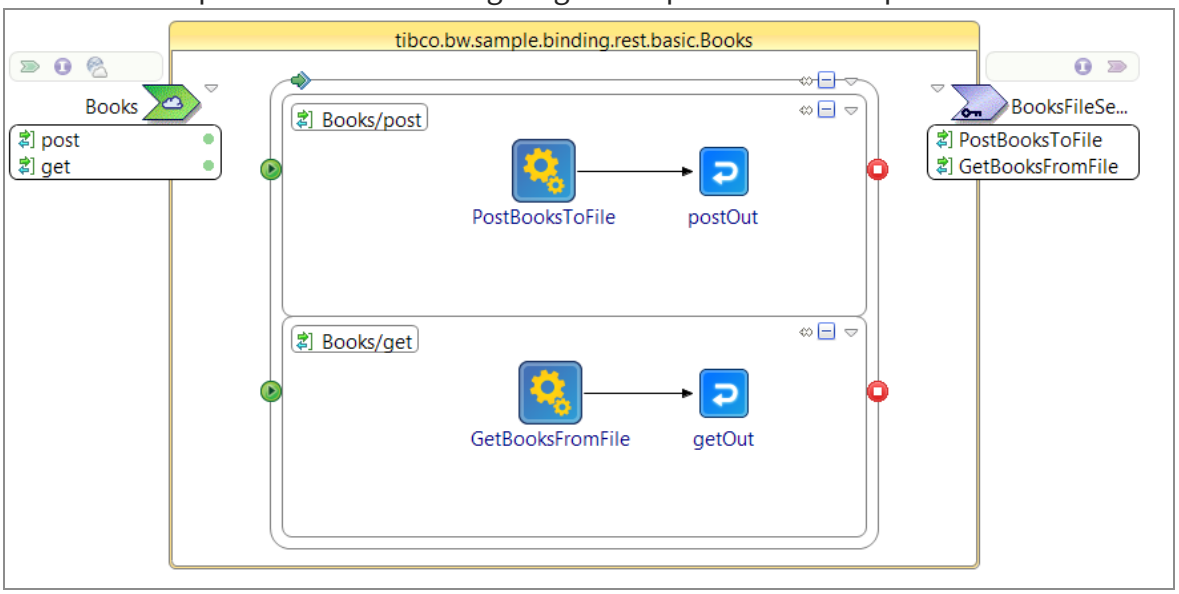
To change the output file location, go to **tibco.bw.sample.binding.rest.Basic.application > Package Unit**, and open **Properties** file for editing. Depending on your Operating System under test, edit the output file location in either the **UnixProfile** column or **WindowsProfile** column.
The `Books_Invoke.log` file is generated at `c:\tmp\RestBasic\`. This file contains POST Books and GET Books output from the Client process.

Understanding the Configuration

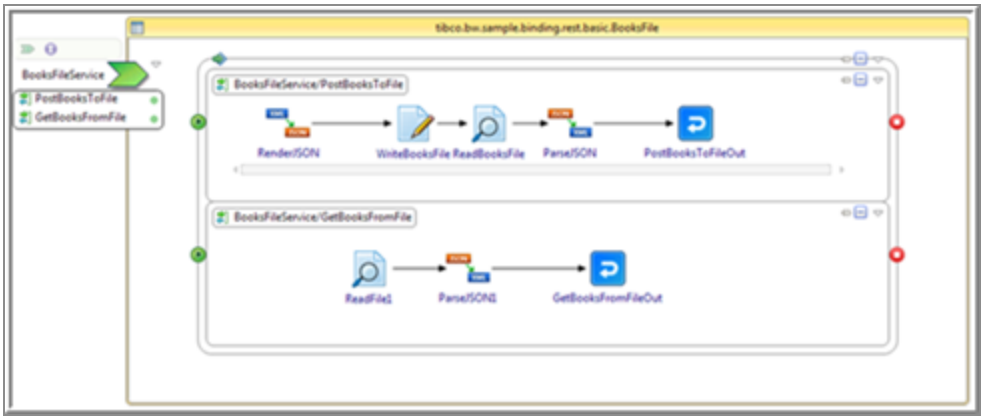
The project contains the following processes.

- **Books Process**: In this process, a REST service is created that performs two operations, POST and GET. The POST operation posts one or more books to the bookstore or books collection (in this case a file) and the GET operation retrieves all the books from the Bookstore (file). In the **Books/post** operation, a sub process is called to add books to the file. The **PostBooksToFile** operation is implemented in the **BooksFile** subprocess. In the **Books/get** operation, a sub process is called to

retrieve books from the file. The **GetBooksFromFile** operation is implemented in the **BooksFile** subprocess. The following diagram depicts the Books process.



- **BooksFile Subprocess**: File handling is managed using the BooksFile sub process, which adds and retrieves books from the file. The BooksFileService service implements the operations to add and get books from the file. In BooksFileService/PostBooksToFile, **Render JSON** activity is used to render data that the user has posted, write it to a file, read the file, parse the data, and then post the data back to the user. In BooksFileService/GetBooksFromFile, the file previously created is read, parsed, and then the data is returned to the user.

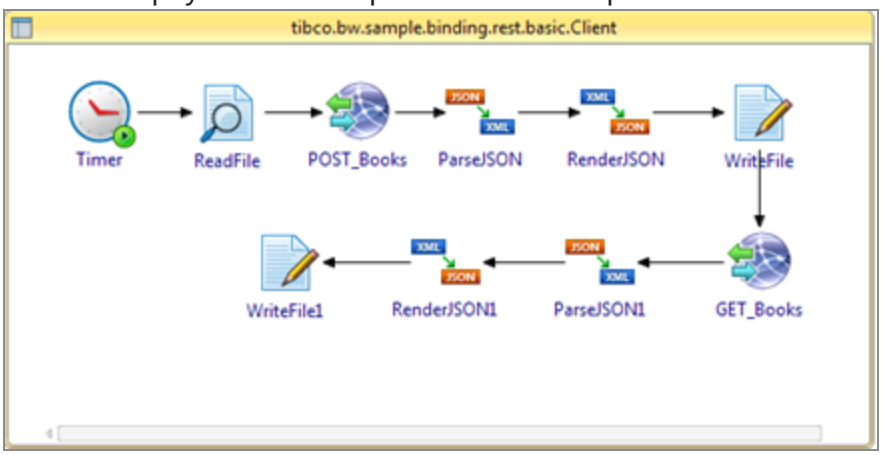


- **Client Process**: This is a REST Client process which uses **Invoke REST API** activity to retrieve data from the REST Server. This process is used to invoke the POST and GET operations on the REST resource.

  For example, in POST_Books, we provide HTTP Client Resource that listens to the

same host and port as that of the server. The Resource Path is "`/books`", HTTP Method is "POST", and Request and Response type is JSON. In the Message Body, we provide the JSON payload which is read from the file present at `BW_HOME\samples\binding\rest\BookStore\samplejson\books.json`.

This JSON payload is then posted to the output file.



Troubleshooting

- If you do not see the REST Swagger API page, verify whether the application has started.
- If you see any problem markers in the project, clean the project by invoking **Project > Clean** or try switching to a clean new workspace.
- If you get any File was not found exception, verify whether the `books.json` file is present at the location as described in the **Input_File** Module Property.

Using REST to Manage Books for a Bookstore

This **REST Bookstore Sample Using a Database** shows how to use the RESTful service to add, delete, update, and retrieve books from Bookstore.

The sample uses the following HTTP methods for the Books REST resource:

- **POST**: Add books to the bookstore
- **GET**: Get books from the bookstore

- **PUT**: Updates books to the bookstore
- **DELETE**: Deletes books from the bookstore

Before you begin

Install the latest Google Chrome version at the standard default location.

Install the PostgreSQL Database:

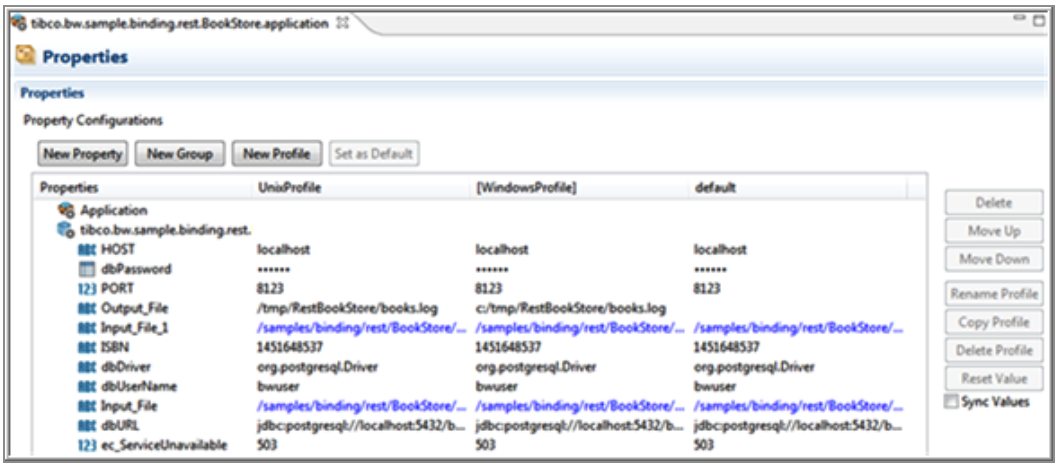
1. Download and install PostgreSQL from http://www.postgresql.org/download/.
2. Go to SQL Shell (psql) from the installed program.
3. The PostgreSQL command line window starts.
4. Open the DB and table creation script, `dbsetup.sql` from the scripts folder under `TIBCO_HOME/bw/n.n/samples/binding/rest/BookStore/scripts`.
5. Run the script from the command line as per the instructions given in the readme file present under the `scripts` folder.

You see the database and the tables created from the pgAdmin UI from the installed program.

> **Note:** If you are accessing the sample from the Welcome page of TIBCO Business Studio™ for BusinessWorks™, go directly to **Step 3**.

Procedure

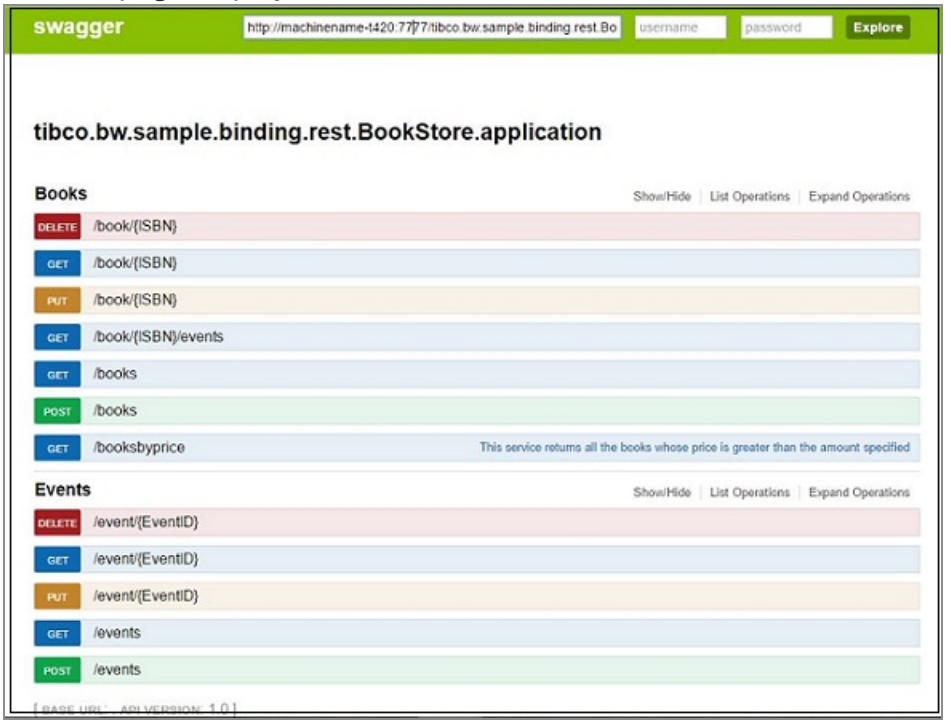
1. In the **samples** directory, select **binding > rest > Bookstore** and double-click **tibco.bw.sample.binding.rest.BookStore**. For more information, see Accessing Samples.
2. In **Project Explorer** expand the **tibco.bw.sample.binding.rest.BookStore** project.
3. Set the default application profile to match the OS you are running on. For more information, see Setting the Default Application Profile.



4. Expand **tibco.bw.sample.binding.rest.Bookstore.application > Package Unit** folder. Provide valid values for the application properties. Provide a valid username, password and database URL to connect to your PostgreSQL database in the module properties, if different from the default setting.
5. Verify your JDBC connection.
    a. Fully expand the **Resources** directory.
    b. Double-click **JDBCConnectionResource.jdbcResource**.
    c. In JDBC Driver, click **Click Here to Set Preferences**.
    d. Set the JDBC driver folder directory preference to the folder where you have downloaded PostgreSQL JDBC Driver as mentioned in the Prerequisites section. Click **Apply** and then click **OK**.
    e. Click the **Test Connection** button to verify the connection.
6. Click **File > Save**.
7. Fully expand the **Processes** directory and double-click **Books.bwp**.
8. Click **Run > Debug Configurations**.
9. At the left hand tree of Debug Configuration wizard, expand **BusinessWorks Application** and select **BWApplication**.
10. Click **Applications** tab and then click the **Deselect All** button if you have multiple applications. Select the checkbox next to **tibco.bw.sample.binding.rest.BookStore**.
11. Click **Debug**.

The sample runs in Debug mode. The console window shows engine messages similar to: Started BW Application [tibco.bw.sample.binding.rest.BookStore.application:1.0]. The OSGI command to list REST and Swagger URLs is `lrestdoc`, which lists the discovery URL: [Application Name]: tibco.bw.sample.binding.rest.BookStore.application [Discovery Url]: http://localhost:7777/tibco.bw.sample.binding.rest.BookStore.application.

12. Launch Google Chrome browser and open http://localhost:7777/tibco.bw.sample.binding.rest.BookStore.application. The Google Chrome page displays as follows.



13. Click any of the operations, such as POST, GET, PUT, and DELETE displayed on the web page.
14. After you have finished with the operations on the page, in TIBCO Business Studio for BusinessWorks, click the **Terminate** icon to stop the process.

Result

The web page lists the following operations for Books and Events:

Books

- Post books
- GET books

- GET book by ISBN
- PUT book by ISBN
- DELETE book by ISBN
- GET book by Price

Events

- POST Events
- GET Events
- GET Event by EventID
- PUT Event by EventID
- DELETE Event by EventID

GET books returns an output similar to the following:

```
{
  "Book": [
    {
      "isbn": "0061122416",
      "name": "The Alchemist",
      "description": "Every few decades a book is published that changes
the lives of its readers forever. The Alchemist is such a book",
      "authorName": "Paul Coelho",
      "releaseDate": "2006-04-25",
      "vintage": true,
      "signed": true,
      "price": 11.9
    },
    {
      "isbn": "0071450149",
      "name": "The Power to Predict",
      "description": "How Real Time Businesses Anticipate Customer
Needs, Create Opportunities, and Beat the Competition",
      "authorName": "Vivek Ranadive",
      "releaseDate": "2006-01-26",
      "vintage": false,
      "signed": true,
      "price": 15.999
    }
 ]
}
```

GET books by ISBN returns an output similar to the following for the ISBN 0061122416:

```
  {
      "isbn": "0061122416",
      "name": "The Alchemist",
      "description": "Every few decades a book is published that changes
the lives of its readers forever. The Alchemist is such a book",
      "authorName": "Paul Coelho",
      "releaseDate": "2006-04-25",
      "vintage": true,
      "signed": true,
      "price": 11.9
    }
```

The `books.log` file is generated with the following information:

```
POST Books------->{"Book":[{"isbn":"1451648537","name":"Steve
Jobs","description":"Biography of Apple Co-Founder Steve
Jobs","authorName":"Walter Isaacson","releaseDate":"2012-10-
24","vintage":false,"signed":false,"price":21},
{"isbn":"0385537859","name":"Inferno","description":"Robert Langdon
returns in Dan Brown's latest fast paced action
thirller","authorName":"Dan Brown","releaseDate":"2013-05-
14","vintage":false,"signed":true,"price":14.09},
{"isbn":"0399103421","name":"The Godfather","description":"The Godfather
is an epic story of a New York's top mafia family, loyalty, and how men
of honor live in their own world, and die by their own
laws.","authorName":"Mario Puzo","releaseDate":"1969-03-
10","vintage":true,"signed":true,"price":5
0}]}************************************************************

GET Books------->{"Book":[{"isbn":"1451648537","name":"Steve
Jobs","description":"Biography of Apple Co-Founder Steve
Jobs","authorName":"Walter Isaacson","releaseDate":"2012-10-
24+05:30","vintage":false,"signed":false,"price":21},
{"isbn":"0385537859","name":"Inferno","description":"Robert Langdon
returns in Dan Brown's latest fast paced action
thirller","authorName":"Dan Brown","releaseDate":"2013-05-
14+05:30","vintage":false,"signed":true,"price":14.09},
{"isbn":"0399103421","name":"The Godfather","description":"The Godfather
is an epic story of a New York's top mafia family, loyalty, and how men
of honor live in their own world, and die by their own
laws.","authorName":"Mario Puzo","releaseDate":"1969-03-
10+05:30","vintage":true,"signed":true,"price":5
0}]}************************************************************
```

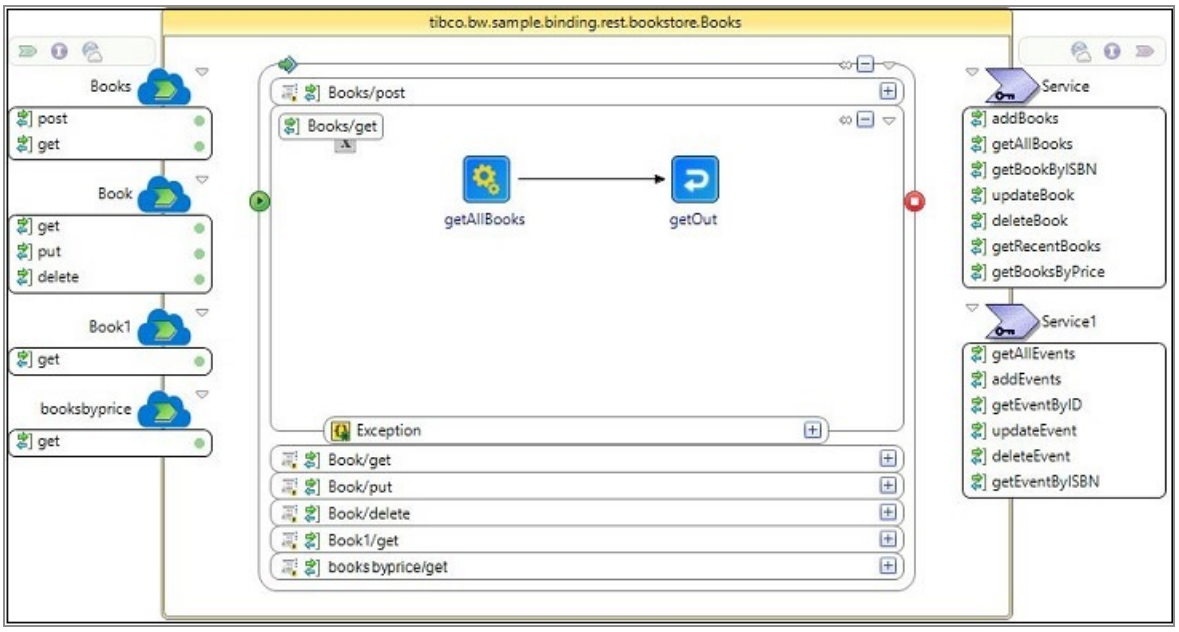
```
GET Book By ISBN------->{"isbn":"1451648537","name":"Steve
Jobs","description":"Biography of Apple Co-Founder Steve
Jobs","authorName":"Walter Isaacson","releaseDate":"2012-10-
24+05:30","vintage":false,"signed":false,"price":2
1}************************************************************

DELETE Book By ISBN-------->"Deleted book with ISBN -
145164853
7"************************************************************

GET Events By ISBN---->
{}************************************************************

GET Books By Price More than 10---->{"Book":
[{"isbn":"0385537859","name":"Inferno","description":"Robert Langdon
returns in Dan Brown's latest fast paced action
thirller","authorName":"Dan Brown","releaseDate":"2013-05-
14+05:30","vintage":false,"signed":true,"price":14.09},
{"isbn":"0399103421","name":"The Godfather","description":"The Godfather
is an epic story of a New York's top mafia family, loyalty, and how men
of honor live in their own world, and die by their own
laws.","authorName":"Mario Puzo","releaseDate":"1969-03-
10+05:30","vintage":true,"signed":true,"price":50},
{"isbn":"0385537829","name":"Angels and Demons","description":"Robert
Langdon returns in Dan Brown's fast paced action
thirller","authorName":"Dan Brown","releaseDate":"2013-05-
14+05:30","vintage":false,"signed":true,"price":15.09},
{"isbn":"0399103439","name":"The Godfather1","description":"The
Godfather is an epic story of a New York's top mafia family, loyalty,
and how men of honor live in their own world, and die by their own
laws.","authorName":"Mario Puzo","releaseDate":"1969-03-
10+05:30","vintage":true,"signed":true,"price":5
0}]}************************************************************

GET Books By Price More than 50---->{"Book":
[null]}*********************************************************
```


Understanding the Configuration

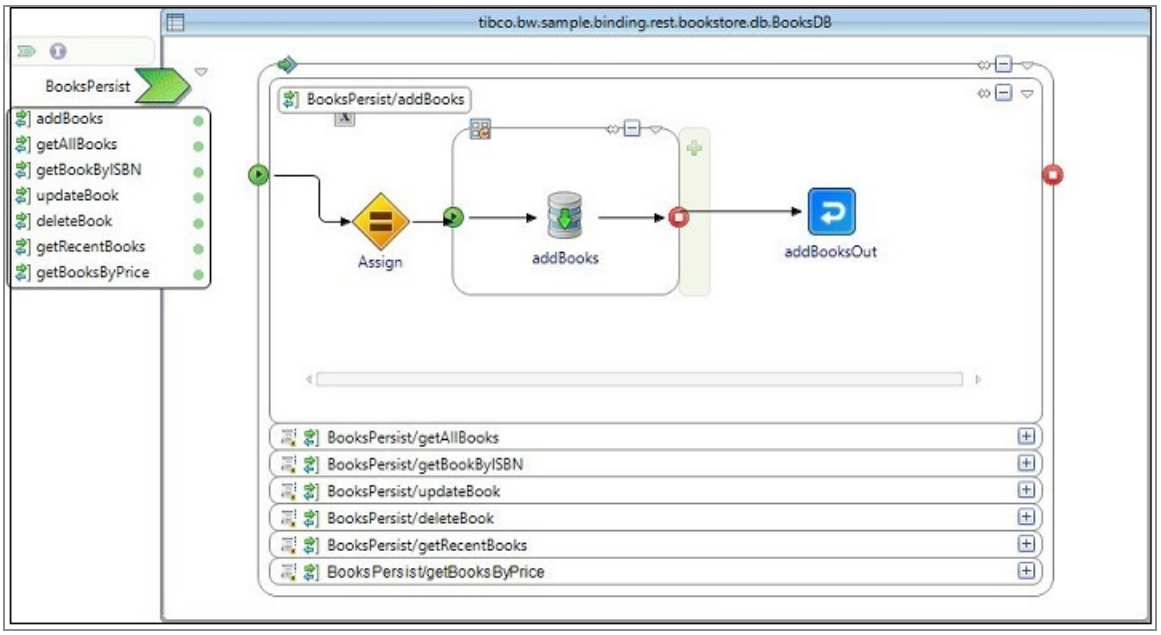
The project contains three processes: Books, Events, Invoke_Client, RESTInvoke.bwp and two subprocesses; BooksDB and EventsDB.

Books Process: In this process, the Book REST service performs the post, get, put, delete operations. It posts one or more books to the bookstore and retrieves all the books from

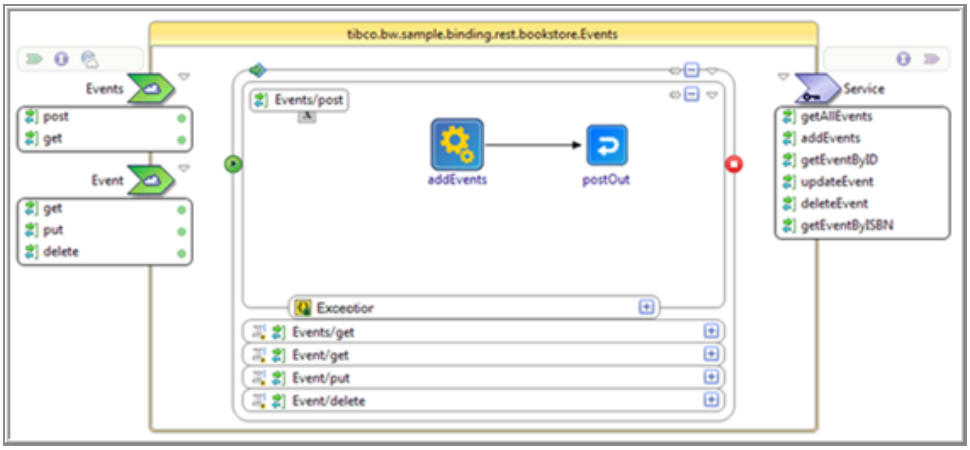
the bookstore. For the post operation, a sub process is called to add books to the database. The addBooks operation is implemented in the BooksDB subprocess. The Book REST service performs three operations: getting a book from the bookstore by ISBN, updating a book in the bookstore using ISBN, and deleting a specific book from the bookstore using ISBN. ISBN is passed as a parameter to these operations. The booksbyprice service gets books from the bookstore using price, which is passed as a query parameter to the GET operation. This service responds with an array of Books if there are books whose price is greater than the provided input (price). If there are no such books, the service returns a Null array.



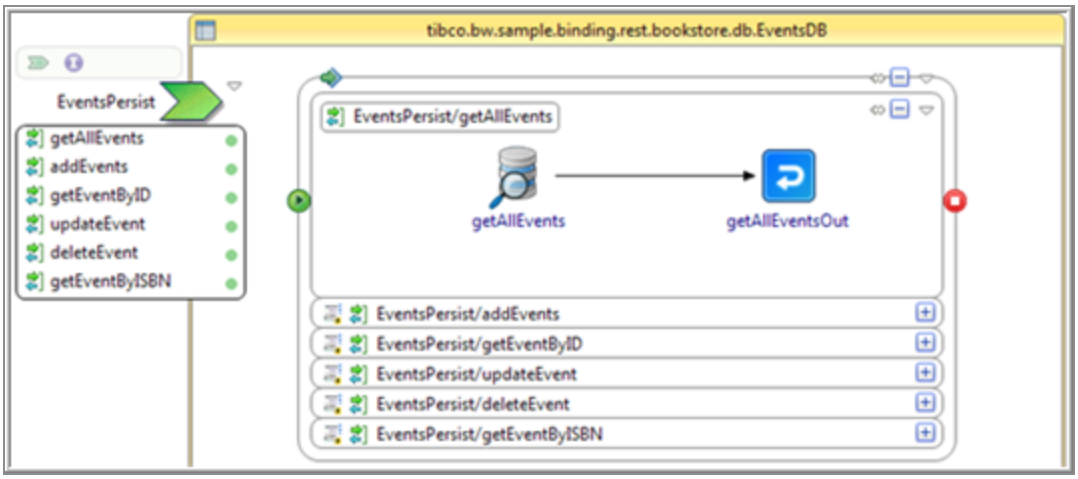
BooksDB Subprocess: The subprocess handles deletes, updates, retrievals of books from the database. The BooksPersist service implements the following operations: adding books to database, getting all books from database, getting a book by ISBN, updating a book by ISBN, deleting a book by ISBN and getting a book by price. Using BooksPersist/addBooks, books are added to the database.



Events Process: It is similar to the Books process and is used to add, delete, update, and retrieve events.

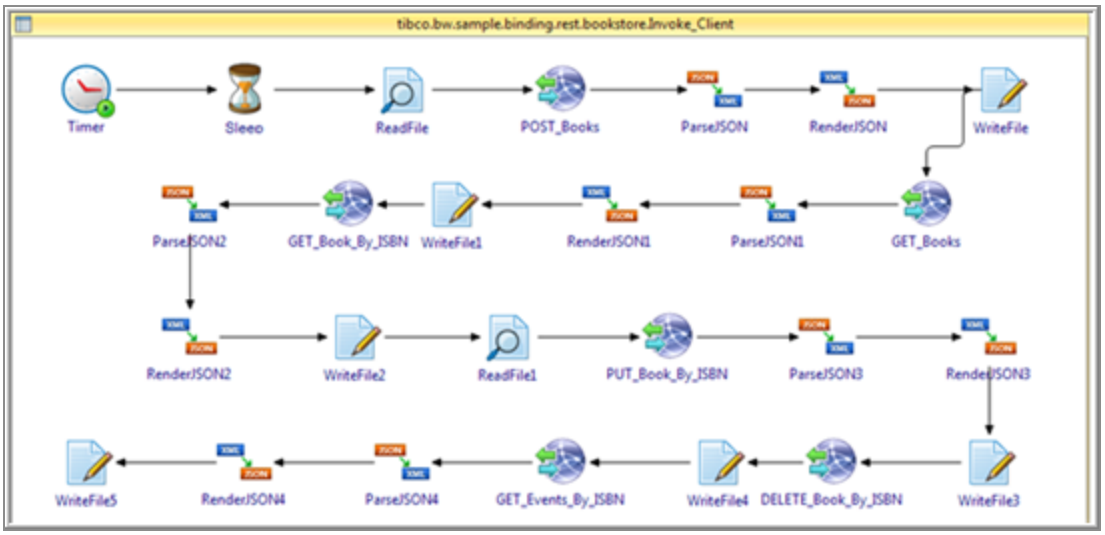


EventsDB Process: It is similar to the BooksDB process and is used to add, delete, update, and retrieve events from the database.

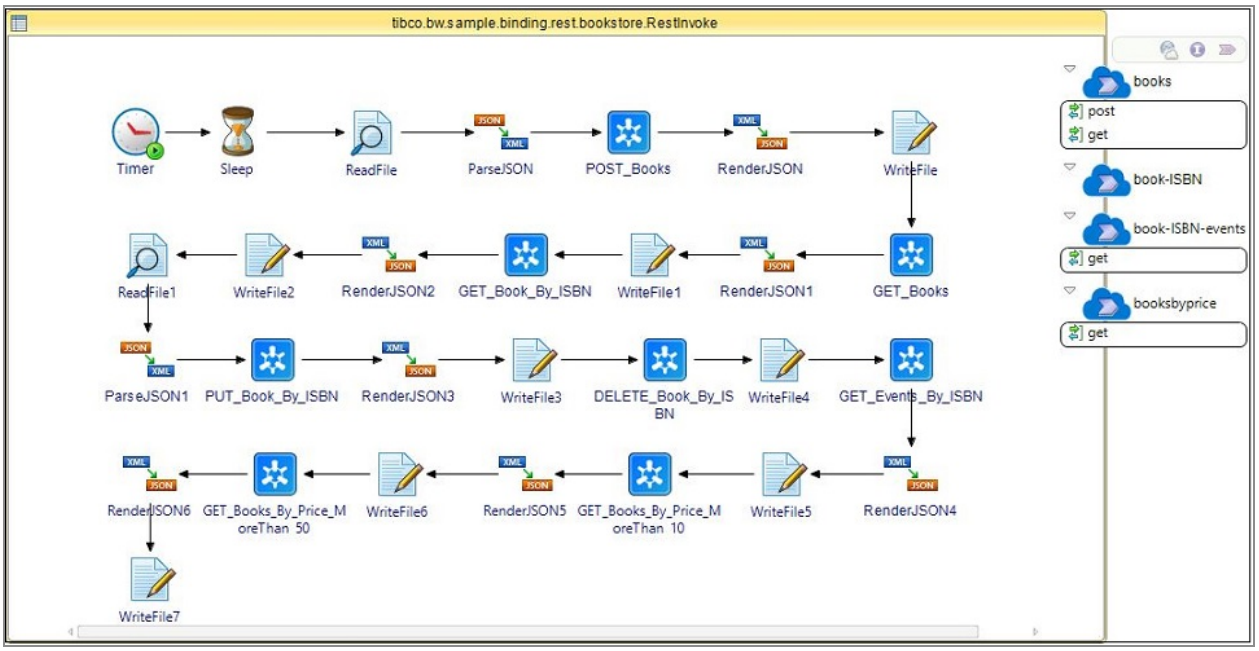


Invoke_Client Process: This is a REST Client process which uses **Invoke REST API** activity to retrieve data from the REST Server. It is similar to Swagger UI. Using this process, you can POST, GET, PUT, and DELETE books by specifying the HTTP Client, Resource Path, HTTP Method, Request Type, and Response Type in the **Invoke REST API** activity.

For example, in POST_Books, we provide HTTP Client Resource which listens to the same host and port as that of server. The Resource Path is "`/books`", HTTP Method is "POST", and Request and Response type is JSON. In the Message Body, we provide the JSON payload which is read from the file present at `BW_HOME\samples\binding\rest\BookStore\samplejson\books.json`. This posts books from the `books.json` payload into the database and you can view the data posted into the output file.

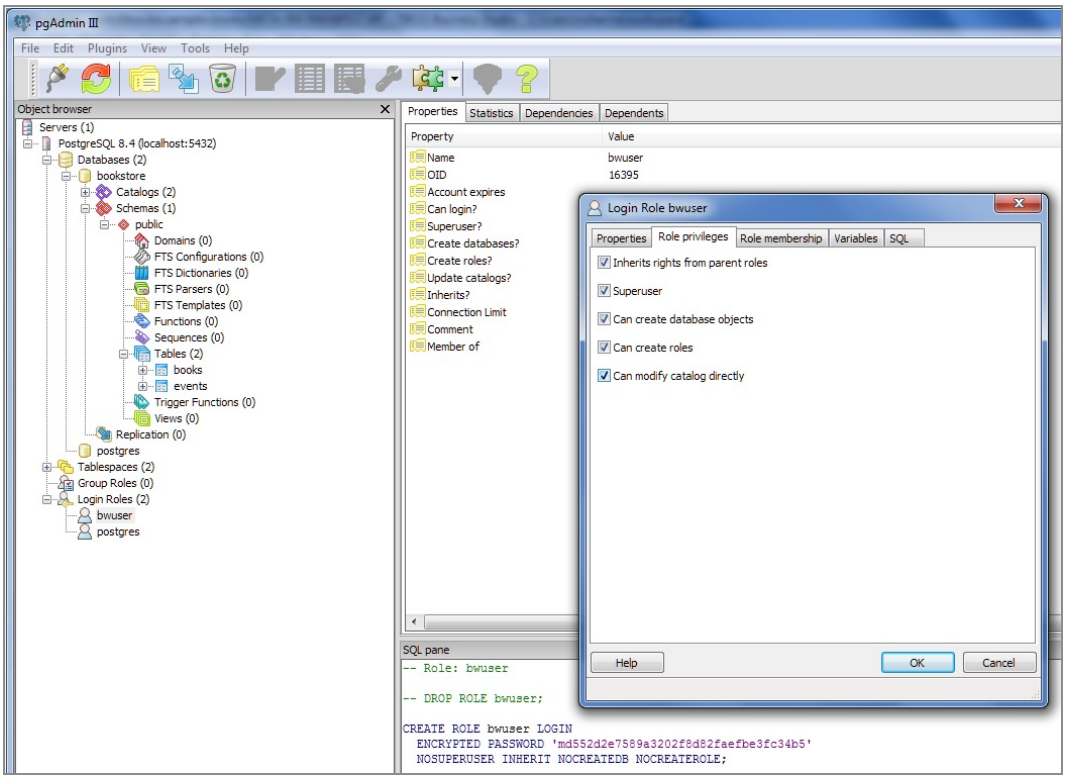


RESTInvoke Process: This is a REST Client process which uses the **REST Invoke** activity to retrieve data from the REST Server. In addition to invoking the existing services, it invokes the service getBooksByPrice which demonstrates null and array responses.



Troubleshooting

- If you are unable to insert rows into the database using the `dbsetup.sql` script in the `scripts` folder, use the sample json payload from the `samplejson` folder to post the data.
- If you do not see the REST Swagger API page, verify whether the application has started.
- If you see any problem markers in the project, clean the project by invoking **Project > Clean** or try switching to a clean new workspace.
- If you get any File was not found exception, verify whether `books.json` and `book_put.json` file is present at the location as specified in the **Input_File** and **Input_File_1 Module Property**.
- If you get an error about unregistered user, open the pgAdmin UI and select all check boxes under the **Role Privileges** tab for the **bwuser**.



Implementing a REST Service Using a JSON File with "allOf" Keyword

This REST sample shows how to use the OpenAPISpecification 3.0 JSON file with "allOf" keyword to create a RESTful service for adding, updating, and retrieving pets data from a pet store.

The sample uses the following HTTP methods for the Pets REST resource:

- GET: Get details of dog from the petstore
- PATCH: Update dog data to the petstore
- POST: Add cat details to the petstore

Implementing a REST Service Using a JSON File with "allOf" Keyword:

Before you begin

Install the latest version of the Google Chrome browser on your system.

Note: If you are accessing the sample from the Welcome page of TIBCO Business Studio for BusinessWorks, navigate to Step 3 directly.

Procedure

1. In the samples directory, select **binding > rest > allOf** and double-click **tibco.bw.sample.binding.rest.allOf**. For more information, see the Accessing Samples section.
2. In the Project Explorer, expand the **tibco.bw.sample.binding.rest.allOf** project.
3. Set the default application profile to match the operating system you are using. For more information, see Setting the Default Application Profile.
4. Fully expand the **Processes** directory and double-click **Pets_Service.bwp**.
5. Click **Run > Debug Configurations**.
6. At the left side pane of the Debug Configuration wizard, expand **BusinessWorks Application** and select **BWApplication**.
7. Click the **Applications** tab and then click the **Deselect All** button, if you have multiple applications. Select the checkbox next to **tibco.bw.sample.binding.rest.allOf.application**.
8. Click **Debug**. This runs the sample in the debug mode.

   The console window shows engine messages similar to:

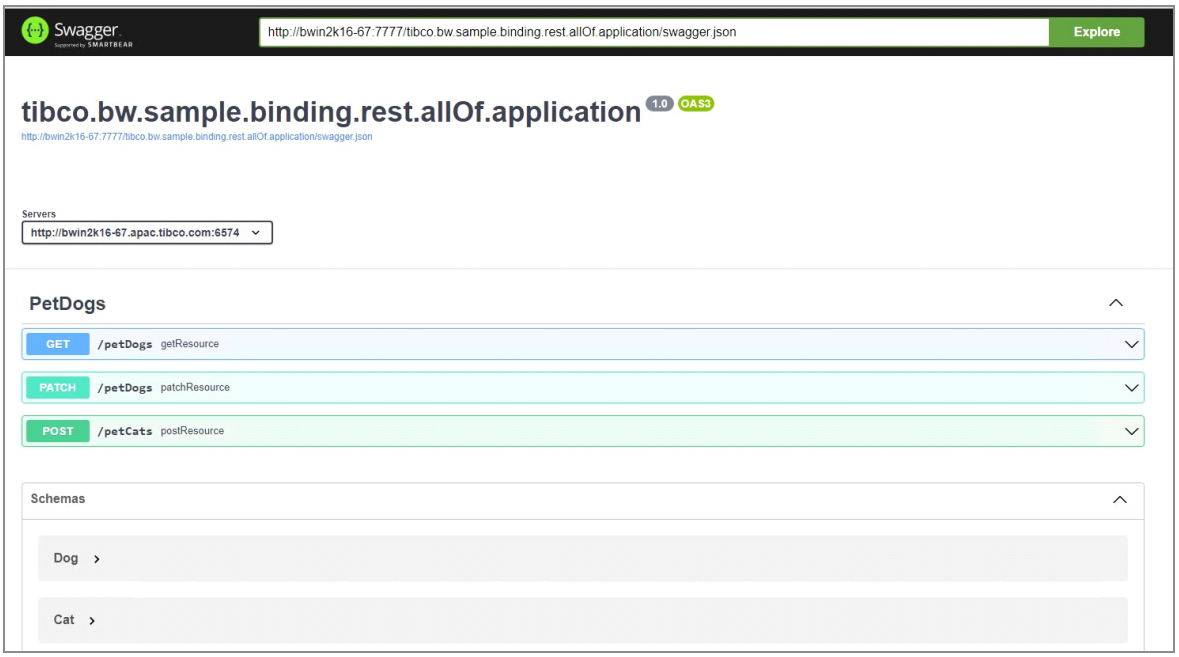
   ```
   Started BW Application
   [tibco.bw.sample.binding.rest.allOf.application:1.0].
   ```

   The OSGI command to list REST and Swagger URLs is **`lrestdoc`**, which lists the following discovery URL:

   ```
   [Application Name]:
   tibco.bw.sample.binding.rest.allOf.application1.0

   [Discovery Url]: http://bwin2k16-
   67:7777/tibco.bw.sample.binding.rest.allOf.application
   ```

9. Launch the Google Chrome browser and open `http://bwin2k16-67:7777/tibco.bw.sample.binding.rest.allOf.application`.

The Swagger API Page displays as follows:



10. Click any of the operations, such as GET, PATCH or POST, displayed on the web page.
11. After completing the operations on the page, in TIBCO Business Studio for BusinessWorks, click the Terminate icon (■) to stop the application.

Result

The web page lists the following operations for dogs and cats:
PetDogs

- GET dog details
- PATCH dog details

PetCats

- POST cat details

GET dogs returns an output similar to the following :

```
{"id":10,"name":"Oreo","status":"Available","bark":false,"breed":"Husk
y"}
```

POST cats returns an output similar to the following:

```
{"id":110,"name":"Kitty","status":"Sold","hunts":true,"age":"12"}
```

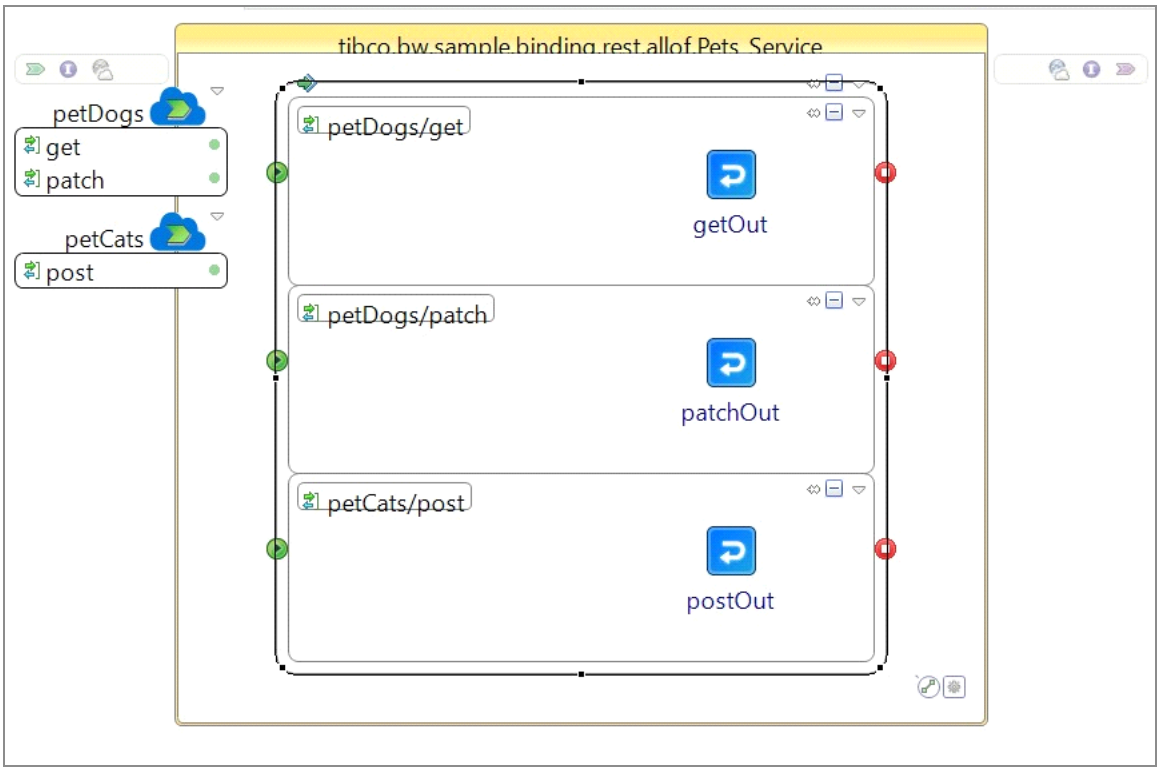
The PetData.log file generated with the following information:

```
**********Output of Pets Data**********

GET Dog having allOf Pet-->
{"id":10,"name":"Oreo","status":"Available","bark":false,"breed":"Husk
y"}************************************************************

PATCH Dog having allOf Pet-->
{"id":20,"name":"Jasper","status":"Sold","bark":true,"breed":"Shepher
d"}************************************************************

POST Cat having allOf Pet-->
{"id":110,"name":"Kitty","status":"Sold","hunts":true,"age":1
2}************************************************************
```

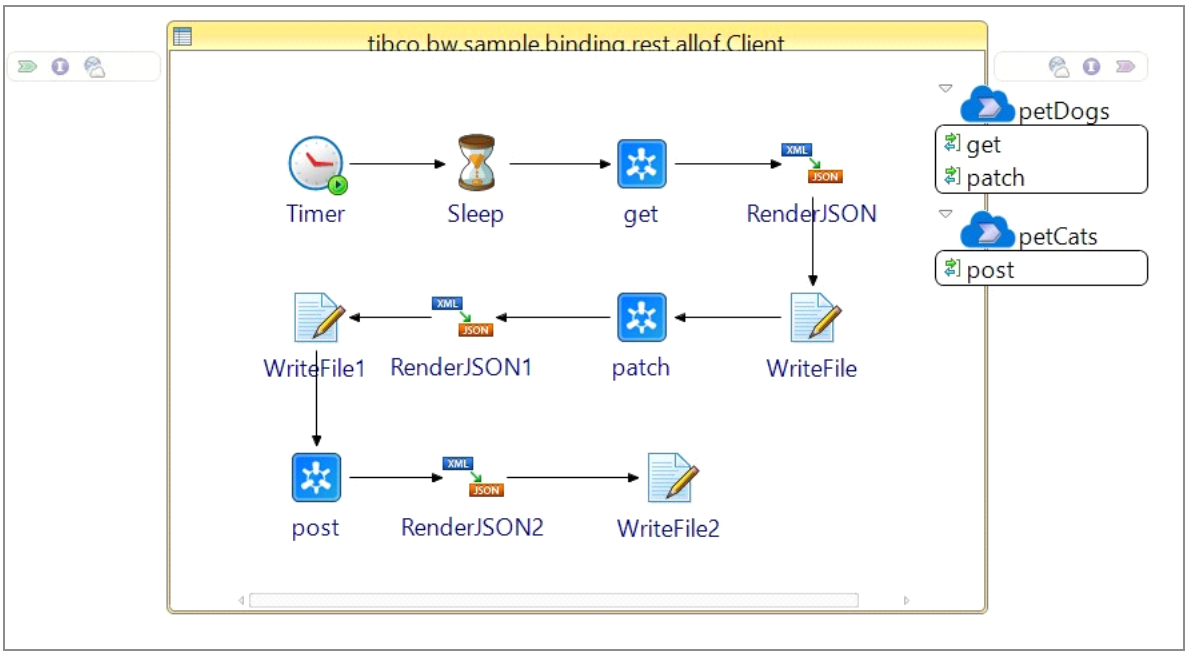
Understanding the Configuration

The project contains two processes, Pets_Service.bwp and Client.bwp.

Pets_Service Process: In this process, the Pets REST service performs the post, get, and patch operations. It updates details of a dog to the petstore and retrieves the dog details from the petstore. The cat details are added using the post operation. This service is created using the Petstore_allOf.json file. The file has Dog, Cat, and Pet objects. The Dog and Cat objects use the "allOf" keyword to combine the main 'Pet' schema with 'Dog-specific' and 'Cat-specific' properties respectively.



Client Process: This is a REST Client process, which uses the REST Invoke activity to retrieve data from the REST Server.



Troubleshooting

- If you do not see the REST Swagger API page, verify whether the application has started.
- If you see any problem markers in the project, clean the project by clicking **Project > Clean** or try switching to a clean new workspace.

Implementing a REST Service Using a JSON File with "anyOf" Keyword

This REST sample shows how to use the OpenAPISpecification 3.0 JSON file with "anyOf" keyword to create a RESTful service for adding pets data from a pet store.

The sample uses the following HTTP **POST** method for the Pets REST resource:

Before you begin

Install the latest version of the Google Chrome browser on your system.

> **Note:** If you are accessing the sample from the Welcome page of TIBCO Business Studio for BusinessWorks, navigate to Step 3 directly.

Procedure

1. In the samples directory, select **binding > rest > anyOf** and double-click **tibco.bw.sample.binding.rest.anyOf**. For more information, see the Accessing Samples section.
2. In the Project Explorer, expand the **tibco.bw.sample.binding.rest.anyOf** project.
3. Set the default application profile to match the operating system you are using. For more information, see Setting the Default Application Profile.
4. Fully expand the **Processes** directory and double-click **PetsService.bwp**.
5. Click **Run > Debug Configurations**.
6. At the left side pane of the Debug Configuration wizard, expand **BusinessWorks**

Application and select **BWApplication**.

7. Click the **Applications** tab and then click the **Deselect All** button, if you have multiple applications. Select the checkbox next to **tibco.bw.sample.binding.rest.anyOf.application**.
8. Click **Debug**. This runs the sample in the debug mode.

   The console window shows engine messages similar to:

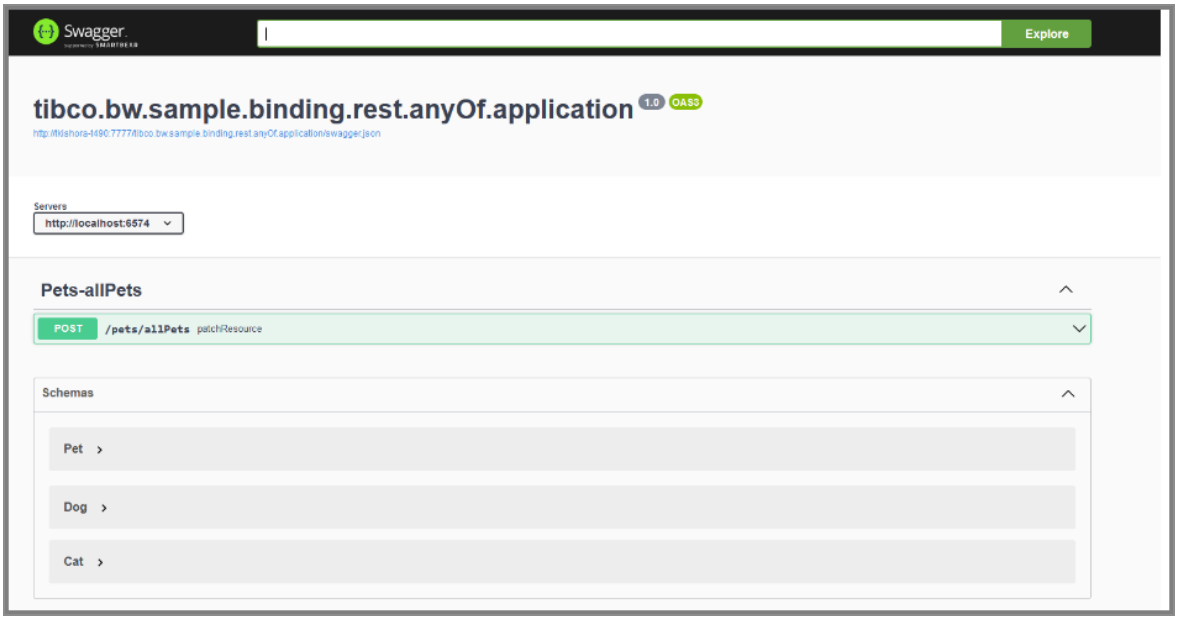
   ```
   Started BW Application
   [tibco.bw.sample.binding.rest.anyOf.application:1.0].
   ```

   The OSGI command to list REST and Swagger URLs is **`lrestdoc`**, which lists the following discovery URL:

   ```
   [Application Name]:
   tibco.bw.sample.binding.rest.anyOf.application1.0

   [Discovery Url]: http://bwin2k16-
   67:7777/tibco.bw.sample.binding.rest.anyOf.application
   ```

9. Launch the Google Chrome browser and open `http://bwin2k16-67:7777/tibco.bw.sample.binding.rest.anyOf.application`.

   The Swagger API Page displays as follows:



10. Click any of the operations, such as GET, PATCH or POST, displayed on the web page.
11. After completing the operations on the page, in TIBCO Business Studio for BusinessWorks, click the Terminate icon ( ) to stop the application.

Result

The web page lists the following operations for dogs and cats:
Pets-allPets

- POST pets details

POST pets return an output similar to the following :

```
{"name": "Oliver","bark": true,"breed": "Shepherd","id": "CT001","age":
6}
```

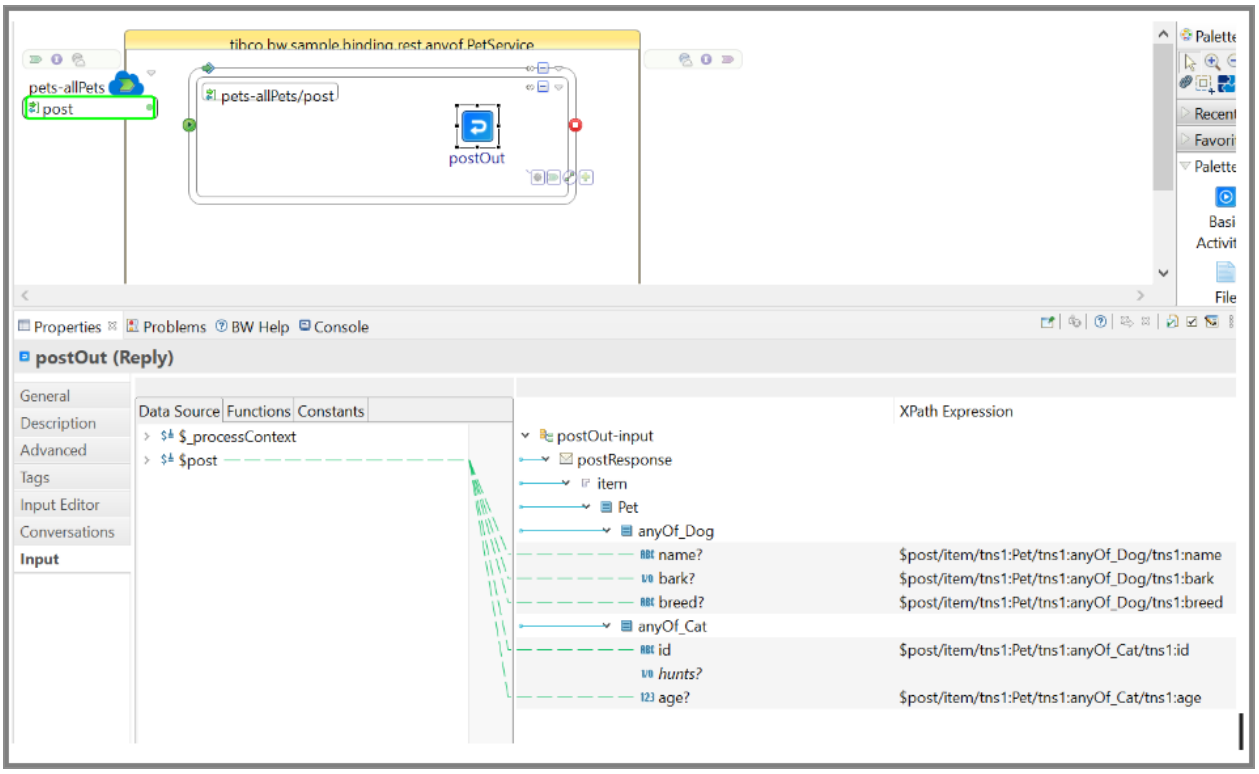
The PetData.log file generated with the following information:

```
***********Output of Pets Data***********
POST having anyOf Pet→ {"anyOf_Dog":
{"name":"Oliver","bark":true,"breed":"Husky"},"anyOf_Cat":
{"id":"CT0014","age":4}}
************************************************************
```

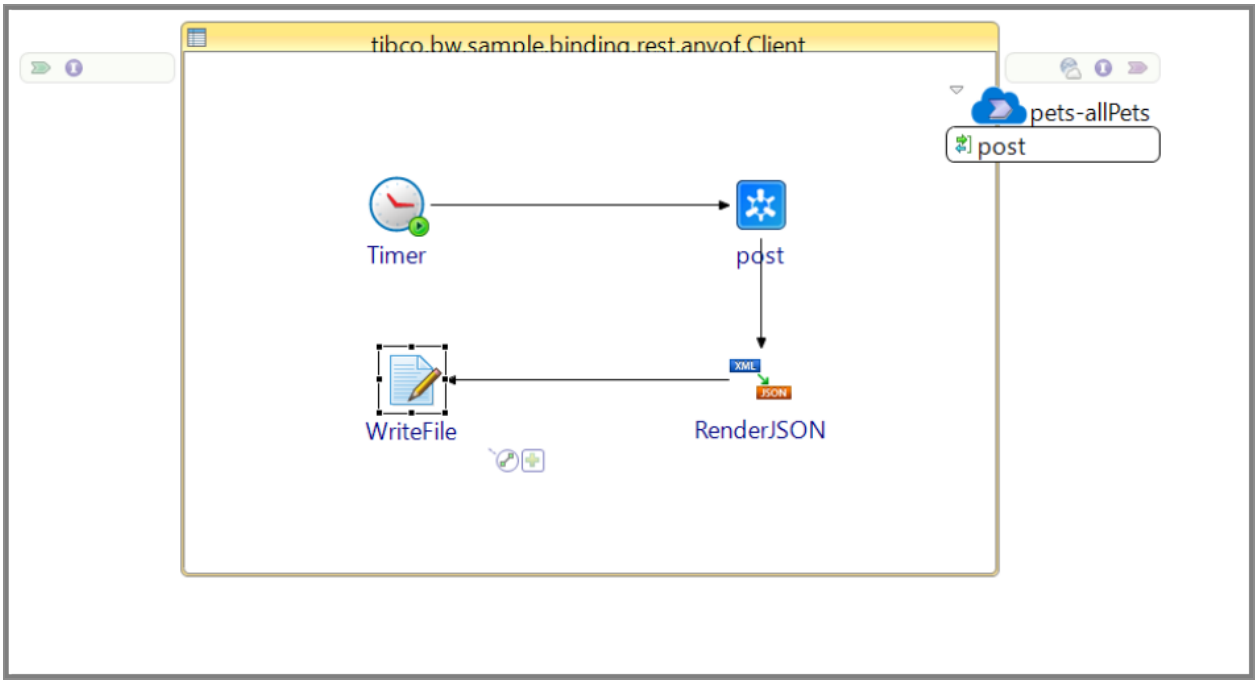
Understanding the Configuration

The project contains two processes, PetsService.bwp and Client.bwp.

PetsService Process: In this process, the Pets REST service performs the post operation. It adds the details of pets to the petstore and responds with the Pets details. This service is created using the Petstore_anyOf.json file. The file has Dog, Cat, and Pet objects. The Pet objects use the "anyOf '' keyword to validate against either the 'Dog' or the 'Cat' schema or both the schemas depending on the configuration provided on the Input tab of the postOut (reply) activity.



Client Process: This is a REST Client process, which uses the REST Invoke activity to call the REST Server. The request can contain a combination of properties which are valid against either one or both the Dog and Cat sub-schemas.



Troubleshooting

- If you do not see the REST Swagger API page, verify whether the application has started.
- If you see any problem markers in the project, clean the project by clicking **Project > Clean** or try switching to a clean new workspace.

Implementing a REST Service Using a JSON File with "oneOf" Keyword

This REST sample shows how to use the OpenAPISpecification 3.0 JSON file with "oneOf" keyword to create a RESTful service for updating pets data from a pet store.

The sample uses the HTTP **PATCH** method for the Pets REST resource.

Before you begin

Install the latest version of the Google Chrome browser on your system.

> **Note:** If you are accessing the sample from the Welcome page of TIBCO Business Studio for BusinessWorks, navigate to Step 3 directly.

Procedure

1. In the samples directory, select **binding > rest > oneOf** and double-click **tibco.bw.sample.binding.rest.oneOf**. For more information, see the Accessing Samples section.
2. In the Project Explorer, expand the **tibco.bw.sample.binding.rest.oneOf** project.
3. Set the default application profile to match the operating system you are using. For more information, see Setting the Default Application Profile.
4. Fully expand the **Processes** directory and double-click **PetsService.bwp**.
5. Click **Run > Debug Configurations**.
6. At the left side pane of the Debug Configuration wizard, expand **BusinessWorks**

Application and select **BWApplication**.

7. Click the **Applications** tab and then click the **Deselect All** button, if you have multiple applications. Select the checkbox next to **tibco.bw.sample.binding.rest.oneOf.application**.
8. Click **Debug**. This runs the sample in the debug mode.

   The console window shows engine messages similar to:

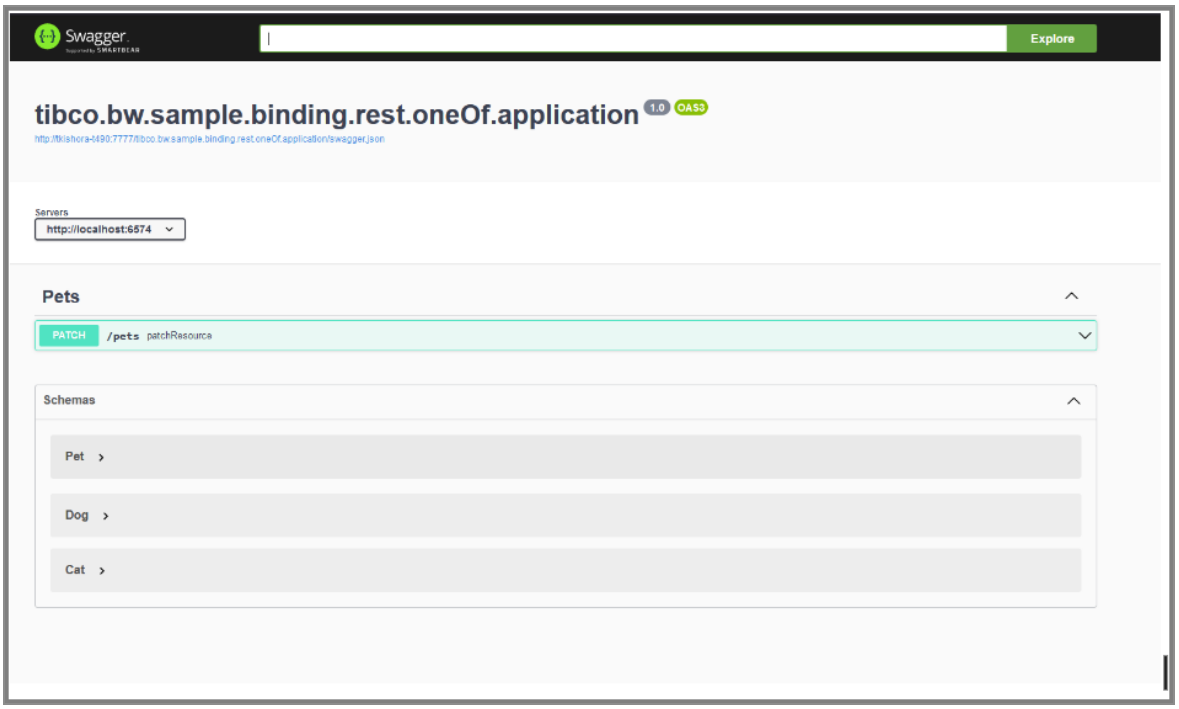
   ```
   Started BW Application
   [tibco.bw.sample.binding.rest.oneOf.application:1.0].
   ```

   The OSGI command to list REST and Swagger URLs is **`lrestdoc`**, which lists the following discovery URL:

   ```
   [Application Name]:
   tibco.bw.sample.binding.rest.oneOf.application1.0

   [Discovery Url]: http://bwin2k16-
   67:7777/tibco.bw.sample.binding.rest.oneOf.application
   ```

9. Launch the Google Chrome browser and open `http://bwin2k16-67:7777/tibco.bw.sample.binding.rest.oneOf.application`.

   The Swagger API Page displays as follows:



10. Click the PATCH operation displayed on the web page.
11. After completing the operations on the page, in TIBCO Business Studio for BusinessWorks, click the Terminate icon (■) to stop the application.

Result

The web page lists the following operations for dogs and cats:
Pets

- PATCH pets details

PATCH pets return an output similar to the following

```
{"name":"Rusty","bark":true,"breed":"Shepherd"}
```

The PetData.log file generated with the following information:

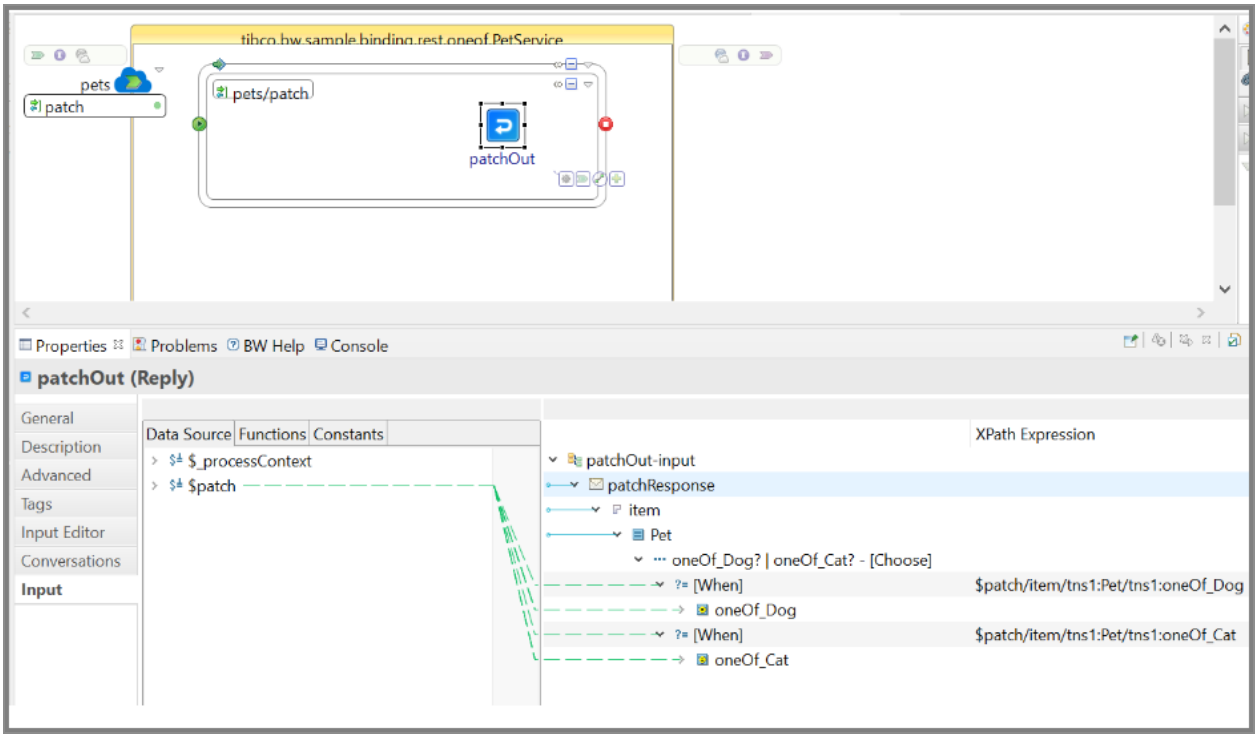
```
**********Output of Pets Data**********

PATCH Pet having oneOf-->
{"oneOf_Dog":
{"name":"Rusty","bark":true,"breed":"Shepher
d"}}**********************************************************
```

```
**********Output of Pets Data**********

PATCH Pet having oneOf-->
{"oneOf_Cat":
{"id":"CT011","hunts":false,"age":
3}}**********************************************************
```

Understanding the Configuration

The project contains two processes, PetsService.bwp and Client.bwp.

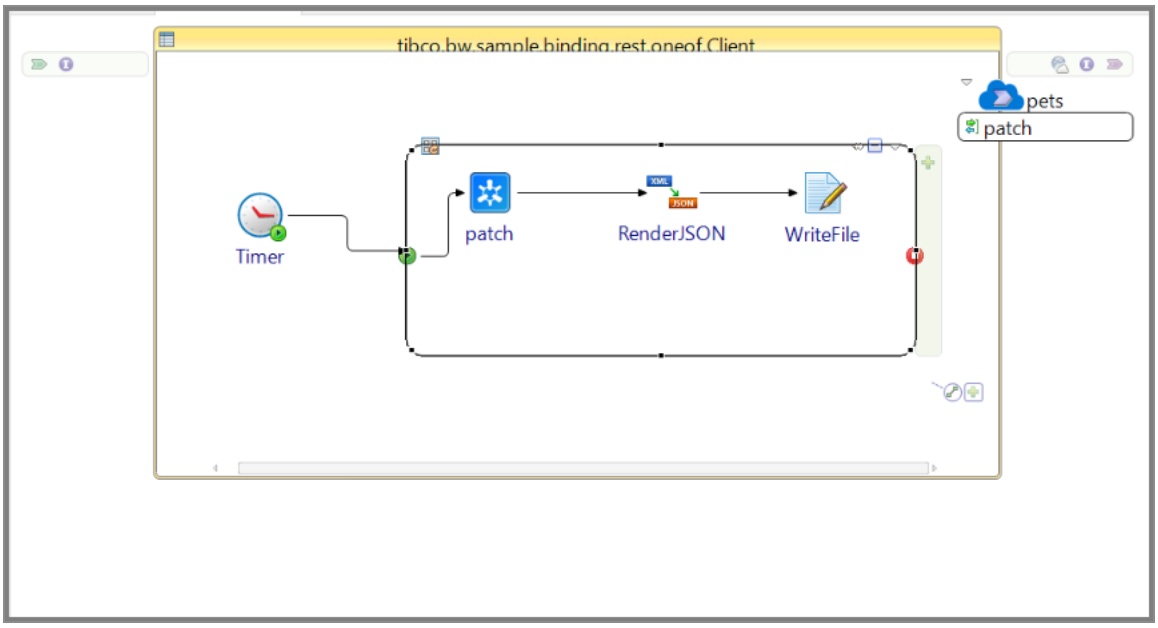
PetsService Process: In this process, the Pets REST service performs the patch operation. It updates the details of the pet to the petstore and retrieves the dog details from the petstore. This service is created using the Petstore_oneOf.json file. The file has Dog, Cat, and Pet objects. The Pet objects use the "oneOf '' keyword to validate against either the 'Dog' or the 'Cat' schema depending on the condition provided on the Input tab of the patchOut (reply) activity.



Client Process: This is a REST Client process, which uses the REST Invoke activity to call the REST Server.

Process contains a ForEach loop to demonstrate that the change in the mapping to input schemas (Either Dog Or Cat) will affect the response received from the server.

The first iteration of the loop will send the input which is of Type Dog. The second iteration of the loop will send the input of type Cat.



Troubleshooting

- If you do not see the REST Swagger API page, verify whether the application has started.
- If you see any problem markers in the project, clean the project by clicking **Project > Clean** or try switching to a clean new workspace.

Implementing Multiple Response based REST Service using a JSON file

This REST Sample shows how to use the OAS 3.0 JSON file to create a RESTful service which has operations configured with multiple responses.

The sample uses the POST HTTP method for the REST resource.

Before you begin

Install the latest version of the Google Chrome verison in the standard default location

Note: If you are accessing the sample from the Welcome page of TIBCO Business Studio for BusinessWorks, navigate to Step 3 directly.

Procedure

1. In the samples directory, select **binding > rest > MultipleResponse** and double-click **tibco.bw.sample.binding.rest.MultipleResponse**. For more information, see the Accessing Samples section.
2. In the Project Explorer, expand the **tibco.bw.sample.binding.rest.MultipleResponse** project.
3. Set the default application profile to match the operating system you are using. For more information, see Setting the Default Application Profile.
4. Fully expand the **Processes** directory and double-click **Service.bwp**.
5. Click **Run > Debug Configurations**.
6. At the left side pane of the Debug Configuration wizard, expand **BusinessWorks Application** and select **BWApplication**.
7. Click the **Applications** tab and then click the **Deselect All** button, if you have multiple applications. Select the checkbox next to **tibco.bw.sample.binding.rest.MultipleResponse.application**.
8. Click **Debug**. This runs the sample in the debug mode.

   The console window shows engine messages similar to:

   ```
   Started BW Application
   [tibco.bw.sample.binding.rest.MultipleResponse.application:1.0].
   ```

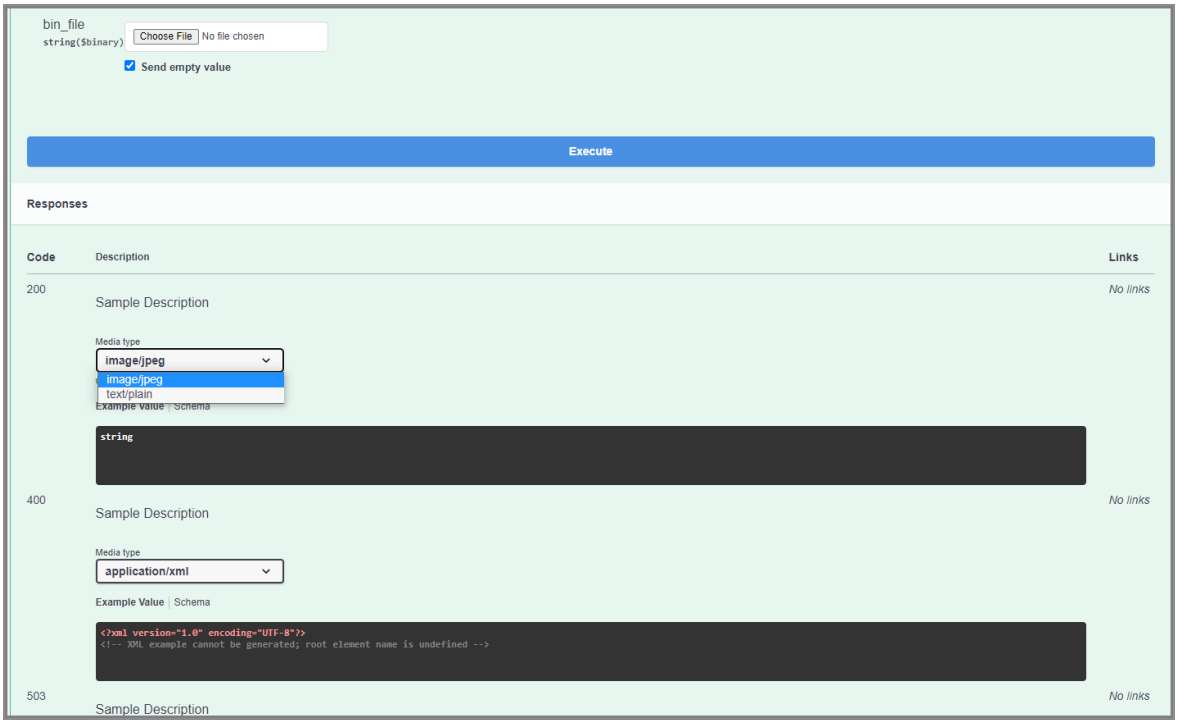
   The OSGI command to list REST and Swagger URLs is **`lrestdoc`**, which lists the following discovery URL:

   ```
   [Application Name]:
   tibco.bw.sample.binding.rest.MultipleResponse.application1.0

   [Discovery Url]: http://TIBCO-
   PF372WXP:7777/tibco.bw.sample.binding.rest.MultipleResponse.applica
   tion
   ```

9. Launch the Google Chrome browser and open `http://TIBCO-`

`PF372WXP:7777/tibco.bw.sample.binding.rest.MultipleResponse.application..`

The Swagger API Page displays as follows:



10. Click the POST operation displayed on the web page.
11. After completing the operations on the page, in TIBCO Business Studio for BusinessWorks, click the Terminate icon (🟥) to stop the application.

Result

The web page lists the following response Media Types for POST operation:

- 200 Status Code with an image/jpeg and text/plain format.
- 400 Status Code with application/xml format.
- 503 Status Code with application/json format.

We have to select the Response Media Type from the dropdown for the 200 status code.

On selecting **image/jpeg**, we can receive the image file which was sent in the request as a form parameter. The Image file is displayed in the Swagger UI.

On selecting text/plain, we receive a string response similar to the following :

`Successful response with simple string`

For fault status codes, we have to use the Postman client, since the Accept headers can be controlled only by the 200 response media types in the Swagger UI.

In Postman we can pass the **Accept** header value corresponding to the content-type which we are expecting in response. In our sample, if we want to receive a response in XML format, we can pass the Accept header value as application/xml or application/json for JSON format.

- For application/json Accept header we receive a response similar to the following :

```
{"msgCode":503,"message":"Replying with custom fault as file not
found."}
```

- For application/xml Accept header we receive a response similar to the following :

```
<client-400-application-xml>Sending 400 fault as file not
found.</client-400-application-xml>
```

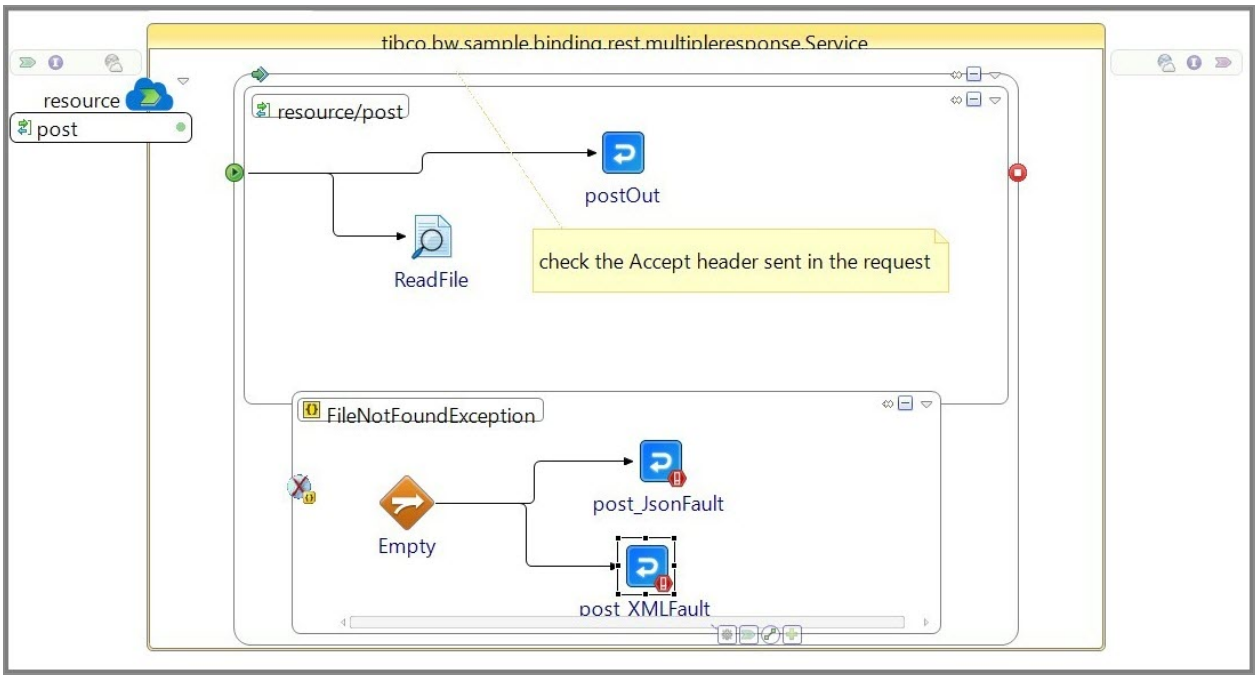
The output.log file is generated with the following information :

```
**********Multiple Response Output**********
Output of POST with text/plain Content-Type --> Successful response with
simple string************
Caught 503 fault with application/json Content-Type -->
{"msgCode":503,"message":"Replying with custom fault as file not
found."}************
Caught 400 fault with application/xml Content-Type --> Sending 400 fault
as file not found.************
```

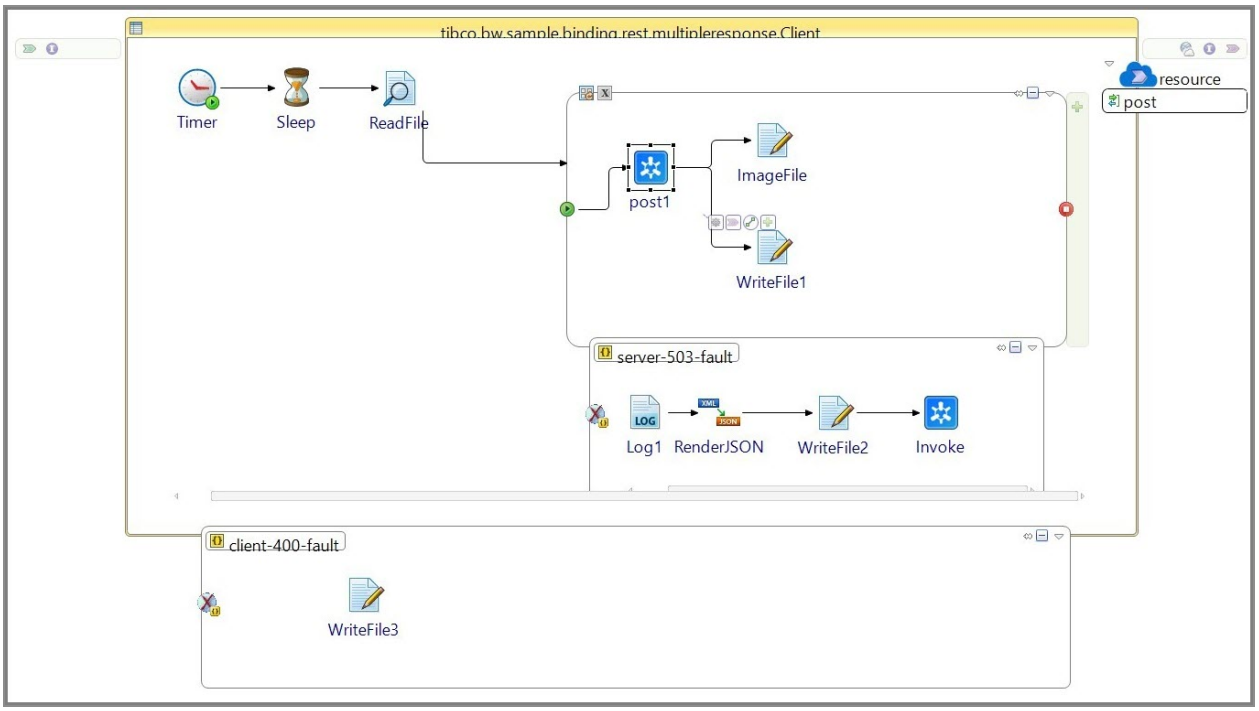
Understanding the Configuration

The project contains two processes: Service and Client.bwp.

Service Process : In this process, the REST service performs the post operation. It uploads an image file and fetches it in response for image/jpeg content-type. This service is created using the MultipleResponse_Swagger.json file. The response to be sent to the client is decided if the condition on Transition is satisfied. The condition checks the Accept header sent by the client. If it matches image/jpeg or text/plain then service responds with the 200 status code. If this condition is not satisfied i.e we pass application/json or application/xml Accept header, then the invalid file is not found and the corresponding fault reply is sent from the Catch block based on transition check for Accept header.



Client Process : This is a REST Client process which uses the REST Invoke activity to retrieve data from the REST Server. An image file is uploaded using the form parameter and the Accept header value is changed on each iteration of the For Each loop. The Accept header takes the value from the loop Variables. Once the 503 fault is caught in the Catch block, we have an Invoke which passes the application/xml Accept header and its corresponding 400 fault is caught by another Catch block.



Troubleshooting

- If you do not see the REST Swagger API page, verify whether the application has started.
- If you see any problem markers in the project, clean the project by clicking **Project > Clean** or try switching to a clean new workspace.
- If you get any File was not found exception, verify whether the `input_image.jpg` file is present at the location as described in the Input_file Module Property.

Using a SOAP Web Service to Make a Query to a Bookstore

Using a SOAP binding, a web service WSDL operation queries for information from a data source. The web service has one operation, GetBooksByAuthor, that is implemented in the process, QueryBooksByAuthorProvider. At run-time, a request containing an author name returns details about the author's books.

Before you begin

Copy the `BookStore.xml` file to `C:\tmp` location.

Procedure

1. In the **samples** directory, select **binding > soap > http > BookStore** and double-click **tibco.bw.sample.binding.soap.http.BookStore**. For more information, see Accessing Samples.
2. In **Project Explorer**, expand the **tibco.bw.sample.binding.soap.http.BookStore** project.
3. Fully expand the **Processes** directory and double-click **QueryBooksByAuthorProvider.bwp**.
4. Click **Run > Debug Configurations**.
5. At the left hand tree of Debug Configuration wizard, expand **BusinessWorks Application** and select **BWApplication**.
6. Click the **Applications** tab and then click the **Deselect All** button if you have multiple

applications.

7. Select the checkbox next to **tibco.bw.sample.binding.soap.http.BookStore.BookStore.application**.
8. Click **Debug**.

   This runs the sample in Debug mode.
9. Click the **Terminate** icon to stop the process.

Result

The query sends a request for "Vivek Ranadive" and returns the following book details:

17:58:24.807 [PVM:In-Memory STWorkProcessor:2] INFO c.t.b.g.L.t.b.b.s.h.BookStore.Log - Message=Book Title: The Power of Now, Author is Vivek Ranadive, Date: 1999, ISBN 0-06-566778-9, Publisher is Tibco Software Inc

Understanding the Configuration

The following processes are defined in the project:

- **BookStoreServiceClient.bwp**: The process implements the GetBooksByAuthor operation using `BooksInterface.wsdl`, and then invokes the QueryBooksByAuthorProvider process. The reply containing the query results is made by mapping to the output of the Query service. A **Timer** activity triggers the client and then triggers a **SOAP Invoke** activity, which is the actual web service client.
- **QueryBooksByAuthorProvider.bwp**: The process definition representing the service's business logic. The service **Receive** activity implements the GetBooksByAuthor operation based on the `BooksInterface.wsdl` file. The **Read File** activity reads the bookstore (an XML file in this sample).

  > **Note:** The file must be on the machine that is used to run the sample.

  The **Parse XML** activity then parses the XML document. Finally, the filter is executed at the end of the process flow in the **Reply** (see Reply activity's Input tab).

The following resources are defined in the project:

- `Resources\BookStore.xml`. The file representing the bookstore database.
- **HTTPConnector.httpConnResource**: The HTTP shared resource to use. If you want

the web service to access a different host machine or port, edit the Host and Port properties, and then regenerate the concrete WSDL from the component binding.

For more information, see *TIBCO ActiveMatrix BusinessWorks™ Application Development*.

The following schemas and service descriptors are defined in the project:

- `Schemas\Books.xsd`: The schema file containing the `type` definitions used by the web service interface.
- `Service Descriptors\BooksService.wsdl`: The WSDL file that describes the Bookstore web service.

Reconfiguring a Web Service by Changing the Transport

This sample documentation describes the way to convert a transport for a Web Service

Before you begin

TIBCO Enterprise Message Service™ must be running.

Sample Description

Converting a transport for a Web Service:

- From SOAP over HTTP to SOAP over JMS
- From SOAP over JMS to SOAP over HTTP

For more information, see Accessing Samples.

Before performing the following steps, import the `SOAP over HTTP ZipCodeServiceProvider.zip` project from the **File Explorer** view by double-clicking the project zip file in TIBCO Business Studio™ for BusinessWorks™.

Converting the Transport from SOAP over HTTP to SOAP over JMS

Procedure

1. Create and configure a **JMS Connection** shared configuration resource.
2. Navigate to the **Module Descriptors > Components** tab and select **ZipPort** service.
3. Click the **Bindings** tab, and in the **Transport Configuration** section, change the **Transport Type** from **HTTP** to **JMS**.
4. Create the JMS Connection shared resource with default or custom values using the JMS Connection wizard after changing the **Transport Type** to **JMS**.
5. Change the **Messaging Style** to **Queue** in the **JMS Connection** shared resource.
6. Create **JNDI Configuration** shared resource with default or custom values.
7. Test the connection using the **Test Connection** button.
8. Click **Bindings > Transport Configuration** and specify the **JMS Destination** value as `queue.sample` or a custom queue name.
9. In the TIBCO Enterprise Message Service server, create a queue with `queue.sample` name, if a queue was not created earlier, using the TIBCO Enterprise Message Service Administration option.
10. Click the **Generate Concrete WSDL for SOAPServiceBinding binding** link in the Binding configuration. Use the Generate WSDL wizard to generate a new concrete WSDL.
11. Delete the previous concrete WSDL, to avoid WSDL cache exceptions, located under **Service Descriptors > Special Folders** that contains HTTP transport details.
12. Specify the folder location where you want to generate the concrete WSDL file.
13. Clear the Embed options (**Abstract WSDL** and **Schema**) check boxes, if both Service and Client are under the same project module.
14. Click **OK** in the WSDL generated successfully confirmation dialog.
15. Click **Transport Connection** to test the connection status.

Result

The Client and Server are executed.

Converting the Transport from SOAP over JMS to SOAP over HTTP

Before performing the following steps, import the `ZipCodeLookup.zip` project from the **File Explorer** view by double-clicking the project zip in TIBCO Business Studio for BusinessWorks.

Procedure

1. Create an HTTP Connection shared configuration resource, and configure the HTTP Host and Port details.
2. Navigate to the **Module Descriptors > Components** tab and select **ZipPort** service.
3. Click the **Bindings** tab, and in the **Transport Configuration** section, change the **Transport Type** from **JMS** to **HTTP**.
4. Create the HTTP Connection shared resource with default or custom values using the HTTP Connection wizard after changing the **Transport Type** to **HTTP**.
5. Click the **Generate WSDL for SOAPReferenceBinding binding** link in the Binding configuration. Use the Generate WSDL wizard to generate a new concrete WSDL.
6. Clear the Embed options (**Abstract WSDL** and **Schema**) check boxes, if both Service and Client are under same project module.

   If required, overwrite with the existing Client concrete WSDL file.
7. Click **OK** in the WSDL generated successfully confirmation dialog.
8. Navigate to **Module Descriptors > Components** tab and select **ZipPort Reference**.
9. Delete the previous SOAPReferenceBinding configured with JMS transport.
10. Optionally, configure the HTTP Client shared resource to override the custom configurations.
11. To avoid WSDL cache exceptions, delete the previous concrete WSDL located under **Service Descriptors > Special Folders** that contains JMS transport details.

Important: If the previous concrete WSDL is not removed or deleted, you see two portTypes. Choose the default selected portType, which is HTTP.

Note: If you are using Context mappings on Client binding, you must reconfigure the Context mappings.

Result: The Client and Server are executed successfully.

Using a SOAP Connection to Publish Messages to Multiple Endpoints

This sample shows how to publish messages to multiple endpoints over HTTP and JMS.

Before you begin

TIBCO Enterprise Message Service must be running.

Procedure

1. In the **samples** directory, select **binding > soap > http > HTTPJMSMultiEndpoint** and double-click **tibco.bw.sample.binding.soap.http.HTTPJMSMultiEndpoint**. For more information, see Accessing Samples.
2. In **Project Explorer** expand the **tibco.bw.sample.binding.soap.http.HTTPJMSMultiEndpoint** project.
3. Verify your TIBCO Enterprise Message Service connection. To verify:
    a. Fully expand the **Resources** directory.
    b. Double-click **JMSConnectionResource.jmsConnResource**.
    c. Click the **Test Connection** button to verify the connection.
4. Fully expand the **Processes** directory and double-click **TestService.bwp**.
5. Click **Run > Debug Configurations**.
6. At the left hand tree of the Debug Configuration wizard, expand **BusinessWorks Application** and select **BWApplication**.
7. Click the **Applications** tab and then click the **Deselect All** button if you have multiple applications. Select the checkbox next to **tibco.bw.sample.binding.soap.http.HTTPJMSMultiEndpoint.application**.

8.
9. Click **Debug**.

   This runs the sample in Debug mode.
10. Click the **Terminate** icon to stop the process.

Result

The file provided for the OUTPUT_FILE property is written with the following content:

JMS : Output of getCityInfo : 61801 Urbana Illinois Urbana, Illinois, United States 40.11 88.207

JMS : Output of getCityDistance : Distance between two cities is : 4 miles. 61801 61820 Urbana Champaign Illinois Illinois Urbana, Illinois, United States Champaign, Illinois, United States

HTTP : Output of getCityInfo : 61801 Urbana Illinois Urbana, Illinois, United States 40.11 88.207

HTTP : Output of getCityDistance : Distance between two cities is : 4 miles. 61801 61820 Urbana Champaign Illinois Illinois Urbana, Illinois, United States Champaign, Illinois, United States

Sending and Receiving a SOAP Bound Attachment

A client process sends Text and GIF files as attachments for the In-Band attachments to the server process. The server process receives these attachments, and returns them as its response. The client process captures the sent and received attachments' size.

Note: It is a **Bound Attachment** when one part of the Input or Output WSDL message is of type "**base64binary**" and that part is mapped with an attachment. In a concrete WSDL, the attachment is described as a mime part of the Multipart message.

Before you begin

Copy the `Attachment.txt` and `tibcologo.gif` files from `TIBCO_HOME\bw\n.n\samples\binding\soap\http\BoundAttachments` to the directory specified for the **AttachmentBaseDir** property.

Procedure

1. In the **samples** directory, select **binding > soap > HTTP > BoundAttachments** and double-click **tibco.bw.sample.binding.soap.http.BoundAttachments**. For more information, see Accessing Samples.
2. In **Project Explorer** expand the **tibco.bw.sample.binding.soap.http.BoundAttachments** project.
3. Fully expand the **Processes** directory and double-click **ClientProcess.bwp**.
4. Click **Run > Debug Configurations**.
5. At the left hand tree of the Debug Configuration wizard, expand **BusinessWorks Application** and select **BWApplication**.
6. Click the **Applications** tab and then click the **Deselect All** button if you have multiple applications. Select the checkbox next to **tibco.bw.sample.binding.soap.http.BoundAttachments.application**.
7. Click **Debug**.

   This runs the sample in Debug mode.
8. In TIBCO Business Studio™ for BusinessWorks™, click the **Terminate** icon to stop the process.

Result

Sent binary file size : 2878 bytes. Received binary file size : 2878 bytes. Sent text file size : 35 Received text file size : 35

Understanding the Configuration

The **ServerProcess** implements the **getsetAttachments** operation. It receives the Client request that has WSDL message parts defined as base64Binary type and has **Binary** and **Text** attachments. Service reply mechanism sends back the same attachments to the **ClientProcess**.

The **ClientProcess** invokes the **ServerProcess** with input attachments (**Binary** and **Text**). **Consumer (Client)** reads the input files and sends them to **Producer (Server)**process. Attachments received from Server are written to the disk on the Client side.

Finally, the Sent and Received file size information for **Binary** and **Text** attachments is written at the location specified in OUTPUT_FILE module property.

These attachments' size matches and also depicts that attachments can be sent or received between the Producer and Consumer where they are defined as part of the WSDL definition.

Interoperability of Using JAX-WS Client to Invoke Service

This example illustrates the interoperability of using JAX-WS client to invoke service exposed on TIBCO ActiveMatrix BusinessWorks™.

Before you begin

Copy the **jaxws_sample.wsdl** from `TIBCO_HOME\bw\n.n\samples\binding\soap\http\JAXWSInterop` to the directory specified for “CONCRETE_WSDL_LOCATION_URL” property.

Procedure

1. In the `samples` directory, select **binding > soap > http > JAXWSInterop** and double-click **tibco.bw.sample.binding.soap.http.JAXWSInterop**. For more information, see Accessing Samples.
2. In the **Project Explorer**, expand **tibco.bw.sample.binding.soap.http.JAXWSInterop** project.
3. Fully expand the **Processes** directory and double-click **Client.bwp**.
4. Place the `jaxws_sample.wsdl` at the location of Module Property value “CONCRETE_WSDL_LOCATION_URL”.
5. Click **Run > Debug Configurations**.
6. At the left hand tree of Debug Configuration wizard, expand **BusinessWorks Application** and select **BWApplication**.

7. Click the **Applications** tab and then click the **Deselect All** button if you have multiple applications. Select the checkbox next to **tibco.bw.sample.binding.soap.http.JAXWSInterop.application**.
8. Click **Debug**.

   This runs the sample in Debug mode.
9. In TIBCO Business Studio™ for BusinessWorks™, click the **Terminate** icon to stop the process.

Result

The result shows the following response in the console:

Response for : Sending JAX-WS client request...

Understanding the Configuration

The sample consists of two processes:

- tibco.bw.sample.binding.soap.http.jaxwsinterop.Client
- tibco.bw.sample.binding.soap.http.jaxwsinterop.ServerProcess

Service process exposes an operation to accept an input string and returns string as part of response.

Client process uses a concrete WSDL generated from service to send a request with input parameters and then writes the response to file.

Client process uses a **Java Invoke** activity that passes the concrete WSDL location to the Java invoke (**tibco.bw.sample.binding.soap.http.jaxwsinterop.Client.java**) , and JAX-WS client code sets value ("Sending JAX-WS client request...") as input and then sends the request.

Invoking a SOAP over HTTP Web Service Using Different SOAP Versions

This sample shows implementation of a service for determining zip code information. This includes getting information for the city for the specific zip code and calculating distance

between two cites defined by their zip codes.
The process provides city information based on a zip code, using HTTP as the transport. The service represents a simple HTTP based service using SOAP 1.1 and SOAP 1.2.

Procedure

Configure the Server

1. In the **samples** directory, select **binding > soap > http > MixedSOAPVersions** and double-click **tibco.bw.sample.binding.soap.http.MixedSOAPVersions**. For more information, see Accessing Samples.
2. In **Project Explorer**, expand the **tibco.bw.sample.binding.soap.http.MixedSOAPVersions** project.
3. Fully expand the **Processes** directory and double-click **Process.bwp**.
4. Click **Run > Debug Configurations**.
5. At the left hand tree of Debug Configuration wizard, expand **BusinessWorks Application** and select **BWApplication**.
6. Click the **Applications** tab, and then click the **Deselect All** button if you have multiple applications. Select the checkbox next to **tibco.bw.sample.binding.soap.http.MixedSOAPVersions.application**.
7. Click **Debug**.

   The sample runs in Debug mode.

Configure the Client

8. Unzip the `TIBCO_HOME\bw\n.n\samples\bindings\soap\http\MixedSOAPVersions\bw5\tibco.bw.binding.soap.http.MixedSOAPVersionsClient.zip` file to a directory.
9. Start TIBCO Designer and open the unzipped project in the directory.
   a. In TIBCO Designer, click **Global Variables** and verify whether the variables are set correctly.
   b. Click the **Tester** tab and then click the green button. Select the checkbox next to the process and then select **Load & Start Current**.
   c. After the process completes, click the **Stop Testing** icon to stop the process.
10. In TIBCO Business Studio for BusinessWorks, click the **Terminate** icon to stop the process.

Result

- For **getCityInfoTest**, the zip code defined in the zip field returns information about Urbana IL.
- For **getCityDistanceTest**, the distance between two zip codes is returned.

The following output is visible in the console:

12:37:20.751 INFO [bwThread:In-Memory STWorkProcessor-3] c.t.b.p.g.L.t.b.s.b.s.h.M.Log - Invoked cityinfo

12:37:20.774 INFO [bwThread:In-Memory STWorkProcessor-4] c.t.b.p.g.L.t.b.s.b.s.h.M.Log - getCityInfo:

city: Urbanastate: Illinois

location: Urbana, Illinois, United States

latitude: 44.11

longitude: 88.207

zip: 61801

12:37:22.642 INFO [bwThread:In-Memory STWorkProcessor-6] c.t.b.p.g.L.t.b.s.b.s.h.M.Log1 - Invoked cityDistance

12:37:22.650 INFO [bwThread:In-Memory STWorkProcessor-7] c.t.b.p.g.L.t.b.s.b.s.h.M.Log - getCityDistance: 4

Understanding the Configuration

The server process has two operations implemented: **getCityInfo** and **getCityDistance** are used to return information, based on zip codes.

The operations implemented in the server process are invoked using the **getCityDistanceTest** and **getCityInfoTest** test processes from the client project.

- The endpoint ConcreteZipInfo_SOAP1.1 using SOAP 1.1 implements getCityInfoTest.
- The endpoint, ConcreteZipInfo_SOAP1.2 using SOAP 1.2 implements getCityDistance.

Invoking a SOAP over HTTP Web Service Using Different Binding Styles

This sample shows the creation and invocation of a SOAP over HTTP web service. It demonstrates how to use two different binding styles, RPC and DOC Literal for the same service on two different endpoints.

Before you begin

Note: To run the sample you can either use TIBCO ActiveMatrix BusinessWorks™ 6.x client or ActiveMatrix BusinessWorks™ 5.x as a client.

Procedure

Configure the Server

1. In the **samples** directory, select **binding > soap > http > MixedStyles** and double-click **tibco_bw_sample_binding_soap_http_mixedstylesprovider**. For more information, see Accessing Samples.
2. In **Project Explorer**, expand the **tibco_bw_sample_binding_soap_http_mixedstylesprovider** project.
3. Set the default application profile to match the OS you are running on. For more information, see Setting the Default Application Profile.
4. Fully expand the **Processes** directory and double-click **Process.bwp**.
5. Click **Run > Debug Configurations**.
6. At the left hand tree of Debug Configuration wizard, expand **BusinessWorks Application** and select **BWApplication**.
7. Click the **Applications** tab and then click the **Deselect All** button if you have multiple applications. Select the checkbox next to **tibco_bw_sample_binding_soap_http_mixedstylesprovider**.
8. Click **Debug**.

   This runs the sample in Debug mode.

Configure the Client

9. Unzip the `TIBCO_`

`HOME\bw\n.n\samples\bindings\soap\http\MixedStyles\bw5\tibco.bw.binding.soap.http.MixedStylesClient.zip` file to a directory.

10. Start TIBCO Designer and open the project unzipped in the directory.
    a. In TIBCO Designer, click **Global Variables** and verify the variables are set correctly.
    b. Click **Tester** and then click the green button. Select the checkbox next to the process and then click **Load & Start Current**.
    c. After the process completes, click the **Stop Testing** icon to stop the process
11. In TIBCO Business Studio™ for BusinessWorks™, click the **Terminate** icon to stop the process.

Result

The Service has the following two different endpoint bindings:

- The **Document/Literal** is the endpoint for ZipPortEndpoint1.
- The **RPC/Literal** is the endpoint for ZipPortEndpoint2.

For the **getCityInfoTest** operation, you can view the information corresponding to the zip code that is defined in the zip field. Its default value is 60801. This should return information about Urbana, IL.

For the **getCityDistanceTest** operation, you can view the information corresponding to the distance between two zip codes. The default values are:

- 60801: Urbana, IL
- 61820: Champaign, IL

The console shows the following output:

15:24:51.971 INFO [bwThread:In-Memory STWorkProcessor-3] c.t.b.p.g.L.t.b.s.b.s.h.M.Log - Invoked cityinfo

15:24:51.992 INFO [bwThread:In-Memory STWorkProcessor-4] c.t.b.p.g.L.t.b.s.b.s.h.M.Log - getCityInfo-DOC:

city: Urbana

state: Illinois

location: Urbana, Illinois, United States

latitude: 40.11

longitude: 88.207

zip: 61801

15:24:53.857 INFO [bwThread:In-Memory STWorkProcessor-6] c.t.b.p.g.L.t.b.s.b.s.h.M.Log1 - Invoked cityDistance

15:24:53.865 INFO [bwThread:In-Memory STWorkProcessor-7] c.t.b.p.g.L.t.b.s.b.s.h.M.Log - getCityDistance-RPC: 4

Understanding the Configuration

At run-time for the first operation, the client sends a SOAP request comprising a zip code and receives a city information of that zip code. For the second operation, the client sends a SOAP request comprising two zip codes and returns the distance between the two cities defined by their zip codes.

The following shared resources are defined in the project:

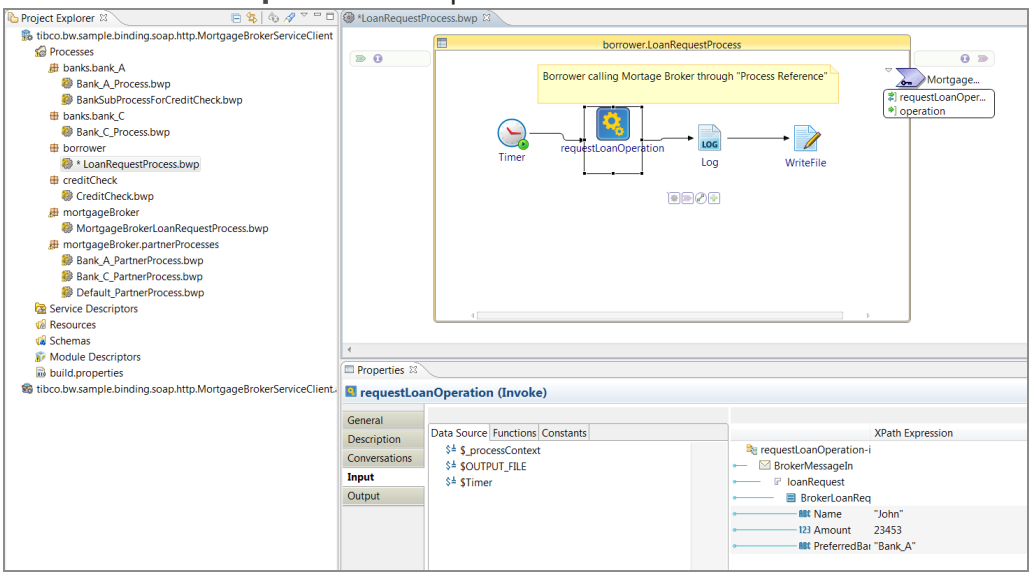
- **HTTPConnectorResource.httpConnResource**: The HTTP Connection Shared Resource containing the transport configuration.
- **CitySchema.xsd**: The XSD Schema file containing the definition of the types used by the web service interface.
- **ZipInfo.wsdl**: The WSDL file that describes the web service.

Implementing a Service with a Service Binding and a Client with a Reference Binding

In this sample, a service implements a simplified online mortgage broker application. The borrower requests a loan through a broker. The broker processes the loan request using one of the third-party partner services. The borrower can either specify the preferred third-party provider or allow the broker to default to one. The third-party partner services request credit rating of the borrower from a credit check service and in turn approves or rejects the loan application based on the credit rating.

Procedure

1. In the **samples** directory, select **binding > soap > http > MortgageBrokerServiceClient** and double-click **tibco.bw.sample.binding.soap.http.MortgageBrokerServiceClient**. For more information, see Accessing Samples.
2. In **Project Explorer**, expand the **tibco.bw.sample.binding.soap.http.MortgageBrokerServiceClient** project.
3. You can modify the values specified for the Name, Amount, and Preferred bank fields by clicking on the **Input** tab of the **Invoke** activity Client in the **borrower.LoanRequestProcess** process.



4. Click **Run > Debug Configurations**.
5. At the left hand tree of **Debug Configuration** wizard, expand **BusinessWorks Application** and select **BWApplication**.
6. Click the **Applications** tab and then click the **Deselect All** button if you have multiple applications. Select the checkbox next to **tibco.bw.sample.binding.soap.http.MortgageBrokerServiceClient.application**.
7. Click **Debug**.

   This runs the sample in Debug mode.
8. Click the **Terminate** icon to stop the process.

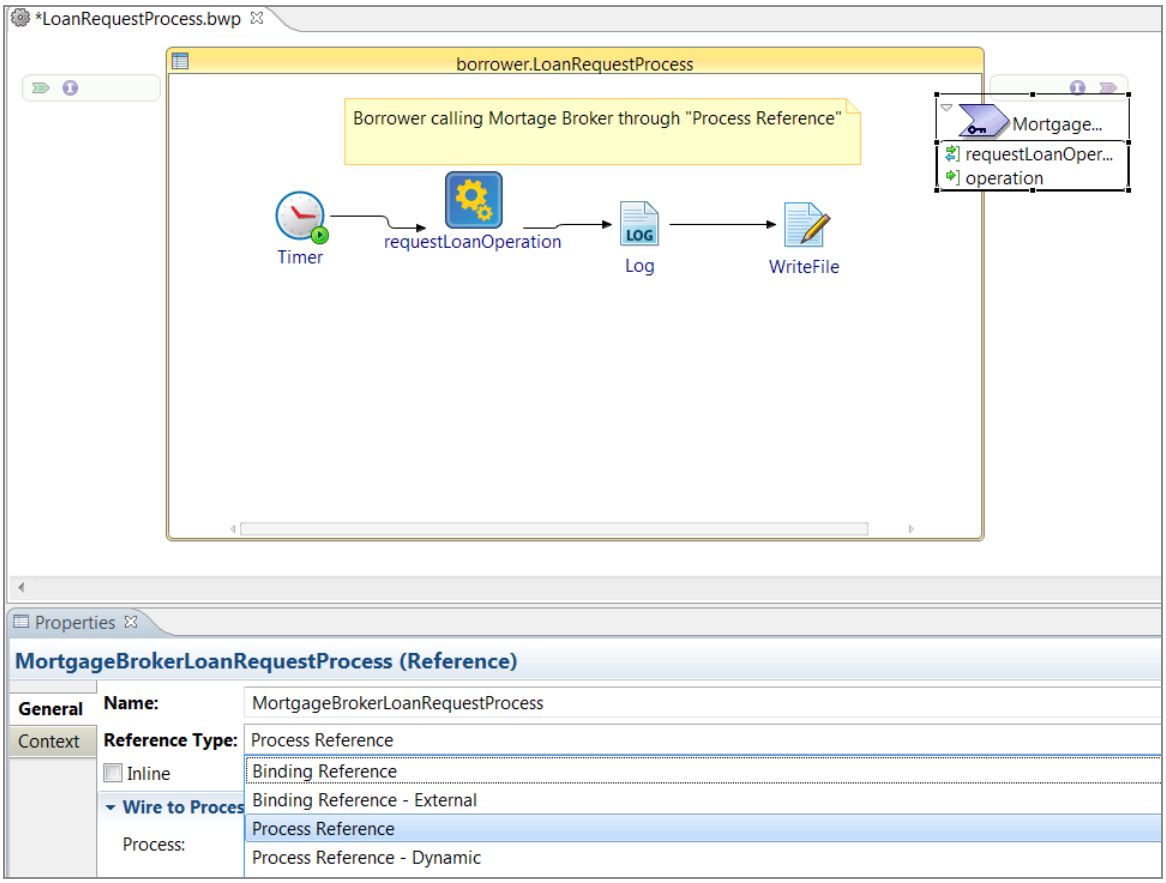
Result

The response is printed to the console. The response contents are either response from Mortgage Broker:Name: John: Status: REJECTED, or response from Mortgage Broker:Name: John : Status: APPROVED.

- When a preferred bank is specified, the corresponding bank service is invoked. If not, Bank_A service is invoked.
- When the Bank_A and Bank_C services are invoked, either an Approved or Rejected message is returned, depending on the (random) credit score for the applicant.

Understanding the Configuration

This sample illustrates various features of the Process Reference. These features are present as options of **Reference Type** available on the **General** tab of the properties view of the Process Reference.



The available options are:

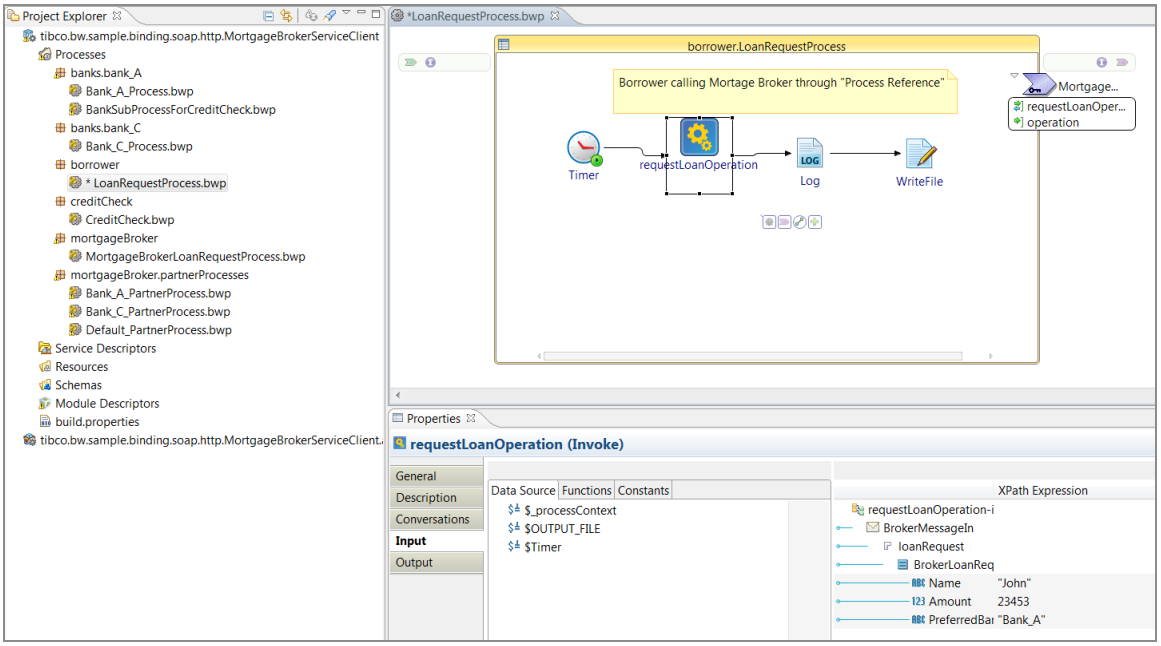
- Binding Reference
- Binding Reference - External
- Process Reference
- Process Reference - Dynamic

The following processes defined in this sample explain their features in the processes.
`borrower.LoanRequestProcess`

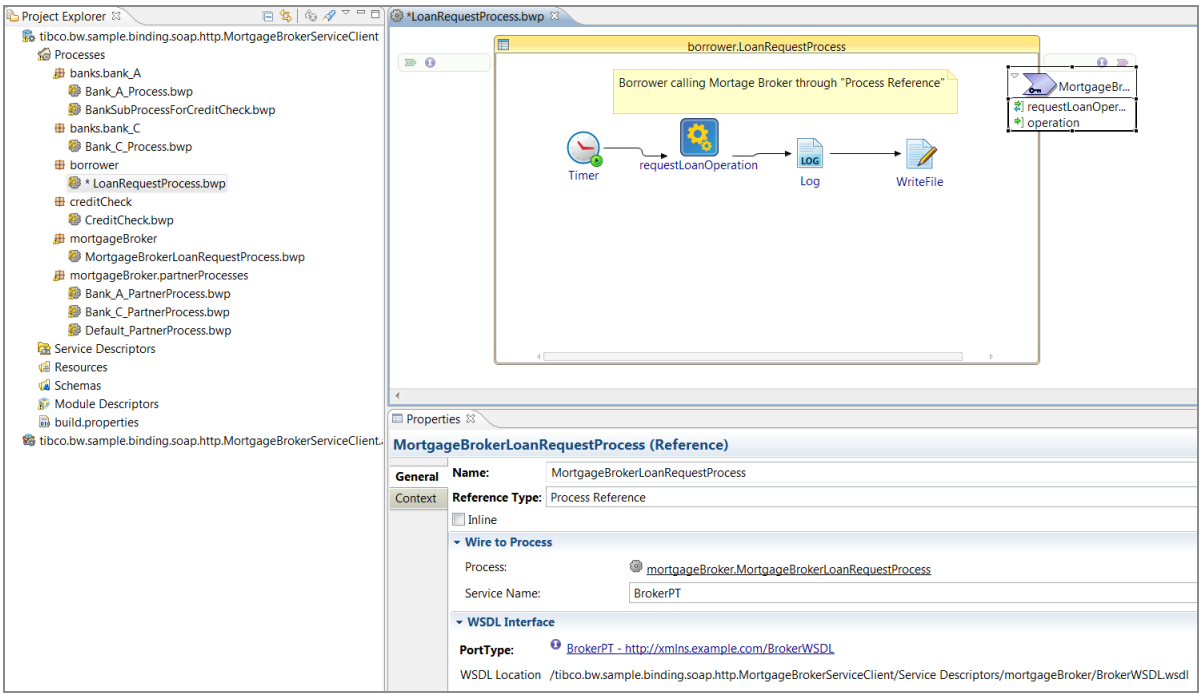
This process acts as a Consumer and allows the borrower to issue a loan request to the Mortgage Broker Service. The borrower specifies the values of the following fields in the requestLoanOperation activity in the borrower.LoanRequestProcess process.

- Borrower name
- Loan amount requested
- Preferred bank. Specify one of the available banks "Bank_A" and "Bank_C". By default, the broker service calls bank A, if no preferred bank is specified.

> **Note:** Add an empty string "", for not specifying a preferred bank.



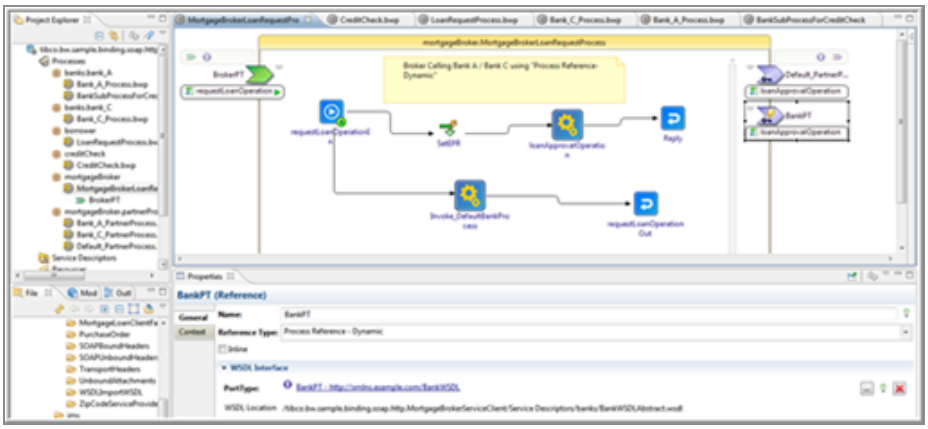
This process uses **Process Reference** option to invoke the Mortgage broker process. This options statically map the two Processes together.



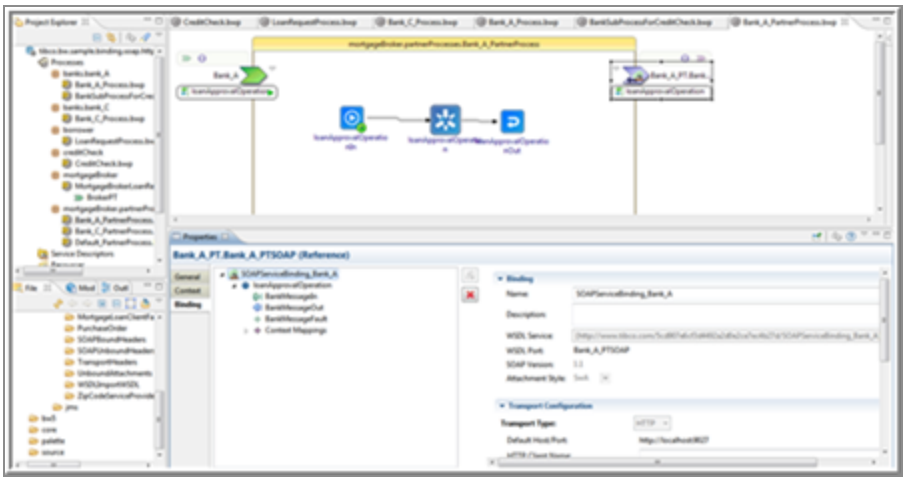
`mortgageBroker.LoanRequestProcess`

This process receives the request from the borrower and calls the available SOAP Service Providers through subprocess calls.

Depending on the value of "Preferred bank" request field, this process either uses **Process Reference -Dynamic** option with **Set EPR** activity to dynamically call **Bank_A_ PartnerProcess** or **Bank_C_PartnerProcess** sub process OR uses the **Process Reference** option to statically invoke the **Default_PartnerProcess** sub process.



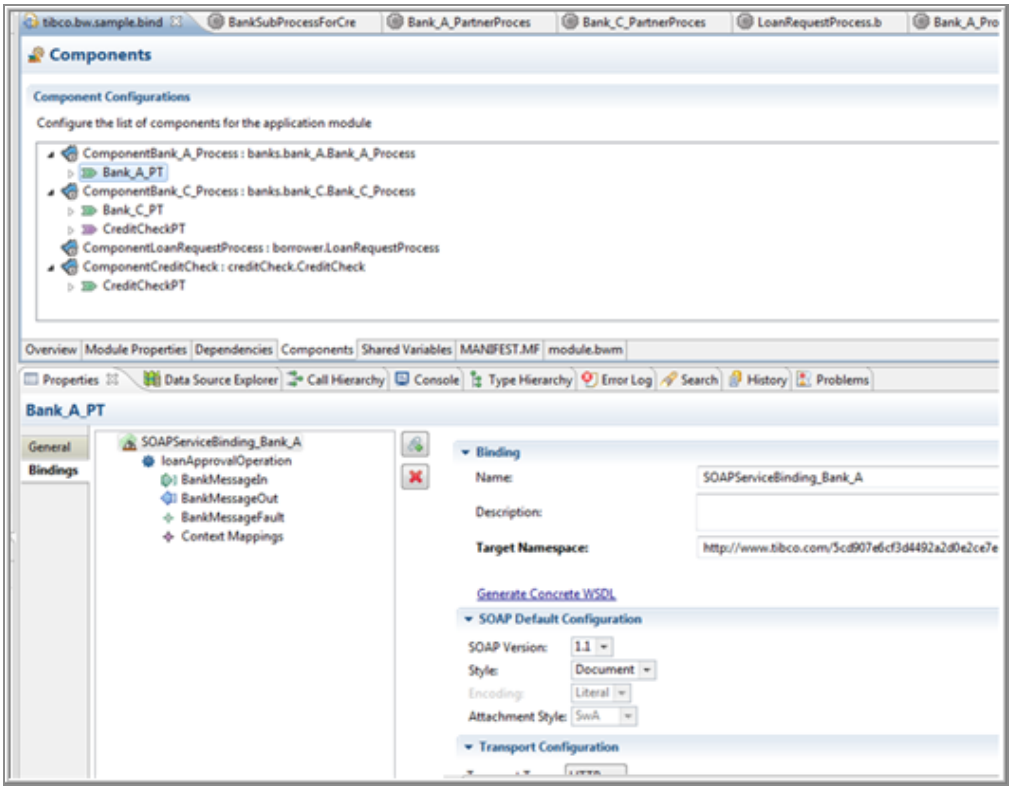
Each subprocess uses the **Binding Reference** option on the **Process Reference** to act as a SOAP Service Consumer, of an available Bank Service.



banks.bank_A.Bank_A_Process

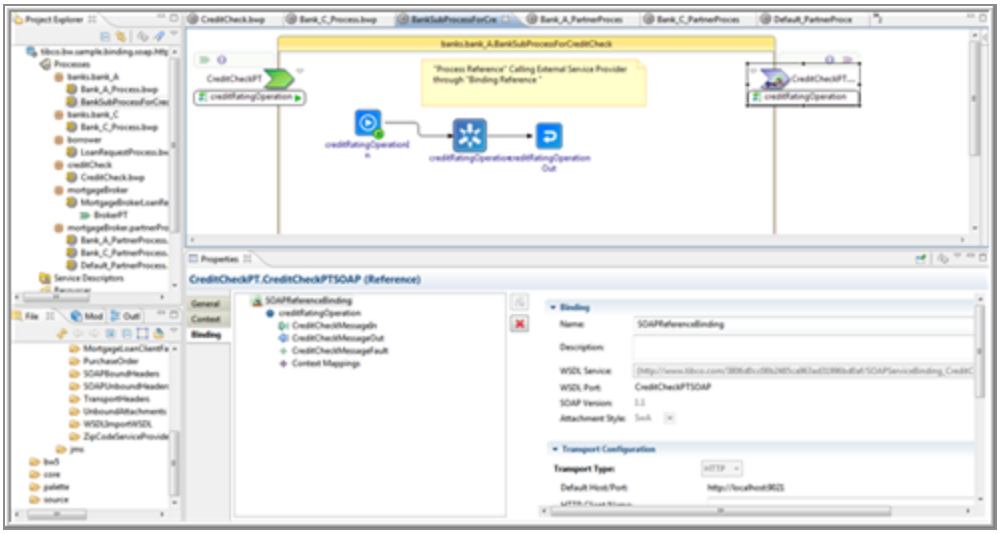
This process acts as both a SOAP Service Provider for the mortgage broker and a SOAP Service Consumer for the Credit Check Service.

Note: SOAP Service Provider configuration is specified at the Component level.



Bank_A_Process invokes the credit check service through "BankSubProcessForCreditCheck" sub process. This subprocess uses **Binding Reference** option, on the Process Reference, to

act as a SOAP Service Consumer.



banks.bank_C.Bank_C_Process

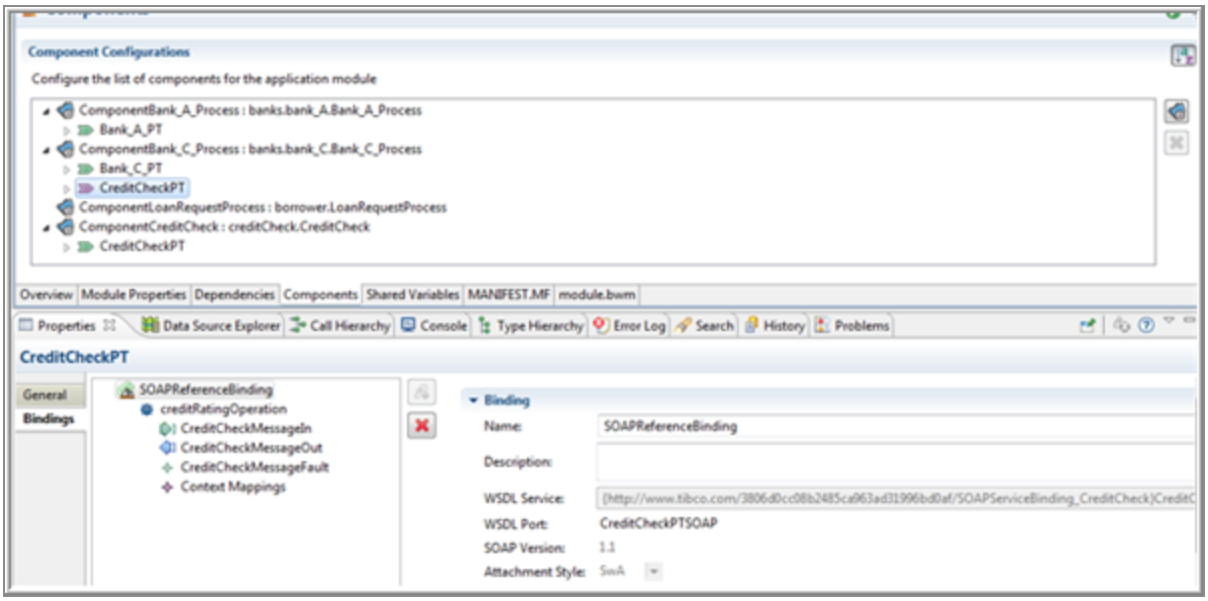
This process receives the details of the borrower from the mortgage broker and queries a credit check services to validate the borrower credentials. The bank can specify the borrower name. The process exposes services to the mortgage broker.

This process acts as both a SOAP Service Provider for the mortgage broker and a SOAP Service Consumer for the Credit Check Service.

Note: SOAP Service Provider configuration is specified at the Component level.

Bank_C_Process uses **Binding Reference-external** option on the Process Reference, to act as a SOAP Service Consumer of the Credit check Service.

Note: Configuration of **Binding Reference-external** is specified at the Component Level.



`creditCheck.CreditCheck`

This process acts as a SOAP Service Provider for credit check functionality. It processes the request returns the credit rating for a customer.

Note: SOAP Service Provider is configured at the Component level.

Implementing a Mortgage Loan Application with Fault

This sample shows a service implementation that invokes fault using a service and reference binding.

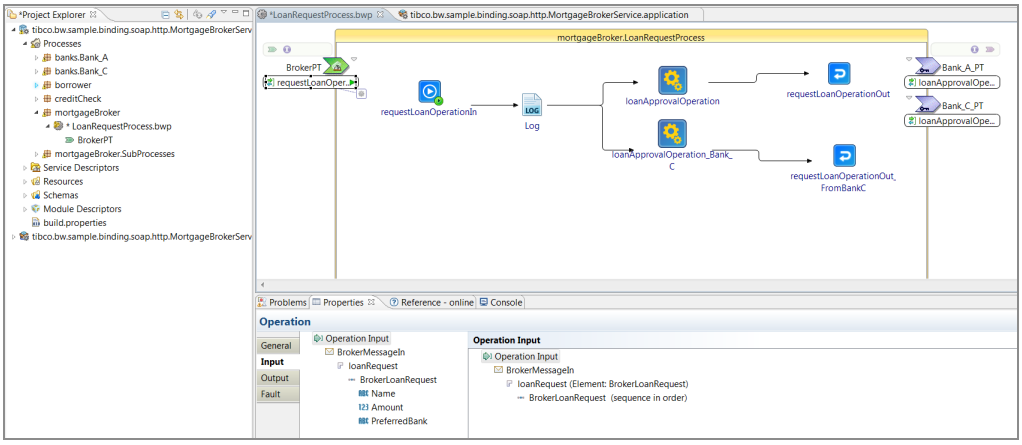
In this sample, the service implements a simplified online mortgage broker application. The borrower requests a loan through a broker and the broker processes the loan request using one of the third-party partner services. The borrower can specify the preferred third-party provider. The third-party partner services request credit rating of the borrower from a credit check service and in turn replies with fault message if requested loan amount is not valid.

Procedure

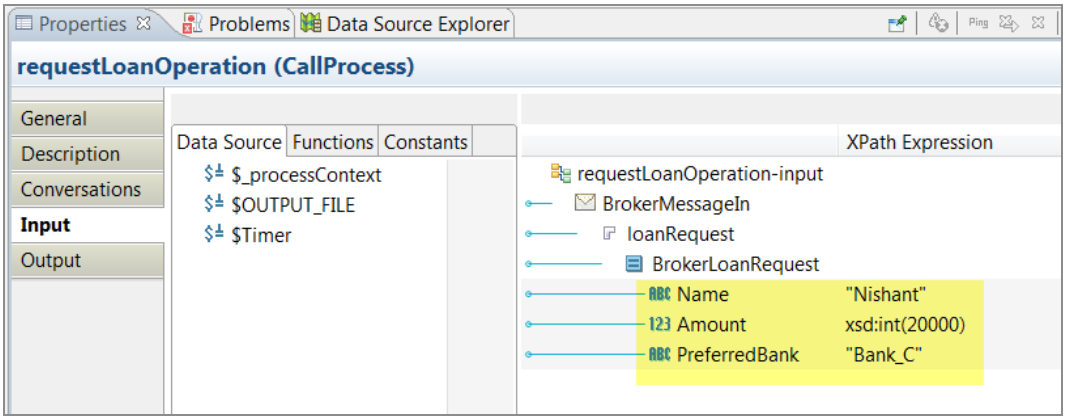
1. In the **samples** directory, select **binding > soap > http > MortgageLoanClientFault**

and double-click **tibco_bw_sample_binding_soap_http_mortgagebrokerservice**. For more information, see Accessing Samples.

2. In **Project Explorer**, expand the **tibco_bw_sample_binding_soap_http_ mortgagebrokerservice** project.
3. Fully expand the **Processes** directory and double-click **LoanRequestProcess.bwp**.
4. Double-click the **requestLoanOperation** activity.



5. Click the **Input** tab and modify the values for the Name, Amount and Preferred bank fields. Enter amount less than or equal to 25000 or greater than equal to 100000 in the Amount field.



6. Click **Run > Debug Configurations**.
7. At the left hand tree of Debug Configuration wizard, expand **BusinessWorks Application** and select **BWApplication**.
8. Click the **Applications** tab and then click the **Deselect All** button if you have multiple

applications. Select the checkbox next to **tibco_bw_sample_binding_soap_http_mortgagebrokerservice_application**.

9. Click **Debug**.

   This runs the sample in Debug mode.

10. Click the **Terminate** icon to stop the process.

Result

When a preferred bank is specified, the corresponding bank service is invoked.

Bank_A and Bank_C, when invoked, returns fault message Requested Loan Amount is not valid. Loan amount should be more than 25000 and less than 100000.

Understanding the Configuration

The following shared resources are defined in the project:

- **HTTPConnectorResource.httpConnResource**: The HTTP Connection Shared Resource that contains the transport configuration.
- **BankLoanSchema.xsd**: The XSD Schema file that contains the definition of the types used by the Bank web service's interface.
- **BrokerSchema.xsd**: The XSD Schema file that contains the definition of the types used by the Broker web service's interface.
- **CreditCheckSchema.xsd**: The XSD Schema file that contains the definition of the types used by the Credit Check web service's interface.
- **BankWSDLAbstract.wsdl**: Common service interface to request for a loan from a bank.
- **CreditCheckWSDL.wsdl**: The WSDL file that describes the Credit Check web service.
- **BrokerWSDL.wsdl**: The WSDL file that describes the Mortgage Broker web service.

The example consists of the following:

- **CreditCheckService**: Provides services for loan approvals. In this example, the borrowers can choose one of the two banks, Bank A and Bank C. These services call CreditCheckService to approve or reject a loan request or to verify if the requested loan amount is not valid.

- **Bank_A_Process / Bank_C_Process**: Provides service to get the credit rating for a customer by calling a process that generates a credit score or reply with a fault message if the requested loan amount is invalid.
- **mortgageBroker.LoanRequestProcess**: Provides service to request for a loan through the Mortgage Broker. The broker in turn checks with the preferred bank for the loan.
- **LoanRequestProcess**: Allows the borrower to issue a loan request. The borrower can specify Name, Amount, and Preferred Bank. Depending on the Preferred Bank value the respective Bank Process is invoked.

Sending and Receiving a SOAP Unbound Attachment

A client sends **Text** and **GIF** files as attachments for unbound attachments to the server process. The Server process receives these attachments and returns them as its response. The Client process captures the sent and received attachments size.

Before you begin

Copy the `Attachment.txt` and `Client.gif` files from `TIBCO_HOME\bw\n.n\samples\binding\soap\http\UnboundAttachments` to the directory given for the **AttachmentBaseDir** property.

> **Note:** It is an **Unbound Attachment** when no part of the Input or Output WSDL message is mapped to an attachment. In the concrete WSDL there is no indication whether an attachment is required in the Input or Output. You can map an attachment in the mimePart config at Binding level only at design time.

Procedure

1. In the **samples** directory, select **binding > soap > HTTP > UnboundAttachments** and double-click **tibco.bw.sample.binding.soap.http.UnboundAttachments**. For more information, see Accessing Samples.
2. In the **Project Explorer** view, expand the **tibco.bw.sample.binding.soap.http.UnboundAttachments** project.

3. Click **Run > Debug Configurations**.
4. At the left hand tree of Debug Configuration wizard, expand **BusinessWorks Application** and select **BWApplication**.
5. Click the **Applications** tab and then click the **Deselect All** button if you have multiple applications. Select the checkbox next to **tibco.bw.sample.binding.soap.http.UnboundAttachments.application**.
6. Click **Debug**.

   This runs the sample in Debug mode.
7. In TIBCO Business Studio™ for BusinessWorks™, click the **Terminate** icon to stop the process.

Result

:

Sent binary file size : 3587 bytes. Received binary file size : 3587 bytes. Sent text file size : 37 Received text file size : 37

Understanding the Configuration

The **ServerProcess** implements the **getsetAttachments** operation along with **Context** activities. It receives the client request that has WSDL message parts defined as **Element of Complex** type. **Binary** and **Text** attachments are retrieved using **Get Context** activity. Service reply mechanism sends back the same attachments to the **ClientProcess** using the **Set Context** activity.

The **ClientProcess** invokes the **ServerProcess** with input attachments (**Binary** and **Text**) using **Set Context** activity. The **Consumer (Client)** reads the input files and sends them to the **Producer (Server)** process using the context mechanism. Attachments received from the server through **Get Context** activity are written to disk on the client side.

Finally, the Sent and Received file size information for Binary and Text attachments is written at the location specified in the OUTPUT_FILE module property.

These attachments' size matches and also depicts that attachments can be sent or received between the **Producer** and **Consumer** where they are defined using the context mechanism outside of WSDL definition.

Managing a Purchase Order

A purchase order moves through a set of operations including submission, verifying status, calculating price, and managing faults. This sample shows the processes used to manage these operations.

Before you begin

Note: To run the sample you can either use TIBCO ActiveMatrix BusinessWorks™ 6.x client or ActiveMatrix BusinessWorks™ 5.x as a client.

Procedure

Configure the Server

1. In the **samples** directory, select **binding > soap > http > PurchaseOrder** and double-click **tibco.bw.sample.binding.soap.http.PurchaseOrder**. For more information, see Accessing Samples.
2. In **Project Explorer** expand the **tibco.bw.sample.binding.soap.http.PurchaseOrder** project.
3. Copy the `ShippingCosts.xml` and `PriceBook.xml` files from `TIBCO_HOME\bw\n.n\samples\binding\soap\http\PurchaseOrder` to the directory given for the INPUT_DIRECTORY property.
4. Click **Run > Debug Configurations**.
5. At the left hand tree of Debug Configuration wizard, expand **BusinessWorks Application** and select **BWApplication**.
6. Click the **Applications** tab and then click the **Deselect All** button if you have multiple applications. Select the checkbox next to **tibco.bw.sample.binding.soap.http.PurchaseOrder.application**.
7. Click **Debug**.

   This runs the sample in Debug mode.

Configure the ActiveMatrix BusinessWorks 5.x Client

8. Extract the ActiveMatrix BusinessWorks 5.x client project `bw5\tibco.bw.sample.binding.soap.http.PurchaseOrder.Client.zip` into a temporary directory.

9. Start TIBCO Designer and import the ActiveMatrix BusinessWorks 5.x client project from the temporary directory.
10. In TIBCO Designer verify the output file, Host, and Port global variables.

    Note: If you change the module properties in the project, you must regenerate the concrete WSDL and import it into the 5.x client project.

11. In TIBCO Designer, click **Tools > Tester > Start**.
12. In TIBCO Business Studio™ for BusinessWorks™, click the **Terminate** icon to stop the process.

Result

See the following output:

Purchase ID 1234 is not valid

Submit PurchaseOrder Successful

Status PurchaseOrder :Pending

Status PurchaseOrder: Processed

Understanding the Configuration

The **SellerService** main process implements the **submitPurchaseOrder** and **checkStatusPurchaseOrder** operations. The **submitPurchaseOrder** operation returns a purchase order ID. A buyer uses this ID to check status when invoking the **checkStatusPurchaseOrder** operation. Each operation calls subprocesses:

- The **SubmitPOExecute** subprocess is called from the **submitPurchaseOrder** operation and initiates the purchase order submission by calling the **SubmitProcessOrder** subprocess and then generates the purchase order ID.
- The **SubmitProcessOrder** subprocess calls the **GetPriceForItem** and **CalculateShippingCosts** subprocesses to calculate item prices and shipping costs respectively. After these calculations complete, the purchase order changes to processed status, from pending status.
- The **GetPriceForItem** subprocess reads data from `PriceBook.xml` and returns the price for an item.

- The **CalculateShippingCosts** subprocess reads data from `ShippingCosts.xml` and returns the shipping costs based on region.
- The **CheckStatus** subprocess is called from the **checkStatusPurchaseOrder** operation and returns the status of purchase order, either pending or processed.

Using SOAP Bound Headers

This sample shows how to use SOAP Bound Headers by passing any data or control information in the SOAP envelope. In this sample the Input WSDL message part is configured as a Header in the WSDL definition.

Procedure

1. In the samples directory, select **binding > soap > http > SOAPBoundHeaders** and double-click **tibco.bw.sample.binding.soap.http.SOAPBoundHeaders**. For more information, see Accessing Samples .
2. In the **Project Explorer** expand the **tibco.bw.sample.binding.soap.http.SOAPBoundHeaders** project.
3. Fully expand the **Processes** directory and double-click **Client.bwp**.
4. Click **Run > Debug Configurations**.
5. At the left hand tree of Debug Configuration wizard, expand **BusinessWorks Application** and select **BWApplication**.
6. Click the **Applications** tab and then click the **Deselect All** button if you have multiple applications. Select the checkbox next to **tibco.bw.sample.binding.soap.http.SOAPBoundHeaders.application**
7. Click **Debug**.

   This runs the sample in the Debug mode.
8. In TIBCO Business Studio™, click the **Terminate** icon to stop the process.

Result

The expected outcome for the file identified in the OUTPUT_FILE property is similar to the following console output:

10:41:08.248 INFO [bwThread:In-Memory STWorkProcessor-2] c.t.b.p.g.L.t.b.s.b.s.h.S.Log - Invoked cityinfo

10:41:08.273 INFO [bwThread:In-Memory STWorkProcessor-3] c.t.b.p.g.L.t.b.s.b.s.h.S.Log - CityInfo response:

city: Urbana

state: Illinois

location: Urbana, Illinois, United States

latitude: 40.11

longitude: 88.207

zip: 61801

10:41:10.299 INFO [bwThread:In-Memory STWorkProcessor-5] c.t.b.p.g.L.t.b.s.b.s.h.S.Log1 - Invoked cityDistance

10:41:10.306 INFO [bwThread:In-Memory STWorkProcessor-6] c.t.b.p.g.L.t.b.s.b.s.h.S.Log1 - CityDistance response:

Distance from cityUrbana to city Champaign is 4

Understanding the Configuration

This sample implements a service for determining information about the zip codes. This includes getting information for the city at the specified zip code, and for getting the distance between two cites (defined by their zip codes). **CityInfo** operation input message is configured as **SOAP Header** data whereas the **CityDistance** operation input message is configured as **SOAP Body** data.

Two operations are implemented in the server process. The service operations **getCityInfo** and **getCityDistance** provide a means for getting the city information about a zip code and distance between the cities. Both operations are invoked from the client side with a small sleep between invocations.

Using SOAP UnBound Headers

SOAP Unbound headers can pass any Out-of-Band data or control information in the SOAP envelope. In this sample a simple getBook operation based on the specified interface WSDL file is implemented showing the use of SOAP Unbound Headers.

Procedure

1. In the **samples** directory, select **binding > soap > HTTP > SOAPHeaders** and double-click **tibco.bw.sample.binding.soap.http.SOAPUnboundHeaders**. For more information, see Accessing Samples.
2. In the **Project Explorer** expand the **tibco.bw.sample.binding.soap.http.SOAPUnboundHeaders** project.
3. Fully expand the **Processes** directory and double-click **ClientProcess.bwp**.
4. Click **Run > Debug Configurations**.
5. At the left hand tree of Debug Configuration wizard, expand **BusinessWorks Application** and select **BWApplication**.
6. Click **Applications** and then click the **Deselect All** button if you have multiple applications. Select the checkbox next to **tibco.bw.sample.binding.soap.http.SOAPUnboundHeaders.application**.
7. Click **Debug**.

   This runs the sample in Debug mode.
8. In TIBCO Business Studio™ for BusinessWorks™, click the **Terminate** icon to stop the process.

Result

:

Expected captchaID : CID007

Received captchaID : CID007

Understanding the Configuration

This sample shows the use of SOAP unbound headers on service and reference.

The service exposes a login method that accepts a username and password, and expects a `captcha` string to be passed using SOAP unbound headers. Verification of the received username and password returns a **captchaID** to caller.

The ClientProcess sends a username and password as part of request message and sends the `captcha` string using headers.

The ServerProcess receives the username and password and, on successful verification of credentials, returns a `captchaID` as part of the headers and returns a Boolean indicating login success or failure.

Using SOAP HTTP Transport Headers

This sample shows how to use SOAP Transport headers received on the service side. The service publishes the transport header elements for two operations.

Procedure

1. In the **samples** directory, select **binding > soap > HTTP > TransportHeaders** and double-click **tibco.bw.sample.binding.soap.http.TransportHeaders**. For more information, see Accessing Samples.
2. In **Project Explorer** expand the **tibco.bw.sample.binding.soap.http.TransportHeaders** project.
3. Fully expand the **Processes** directory and double-click **Client.bwp**.
4. Click **Run > Debug Configurations**.
5. At the left hand tree of Debug Configuration wizard, expand **BusinessWorks Application** and select **BWApplication**.
6. Click the **Applications** tab and then click the **Deselect All** button if you have multiple applications. Select the checkbox next to **tibco.bw.sample.binding.soap.http.TransportHeaders.application**.
7. Click **Debug**.

   This runs the sample in Debug mode.
8. In TIBCO Business Studio™, click the **Terminate** icon to stop the process.

Result

For **getCityInfoTest**, you see the information corresponding to the zip code that is defined in the zip field. Its default value is **61801**, which returns information about **Urbana IL**.

For **getCityInfoTestHeaders**, you see the information corresponding to the distance between two zip codes. The default values are 61801 for **Urbana IL** and 61820 for **Champaign IL**.

HTTP Transport Headers from the client side are captured along with input data on the service side. Transport Headers parameters are saved on service side.

The file specified for the OUTPUT_FILE property is written with the following contents:

HTTP Transport Header elements for operation CityInfo:

method: POST

requesturi: /SOAPServiceBinding/zipPort/

httpversion: 1.1

querystring:

protocol: HTTP/1.1

port: 11005

Header->accept:

Header->accept-charset:

Header->accept-encoding: Header->content-type: text/xml; charset=UTF-8

content-length: 167

connection:

cookie:

pragma:

HTTP Transport Header elements for operation CityDistance:

method: POST

requesturi: /SOAPServiceBinding/zipPort/

httpversion: 1.1

querystring:

protocol: HTTP/1.1

port: 11005

Header->accept:

Header->accept-charset:

Header->accept-encoding: Header->content-type: text/xml; charset=UTF-8

content-length: 360

connection:

cookie:

pragma:

Understanding the Configuration

This sample implements a service to determine the information about the zip codes along with passing the HTTP Transport parameters using the context resource. This includes getting information for the city at the specified zip code, and for getting the distance between two cites (defined by their zip codes).

Two operations are implemented in the server process. The service operations **getCityInfo** and **getCityDistance** provide a means for getting the city information about a zip code and distance between the cities. HTTP Transport Headers are captured on the service side using the Context activities.

Both operations are invoked from the client side with a small sleep between invocations. HTTP Transport Headers details from client side are captured as an output on the service side.

Sending a SOAP Request and Receiving a SOAP Reply

A SOAP client is configured to send messages to a SOAP server, which processes the requests and sends the responses back to the SOAP client.

Before you begin

Note: To run the sample you can either use TIBCO ActiveMatrix BusinessWorks 6.x client or TIBCO ActiveMatrix BusinessWorks 5.x as a client.

Procedure

Configure the Server

1. In the **samples** directory, select **binding > soap > http > WSDLImportWSDL**and double-click **tibco.bw.sample.binding.soap.http.WSDLImportWSDL**. For more information, see Accessing Samples.
2. In **Project Explorer** expand the **tibco.bw.sample.binding.soap.http.WSDLImportWSDL** project.
3. Fully expand the **Processes** directory and double-click **QueryBooksByAuthor.bwp**.
4. Click **Run > Debug Configurations**.
5. At the left hand tree of Debug Configuration wizard, expand **BusinessWorks Application** and select **BWApplication**.
6. Click the **Applications** tab and then click the **Deselect All** button if you have multiple applications. Select the checkbox next to **tibco.bw.sample.binding.soap.http.WSDLImportWSDL.application**.
7. Click **Debug**.

   This runs the sample in Debug mode.

Configure the ActiveMatrix BusinessWorks 5.x Client

8. In TIBCO Designer, import the client project, `bw5\tibco.bw.sample.binding.soap.http.WSDLImportWSDL.client`.
9. Update the HTTP shared resource configuration and verify the Host and Port values and OUTPUT_FILE global variable.

   If you updated `HTTPConnector.httpConnResource` in the project, you must regenerate the concrete WSDL and import it into the 5.x client project.
10. In TIBCO Designer, click **Tools > Tester > Start**.
11. In TIBCO Business Studio™, click the **Terminate** icon to stop the process.

Result

The output of the **SOAP Request Reply** activity shows the book values returned by the server, Advanced Java John Doe March, 2002 0596002111 O'Connelly.

Understanding the Configuration

This sample shows how to generate a concrete WSDL, when an abstract WSDL imports another WSDL, and how it is exposed by a service using a SOAP over HTTP binding. The server side is implemented in ActiveMatrix BusinessWorks 6.x and the client side is implemented in ActiveMatrix BusinessWorks 5.x. The client side shows a SOAP palette resource named **SOAP Request Reply**, which is used to send a SOAP request to the server and receive a SOAP response.

The abstract WSDL is named `BooksInterface.wsdl`:

- The `BooksInterface.wsdl` abstract file imports the `BooksMessage.wsdl` file, which contains messages used by the operations in `BooksInterface.wsdl`.
- The `BooksMessage.wsdl` file imports the `Books.xsd` schema file, which contains elements used by the messages in the `BooksMessage.wsdl` file.
- The `Books.xsd` schema file includes the `BooksType.xsd` schema file, which contains the types used by the elements in the `Books.xsd` file.
- The SOAP over HTTP binding is used in the process service.

tibco.bw.sample.binding.soap.http.WSDLImportWSDL in ActiveMatrix BusinessWorks 6.x contains the **QueryBooksByAuthor** process, which shows how to receive a SOAP request from the SOAP client and process the request. This process runs as follows:

1. The service exposed via SOAP over HTTP receives a SOAP request.
2. The **Reply** activity from the Basic Activities palette sends a response back to the client.

tibco.bw.sample.binding.soap.http.WSDLImportWSDL.client in ActiveMatrix BusinessWorks 5.x contains the **QueryBooksByAuthor** process, which shows how to send a SOAP request to the SOAP server and process the response. This process definition runs as follows:

1. The **Timer** activity starts a job.

2. The **SOAP Request Reply** activity sends a SOAP request to the SOAP server and receives the response from the server and writes the response to OUTPUT_FILE location.

Implementing a HTTP Service That Returns Information Based on ZIP Codes

The HTTP service returns information about a city, given a ZIP code. It also provides the distance between two cities as defined by their ZIP codes.

Before you begin

Note: To run the sample you can either use TIBCO ActiveMatrix BusinessWorks™ 6.x client or ActiveMatrix BusinessWorks™ 5.x as a client.

Procedure

Configure the Server

1. In the **samples** directory, select **binding > soap > http > ZipCodeServiceProvider** and double-click **tibco.bw.sample.binding.soap.http.ZipCodeServiceProvider**. For more information, see Accessing Samples.
2. In **Project Explorer** expand the **tibco.bw.sample.binding.soap.http.ZipCodeServiceProvider** project.
3. Fully expand the **Processes** directory and double-click **ZipCodeService.bwp**.
4. Click **Run > Debug Configurations**.
5. At the left hand tree of Debug Configuration wizard, expand **BusinessWorks Application** and select **BWApplication**.
6. Click the **Applications** tab and then click the **Deselect All** button if you have multiple applications. Select the checkbox next to **tibco.bw.sample.binding.soap.http.ZipCodeServiceProvider.application**.
7. Click **Debug**.

   This runs the sample in Debug mode.

Configure the ActiveMatrix BusinessWorks 5.x Client

8. In TIBCO Designer, import the client project, `bw5\tibco.bw.sample.binding.soap.http.ZipCodeServiceProvider.client`.
9. Update the HTTP shared resource configuration and verify the Host and Port values and OUTPUT_FILE global variable.

   If you updated **HTTPConnector.httpConnResource** in the project, you must regenerate the concrete WSDL and import it into the 5.x client project.
10. In TIBCO Designer, click **Tools > Tester > Start**.
11. In TIBCO Business Studio™, click the **Terminate** icon to stop the process.

Result

For the getCityInfo service, information corresponding to the zip code defined in the ZIP field is returned. The default value is 61801, which returns information about Urbana, IL.

For the getCityDistance service, information corresponding to the distance between two ZIP codes is returned. The default values are 61801 for Urbana IL and 61820 for Champaign IL.

Understanding the Configuration

The project contains the **ZipCodeService** process that provides city information about a ZIP code. HTTP is used for the transport. The project represents a simple HTTP based service using SOAP. The **getCityDistance** and **getCityInfo** operations are implemented in the **ZipCodeService** process.

Implementing a JMS Service That Writes Attachment to File

This sample reads a GIF file as a binary data on the Consumer (client) and sends the request over JMS transport with attachment configured in **Set Context** activity. The message format is a RPC-literal element. The SOAP Service receives the message and writes it to the disk. The Service process then reads the file from the disk and sends it back to the SOAP Consumer (client). Finally, the SOAP Consumer (Reference binding) writes the same file to the disk.

Before you begin

TIBCO Enterprise Message Service must be running.

Copy the `ImageInput1.gif file` from the `TIBCO_HOME\bw\n.n\samples\binding\soap\jms\AttachmentWriteToFile` location to the `c:\tmp` folder on Windows or `/tmp` on UNIX. This file is used as an input attachment.

In TIBCO EMS administration, create `queue.sample` before executing the sample.

Procedure

1. In the samples directory, select **binding > soap > jms > AttachmentWriteToFile** and double-click **tibco.bw.sample.binding.soap.jms.AttachmentWriteToFile**. For more information, see Accessing Samples.
2. In Project Explorer expand the **tibco.bw.sample.binding.soap.jms.AttachmentWriteToFile** project.
3. Fully expand the **Processes** directory and double-click **Server.bwp**.
4. Verify your TIBCO Enterprise Message Service connection.
    a. Fully expand the **Resources** directory.
    b. Double-click **JMSConnectionResource.jmsConnResource**.
    c. In the Basic Configuration dialog, click **Test Connection** to verify the connection.
5. Click **Run > Debug Configurations**.
6. At the left hand tree of Debug Configuration wizard, expand **BusinessWorks Application** and select **BWApplication**.
7. Click the **Applications** tab and then click the **Deselect All** button if you have multiple applications. Select the checkbox next to **tibco.bw.sample.binding.soap.jms.AttachmentWriteToFile.application**.
8. Click **Debug**.

    This runs the sample in Debug mode.
9. In TIBCO Business Studio™, click the **Terminate** icon to stop the process.

Result

the following information:

Sent file size : 2514 bytes. Received file size : 2514 bytes.

Understanding the Configuration

SOAP Consumer (Client) sends the binary data using context over JMS transport. Service receives the attachment data and writes the attachment to the disk at location specified in SENT_RECV_FILES_WTF property. Threshold size WRITE_THRESHOLD and WriteToFile Directory is set on both Service and Reference binding.

The Consumer (Client) receives the attachment data from the Service using context and writes the content to disk. Both Service Provider and Consumer (Client) write data to the disk based on the Binding Persistence Configuration.

Note: Generated attachment files, that is, `printServer.gif` and `printClient.gif` on Service Provider and Consumer (client) must be identical to the provided INPUT_FILE that is specified in the Module Properties.

Sending a One Way Message over JMS

The SOAP Invoke activity is used to send a one way message over JMS. The message format is a doc-literal element. Because this is a one way message, no response is expected from the server.

Before you begin

TIBCO Enterprise Message Service must be running.

Procedure

1. In the **samples** directory, select **binding > soap > jms > OneWayDocLiteral** and double-click **tibco.bw.sample.binding.soap.jms.OneWayDocLiteral**. For more information, see Accessing Samples.
2. In **Project Explorer** expand the **tibco.bw.sample.binding.soap.jms.OneWayDocLiteral** project.
3. Fully expand the **Processes** directory and double-click **JMSClient.bwp**.

4. Verify your TIBCO Enterprise Message Service connection.
    a. Fully expand the **Resources** directory.
    b. Double-click **JMSConnectionResource.jmsConnResource**.
    c. In the Basic Configuration dialog, click the **Test Connection** button to verify the connection.
5. Click **Run > Debug Configurations**.
6. At the left hand tree of Debug Configuration wizard, expand **BusinessWorks Application** and select **BWApplication**.
7. Click the **Applications** tab and then click the **Deselect All** button if you have multiple applications. Select the checkbox next to **tibco.bw.sample.binding.soap.jms.OneWayDocLiteral.application**.
8. Click **Debug**.

    This runs the sample in Debug mode.
9. In TIBCO Business Studio™, click the **Terminate** icon to stop the process.

Result

The console displays messages similar to the following:

16:11:19.386 INFO [bwThread:In-Memory STWorkProcessor-2] c.t.b.p.g.L.t.b.s.b.s.j.O.Log - [Server] Received Request :: date=2016-04-21+05:30 boolean=true double=1.1 int=22 string=testStr byte=33 short=44

The expected outcome is:

[Server] Received Request :: date=2016-04-21+05:30 boolean=true double=1.1 int=22 string=testStr byte=33 short=44

Understanding the Configuration

The project comprises the following two processes:

- JMSClient
- JMSServiceProcess

The JMSClient process sends a one way request to the JMSServiceProcess through a SOAP doc-literal message.

Using JMS Destination Bridges

This sample illustrates the support of JMS destination bridges in TIBCO ActiveMatrix BusinessWorks™.

Before you begin

- TIBCO Enterprise Message Service must be running.
- In TIBCO EMS administration, create `queue.sample` and `topic.sample` before executing the sample.
- In TIBCO EMS administration, create the `GenericConnectionFactory` factory before executing the sample.
- A bridge must be created between Topic to Queue, that is, between `topic.sample` and `queue.sample`.
- Create a queue replyQueue.

Procedure

1. In the samples directory, select **binding > soap > jms > TopicQueueBridge** and double-click **tibco.bw.sample.binding.soap.jms.TopicQueueBridge**. For more information, see Accessing Samples.
2. In **Project Explorer** expand the **tibco.bw.sample.binding.soap.jms.TopicQueueBridge** project.
3. Verify your TIBCO Enterprise Message Service connection.
    a. Fully expand the **Resources** directory.
    b. Double-click **JMSConnectionResource.jmsConnResource**.
    c. In the Basic Configuration dialog, click the **Test Connection** button to verify the connection.
4. Fully expand the **Processes** directory and double-click **client.bwp**.
5. Click **Run > Debug Configurations**.
6. At the left hand tree of Debug Configuration wizard, expand **BusinessWorks Application** and select **BWApplication**.
7. Click the **Applications** tab and then click the **Deselect All** button if you have multiple applications. Select the checkbox next to

tibco.bw.sample.binding.soap.jms.TopicQueueBridge.application.

8. Click **Debug**.

   This runs the sample in Debug mode.

9. Click the **Terminate** icon to stop the process.

Result

The Console shows the following response:
Request Received.

Response received with value :0.5

Understanding the Configuration

The process sends a message to the Topic and is receives from the Queue using SOAP over JMS.

This process runs as follows:

- Client sends a request.
- WS Service receives the request and calls the Service process. The Service process processes the data and returns the response.

Implementing a JMS Service that Returns Information Based on Zip Codes

The JMS service returns information about a city, given a zip code. It also provides the distance between two cities as defined by their zip codes.

Before you begin

TIBCO Enterprise Message Service must be running.

Procedure

1. In the **samples** directory, select **binding > soap > jms > ZipCodeLookup** and

double-click **tibco.bw.sample.binding.soap.jms.ZipCodeLookup**. For more information, see Accessing Samples.

2. In **Project Explorer** expand the **tibco.bw.sample.binding.soap.jms.ZipCodeLookup** project.
3. Fully expand the **Processes** directory and double-click **ZipInfoService.bwp**.
4. Verify your TIBCO Enterprise Message Service connection.
    a. Fully expand the **Resources** directory.
    b. Double-click **JMSConnectionResource.jmsConnResource**.
    c. In the Basic Configuration dialog, click the **Test Connection** button to verify the connection.
5. Click **Run > Debug Configurations**.
6. At the left hand tree of Debug Configuration wizard, expand **BusinessWorks Application** and select **BWApplication**.
7. Click the **Applications** tab and then click the **Deselect All** button if you have multiple applications. Select the checkbox next to **tibco.bw.sample.binding.soap.jms.ZipCodeLookup.application**.
8. Click **Debug**.

    This runs the sample in Debug mode.
9. In TIBCO Business Studio™, click the **Terminate** icon to stop the process.

Result

For the **getCityInfo** service, information corresponding to the zip code defined in the **zip** field is returned. The default value is 61801, which returns information about **Urbana, IL**.

For the **getCityDistance** service, information corresponding to the distance between two zip codes is returned. The default values are 61801 for **Urbana IL** and 61820 for **Champaign IL**.

Understanding the Configuration

The project contains the **ZipInfoService** process that provides city information about a zip code. JMS is used for the transport. The project represents a simple JMS based service

using SOAP. The **getCityDistance** and **getCityInfo** operations are implemented in the **ZipInfoService** process.

Implementing a JMS Service That Returns an Attachment

The JMS service returns information about a city, given a zip code as an attachment.

Before you begin

TIBCO Enterprise Message Service must be running.

Copy the `CityInfo.xml` from `TIBCO_HOME\bw\n.n\samples\binding\soap\jms\ZipCodeLookupWithAttachment` file to the `c:\tmp` folder. The file has city information and is used to send as the attachment.

In TIBCO EMS administration, create `queue.sample` and `topic.sample` before executing the sample.

Procedure

1. In the **samples** directory, select **binding > soap > jms > ZipCodeLookupWithAttachment** and double-click **tibco.bw.sample.binding.soap.jms.ZipCodeLookupWithAttachment**. For more information, see Accessing Samples.
2. In **Project Explorer** expand the **tibco.bw.sample.binding.soap.jms.ZipCodeLookupWithAttachment** project.
3. Set the default **ApplicationProfile** to match the OS you are running on. For more information, see Setting the Default Application Profile.
4. Fully expand the **Processes** directory and double-click **ZipInfoService.bwp**.
5. Verify your TIBCO Enterprise Message Service connection.
    a. Fully expand the **Resources** directory.
    b. Double-click **JMSConnectionResource.jmsConnResource**.
    c. In the Basic Configuration dialog, click **Test Connection** to verify the connection.

6. Click **Run > Debug Configurations**.
7. At the left hand tree of Debug Configuration wizard, expand **BusinessWorks Application** and select **BWApplication**.
8. Click the **Applications** tab and then click the **Deselect All** button if you have multiple applications. Select the checkbox next to **tibco.bw.sample.binding.soap.jms.ZipCodeLookupWithAttachment.application**.
9. Click **Debug**.

   This runs the sample in Debug mode.
10. In TIBCO Business Studio™, click the **Terminate** icon to stop the process.

Result

The `output.log` file at `C:\tmp\ZipCodeLookupWithAttachment` location shows the following output:

Zip Code : 94086

City : SUNNYVALE

State : CA

Core Features

This section includes a set of samples that demonstrate core features, including using shared modules, managing faults, invoking subprocesses and so on.

Using Process and Module Properties

This sample demonstrates the use of process and module properties.

Procedure

1. In the **samples** directory, select **core > config > BasicPropertyConfiguration** and double-click **tibco_bw_sample_core_config_processandmoduleproperties.zip**. For more information, see Accessing Samples.
2. In **Project Explorer**, expand the **tibco_bw_sample_core_config_ processandmoduleproperties** project.
3. Click **Run > Debug Configurations**.
4. At the left hand tree of Debug Configuration wizard, expand **BusinessWorks Application** and select **BWApplication**.
5. Click the **Applications** tab and then click the **Deselect All** button if you have multiple applications. Select the checkbox next to **tibco_bw_sample_core_config_ processandmoduleproperties_application**.
6. Click **Debug**. This runs the sample in Debug mode.

   The **WriteFileMProp** activity writes to the file name that is defined in the **MP_ FileName1** Module Property. The **WriteFilePProp** activity writes to the file name that is defined in the **PP_FileName2** Process Property. By default, this value is set to `C:\tmp\BasicPropertyConfiguration\fileProcessPropDefault.log`.
7. Click the **Terminate** icon to stop the process.

Result

The console prints a message similar to the following:

INFO [bwThread:In-Memory STWorkProcessor-1] c.t.b.p.g.L.t.b.s.c.c.b.p.Log - -->Completed WriteToFileProcessA Job: bw0a100

Using a Process Conversation to Correlate JMS Message Jobs

JMS client and server processes correlate with each other by sending and receiving JMS messages through the process conversation mechanism.

Before you begin

Use TIBCO ActiveMatrix BusinessWorks™ 6.x server. However, if you want to run ActiveMatrix BusinessWorks™ 5.x server, install ActiveMatrix BusinessWorks 5.x.

TIBCO Enterprise Message Service must be running.

Procedure

1. In the **samples** directory, select **core > conversation > MultiJobsCorrelation** and double-click **tibco.bw.sample.core.conversation.MultiJobsCorrelation**. For more information, see Accessing Samples.
2. In the **Project Explorer**, expand the **tibco.bw.sample.core.conversation.MultiJobsCorrelation** project.
3. Fully expand **tibco.bw.sample.core.conversation.MultiJobsCorrelation.application** and double-click **Properties**. The output file location defined for the application is displayed in the dialog.
4. Ensure that the TIBCO Enterprise Message Service server is up and running.
5. Click **Run > Debug Configurations**.
6. At the left hand tree of Debug Configuration wizard, expand **BusinessWorks Application** and select **BWApplication**.
7. Click the **Applications** tab and then click the **Deselect All** button if you have multiple applications. Select the checkbox next to **tibco.bw.sample.core.conversation.MultiJobsCorrelation.application**.
8. Click **Debug**.

This runs the sample in Debug mode.

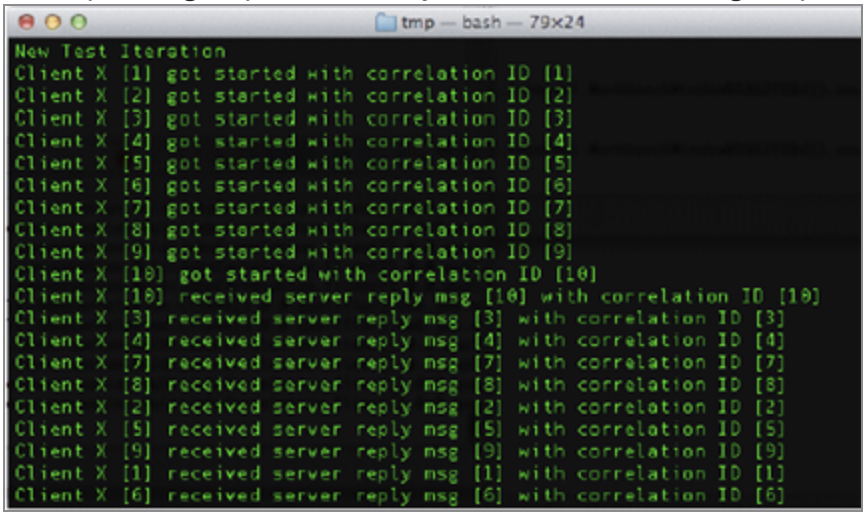
9. Click the **Terminate** icon to stop the process.
10. For the server created in ActiveMatrix BusinessWorks 5.x:
    a. In TIBCO Business Studio™ for BusinessWorks™, select **Module Descriptors > Components** and remove **Receiver.bwp**.
    b. Repeat steps 6 through 8 from above.
    c. Unzip, open, and run **bw5/tibco.bw.sample.core.conversation.MultiJobsCorrelation.Server** in the TIBCO Designer 5.x Tester.

Result

From the Debugger, expand BWNode to see the jobs created. Jobs are completed in the order of server side reply and not in the order of their creation. The following image shows the resulting console screen.

On UNIX, you can also verify the expected result by tailing the client output to the file `/tmp/MultiJobsCorrelation/MultiJobsCorrelation.log`.

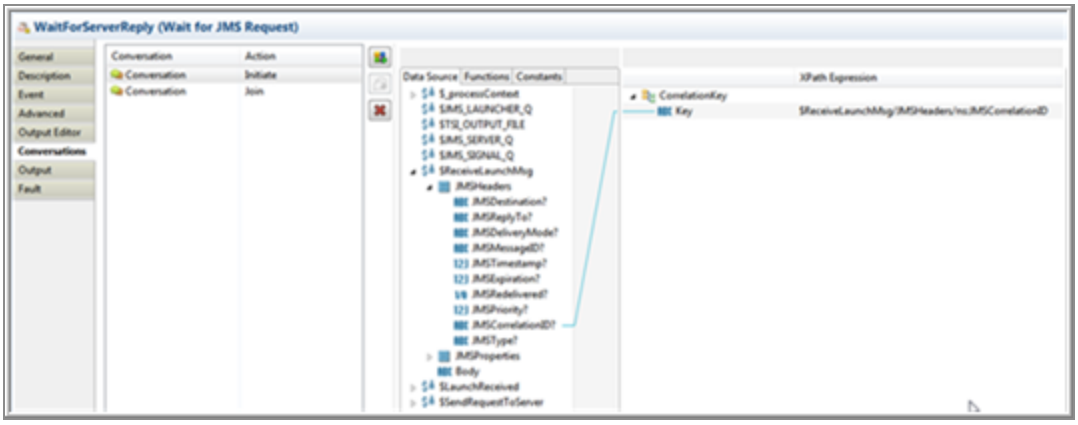
The `MultiJobsCorrelation.log` at the `C:\tmp\MultiJobsCorrelation` location shows requests 1 to 10 when sent in order with the corresponding correlation IDs. The server response is received out of order and the runtime engine correlates each reply to the corresponding request correctly. You see the following output.



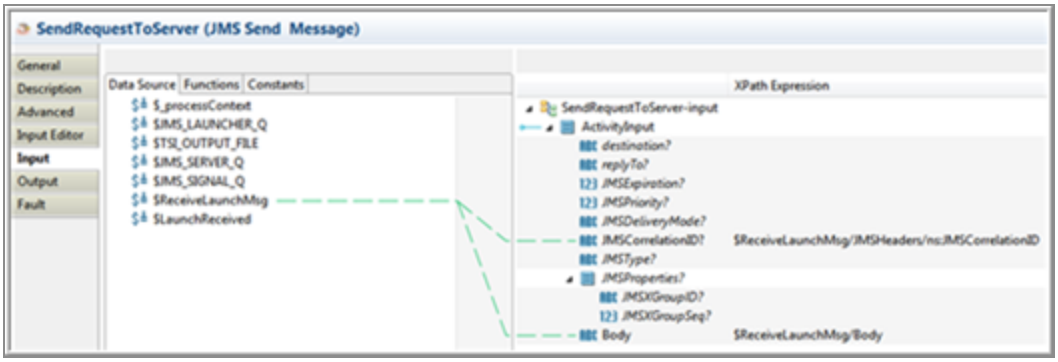
Understanding the Configuration

A conversation is configured with an Initiator and a Joiner. Click the **WaitForServerReply** activity (a JMS Signal-In activity).

Click the **Conversations** tab. In this case, **WaitForServerReply** is both the **Initiator** and **Joiner** for this conversation.



The **WaitForServerReply** activity's Initiate action is configured with *$ReceiveLaunchMsg/JMSHeader/ns:JMSCorrelationID* where **SendRequestToServer** activity Input is also configured with the same *JMSCorrelationID*. See the following.



The reason for this configuration is on the ActiveMatrix BusinessWorks 5.x JMSServerProcess side. The **Reply To Client** activity is configured to send back the same JMSCorrelationID, for the Reply to match with the correct Server Request job.

Using Modularity with the Java Palette

Three application modules and one shared module demonstrate calling from a Java Invoke activity and an Invoke activity through a Call Process.

Before you begin

1. For the Java invoke to work, create a new application on https://apps.twitter.com/ and generate Access Token, Access Token Secret, Consumer Key, Consumer Secret for the app created, and use it in the parameters for SearchTwitter activity.
2. To create a token:
    a. Login to https://apps.twitter.com/ with your Twitter account.
    b. Click **Create New App**.
    c. Provide **Name** and **Description** and create your Twitter application. Your application is successfully created.
    d. Go to the **API Keys** tab and click **Create my access token**. Your application access token is successfully generated.
    e. Click **Test OAuth**. The OAuth Settings page is generated with the following tokens:
        - Consumer key
        - Consumer secret
        - Access token
        - Access token secret

Procedure

1. In the **samples** directory, select **core** > **modularity** > **ProcessAndJavaCodeModularity**. The following projects are visible:
    - **tibco.bw.sample.core.modularity.ProcessAndJavaCodeModularity.app**
    - **tibco.bw.sample.core.modularity.ProcessAndJavaCodeModularity.importpackage**
    - **tibco.bw.sample.core.modularity.ProcessAndJavaCodeModularity.shared**
    - **tibco.bw.sample.core.modularity.ProcessAndJavaCodeModularity.subprocesscall**
2. First click **tibco.bw.sample.core.modularity.ProcessAndJavaCodeModularity.shared** and then click the remaining three projects.
3. In **Project Explorer**, expand the

tibco.bw.sample.core.modularity.ProcessAndJavaCodeModularity.app project. For more information, see Accessing Samples.

4. Fully expand the **Processes** directory and double-click **Process.bwp**.
5. Click **Run** > **Debug Configurations**.
6. At the left hand tree of Debug Configuration wizard, expand **BusinessWorks Application** and select **BWApplication**.
7. Click the **Applications** tab and then click the **Deselect All** button if you have multiple applications. Select the checkbox next to **tibco.bw.sample.core.modularity.ProcessAndJavaCodeModularity.app.application**.
8. Click **Debug**. This runs the sample in Debug mode.

   The console window shows the log messages such as application module started successfully and application started successfully.
9. Run **tibco.bw.sample.core.modularity.ProcessAndJavaCodeModularity.importpackage.application** and **tibco.bw.sample.core.modularity.ProcessAndJavaCodeModularity.subprocesscall.application** in a similar way, one after the other.

Understanding the Configuration

The sample has three Application Modules and one Shared Module:

- **tibco.bw.sample.core.modularity.ProcessAndJavaCodeModularity.app**: This is an application module that contains the TIBCO BusinessWorks ™ Processes. It uses the Java invoke in the shared module **tibco.bw.sample.core.modularity.ProcessAndJavaCodeModularity.shared**. The application module refers to the shared module using the Require-Bundle Manifest header. The sample **Java Invoke** activity uses the Java invoke to connect to Twitter to search for the string "Lady Gaga". Twitter returns a payload of tweets matching the search string, which the **Write To File** activity then writes to a file in a temporary directory.
- **tibco.bw.sample.core.modularity.ProcessAndJavaCodeModularity.importpackage** : This is similar to the

tibco.bw.sample.core.modularity.ProcessAndJavaCodeModularity.app module, except here the application module uses **Imported Packages** instead of **Required Modules** in **Module Descriptors > Dependencies**.

- **tibco.bw.sample.core.modularity.ProcessAndJavaCodeModularity.subprocesscall** : This is an application module that calls the web service hosted in the shared module.
- **tibco.bw.sample.core.modularity.ProcessAndJavaCodeModularity.shared**: This is a shared module that contains code to make a REST call to Twitter to search for a specified search term. Since this is a shared module it cannot be executed by itself. However, it can be called from an application module such as `tibco.bw.sample.core.modularity.ProcessAndJavaCodeModularity.app`, `tibco.bw.sample.core.modularity.ProcessAndJavaCodeModularity.importpackage`, and `tibco.bw.sample.core.modularity.ProcessAndJavaCodeModularity.subprocesscall`. This functionality is exposed by the shared module both as a Java class and a web service, which can be invoked by an Invoke activity.

Process

Scenario 1

Modules Executed

- **tibco.bw.sample.core.modularity.ProcessAndJavaCodeModularity.shared**
- **tibco.bw.sample.core.modularity.ProcessAndJavaCodeModularity.app**

This scenario demonstrates calling Java invoke (in a Shared Module) from a **Java Invoke** activity in an Application module. The dependency on the Java package in the Shared Module is modeled using **Required Modules** in **Module Descriptors > Dependencies**.

Scenario 2

Modules Executed

- **tibco.bw.sample.core.modularity.ProcessAndJavaCodeModularity.shared**
- **tibco.bw.sample.core.modularity.ProcessAndJavaCodeModularity.importpackage**

This scenario shows how to call the Java invoke (in a shared module) from a **Java Invoke** activity in an application module. The dependency on the Java package in the shared module is modeled using **Import-Package** in **Module Descriptors > Dependencies**.

Scenario 3

Modules Executed

- **tibco.bw.sample.core.modularity.ProcessAndJavaCodeModularity.shared**
- **tibco.bw.sample.core.modularity.ProcessAndJavaCodeModularity.subprocesscall**

This scenario shows how to call the Java invoke (in a shared module) from an **Invoke** activity in the application module through a **Call Process**. There is no Java package or bundle dependency on the shared module.

The following sub process is in the shared module:

TwitterSearch SubProcess: This subprocess is invoked from `tibco.bw.sample.core.modularity.ProcessAndJavaCodeModularity.subprocesscall`. In this subprocess, there are two JavaInvoke Activities - OpenTwitterConnection and SearchTwitter. The source code for these activities is located under `src` directory under `com.example.TwitterConnection` package in the same project.

OpenTwitterConnection activity creates a TwitterConnection object which is then passed to SearchTwitter activity. SearchTwitter activity requires the following 5 input parameters:

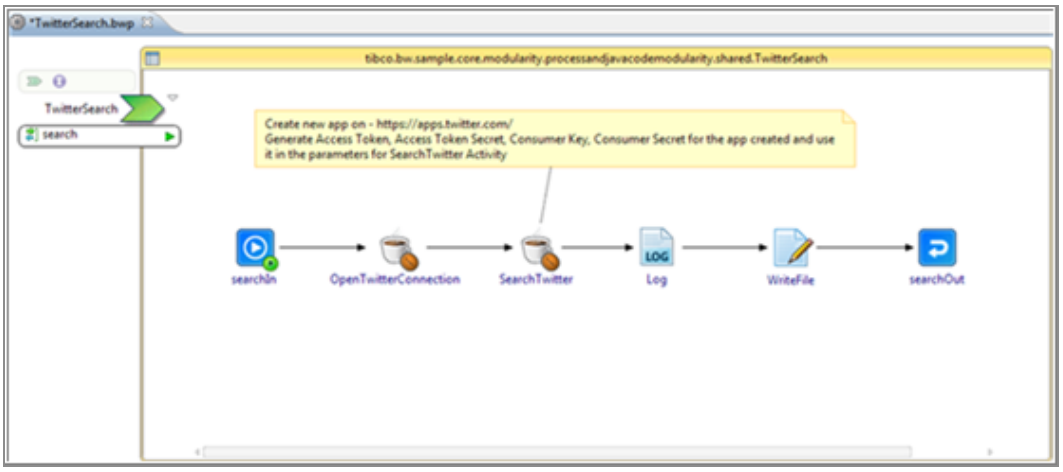
- SearchString
- AccessToken
- AccessTokenSecret
- ConsumerKey
- ConsumerSecret

SearchString signifies the string used by the user to search tweets. In this project, the SearchString is "Lady Gaga". This SearchString is passed from the main process in **tibco.bw.sample.core.modularity.ProcessAndJavaCodeModularity.subprocesscall**.

AccessToken, **AccessTokenSecret**, **ConsumerKey**, and **ConsumerSecret** are required for authentication to the Twitter Account.

The steps to generate these are described in the **Prerequisites** section. You can change the values of these module properties from **Module Descriptors** > **Module Properties**.

> **Note:** The Java invoke uses Twitter API v1.1 which requires the request to be authenticated using OAuth.



The process for Scenario 1 and Scenario 2 is very similar to the subprocess of Scenario 3, except that Scenario 1 and 2 have a Timer activity as a process starter instead of a service.

Troubleshooting

Note: If you open Application Modules especially subprocesscall application module before the shared module, then you may encounter errors such as Import Configuration Error, Process Reference configuration Error, Application configuration Error, and so on.

- If you receive the following error message:

  `"401:Authentication credentials (https://dev.twitter.com/docs/auth) were missing or incorrect. Ensure that you have set valid consumer key/secret, access token/secret, and the system clock is in sync."`

  make sure you have generated Access Token, Access Token Secret, Consumer Key, Consumer Secret as per the steps specified in the **Prerequisites** section

- If you see any problem markers in the project, ensure `tibco.bw.sample.core.modularity.ProcessAndJavaCodeModularity.shared` is imported into the workspace as all other application modules depend on this shared module.

Configuring a Conversation to Join a Wait for File Change Activity

The **Wait For File Change** activity initiates and joins conversation with a File Poller activity to implement a process conversation.

Procedure

1. In the **samples** directory, select **core > conversation > WaitForFileChange** and double-click **tibco.bw.sample.core.conversation.WaitForFileChange**. For more information, see Accessing Samples .
2. In the **Project Explorer** view, expand the **tibco.bw.sample.core.conversation.WaitForFileChange** project.
3. Fully expand **tibco.bw.sample.core.conversation.WaitForFileChange.application** and double-click **Properties**. The output file locations defined for the application displays in the dialog.
4. Click **Run > Debug Configurations**.
5. At the left hand tree of Debug Configuration wizard, expand **BusinessWorks Application** and select **BWApplication**.
6. Click the **Applications** tab and then click the **Deselect All** button if you have multiple applications. Select the checkbox next to **tibco.bw.sample.core.conversation.WaitForFileChange.application**.
7. Click **Debug**.

   This runs the sample in Debug mode.
8. After completing the Scenario1 and 2, click the **Terminate** icon to stop the process.

Result

The request is Approved.

Understanding the Configuration

Procedure

1. `Trigger.bwp` process creates `Trigger.log` and this triggers a job.
2. `Trigger.bwp` process creates `Req.log` and writes "Approved" in the same file. This triggers event for the "Wait For File Change".
3. When the **Wait for File Change** activity runs, the process instance suspends and waits for the specified change to occur before resuming.

   Output file gets created at: `c:\tmp\WaitForFileChange\WaitForFileChange.log` with content: "The request is Approved."
4. The **RemoveApprovalFile** activity removes the `Req.log` file from `c:\tmp\WaitForFileChange`.
5. **TriggerAnotherJob**: This activity triggers another job and process execution waits at **Wait For File Change** activity for matching the correlation keys.
6. If the user creates `Req.log` file at location `c:\tmp\WaitForFileChange` with content "Rejected", then conversation gets executed successfully and the output file at `c:\tmp\WaitForFileChange\WaitForFileChange.log` gets updated as mentioned below:

   The request is Approved.

   The request is Rejected.

Using an Asynchronous Event Handler to Implement a Simple Mortgage Workflow

Mortgage application consumer and provider processes interact with each other through the message correlation mechanism. The event handler invocation allows for asynchronous event processing and runs the logic parallel to the main business logic of the process.

Procedure

1. In the **samples** directory, select **core > event > AsyncEventHandler** and double-click **tibco.bw.sample.core.event.AsyncEventHandler**. For more information, see

Accessing Samples.

2. In **Project Explorer**, expand the **tibco.bw.sample.core.event.AsyncEventHandler** project.
3. Fully expand the **Processes** directory and double-click the **MortgageAppConsumer.bwp** and the **MortgageAppProvider.bwp** processes.
4. Click **Run > Debug Configurations**.
5. At the left hand tree of Debug Configuration wizard, expand **BusinessWorks Application** and select **BWApplication**.
6. Click the **Applications** tab and then click the **Deselect All** button if you have multiple applications. Select the checkbox next to **tibco.bw.sample.core.event.AsyncEventHandler.application**.
7. Click **Debug**.

   This runs the sample in Debug mode.
8. Click the **Terminate** icon to stop the process.

Result

On the console, the log messages are displayed similar to the following:

12:19:45.639 [bwThread:In-Memory STWorkProcessor-2] INFO c.t.b.p.g.L.t.b.s.c.e.A.LogSubmitMortgage - ### Received SubmitMortgageApp()

12:19:45.670 [bwThread:In-Memory STWorkProcessor-4] INFO c.t.b.p.g.L.t.b.s.c.e.A.Log - Cancel Application F54734546

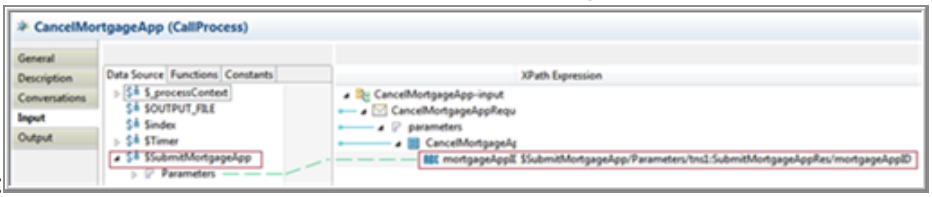
12:19:45.685 [bwThread:In-Memory STWorkProcessor-4] INFO c.t.b.p.g.L.t.b.s.c.e.A.Log - Cancel Application F54734546

12:19:45.685 [bwThread:In-Memory STWorkProcessor-4] INFO c.t.b.p.g.L.t.b.s.c.e.A.Log - Cancel Application F54734546

12:19:45.701 [bwThread:In-Memory STWorkProcessor-8] INFO c.t.b.p.g.L.t.b.s.c.e.A.Log - ### Done, get Mortgage Rate: 9.1

Understanding the Configuration

There are two processes in the supplied project:

- **MortgageAppProvider**: This is a Mortgage service with three operations, the **SubmitMortgageApp**, **SubmitFinDoc** and the **CancelMortgageApp** operations defined in the event handler container. Both the **CancelMortgageApp** and **SubmitFinDoc** operations have a correlation key obtained from the response to the operation **SubmitMortgageApp**.
- **MortgageAppConsumer**: The consumer invokes the **CancelMortgageApp** operation before the operation **SubmitFinDoc**: This way, the service calls **CancelMortgageApp** after receiving the correlation message key, *mortgageAppID*: The **CancelMortgageApp** activity has the input properties configured as shown in the following image: 

Using a Cross Process Conversation to Determine the Mortgage Rate

The process conversation mechanism is demonstrated using a mortgage rate calculation scenario that performs the following:

- A user passes the customer's name and credit profile information.
- If the credit profile value of the customer is A, the mortgage lending rate is 3.5 and if the credit profile is B, the lending rate is 2.0.
- Depending on the mortgage lending rate calculated, the lending institution decides either to accept or to reject the application. This particular lender decides to lend money only with the 3.5 mortgage rate. If the calculated rate is lower for any credit profile, the application is rejected by the lender.

Procedure

1. In the **samples** directory, select **core > conversation > CrossProcessConversation**

and double-click **tibco.bw.sample.core.conversation.CrossProcessConversation**. For more information, see Accessing Samples.

2. In the **Project Explorer** view, expand the **tibco.bw.sample.core.conversation.CrossProcessConversation** project.
3. Set the default **ApplicationProfile** to match the OS you are running on. For more information, see Setting the Default Application Profile.
4. Fully expand the **Processes** directory and double-click **MortgageAppConsumer.bwp**.
5. Click **Run > Debug Configurations**.
6. At the left hand tree of Debug Configuration wizard, expand **BusinessWorks Application** and select **BWApplication**.
7. Click the **Applications** tab and then click the **Deselect All** button if you have multiple applications. Select the checkbox next to **tibco.bw.sample.core.conversation.CrossProcessConversation.application**.
8. Click **Debug**.

   This runs the sample in Debug mode.
9. Click the **Terminate** icon to stop the process.

Result

The `CrossProcessConversation.log` output file at `C:\tmp\CrossProcessConversation` is created with the following content:

Received application from BWUser user with credit profile B

Mortgage Cancelled

DONE

The console displays the following log message:

12:20:42.735 INFO [bwEngThread:In-Memory STWorkProcessor-1] c.t.b.p.g.L.t.b.s.c.c.C.SubmitMortgageAppLog - Received application from BWUser user with credit profile B

12:20:42.779 INFO [bwEngThread:In-Memory STWorkProcessor-4] c.t.b.p.g.L.t.b.s.c.c.C.CancelMortgageAppLog - Mortgage Cancelled

12:20:42.818 INFO [bwEngThread:In-Memory STWorkProcessor-5] c.t.b.p.g.L.t.b.s.c.c.C.Log - DONE

Understanding the Configuration

Operations implemented in the Service Provider process are invoked from the Consumer process. The following processes are implemented:

- The **MortgageAppConsumer** process invokes different operations implemented in the **MortgageServiceProvider** process. The **SubmitFinDoc** or **CancelMortgage** operation is called, based on the mortgage rate returned by the **SubmitMortgageApp** operation for a customer.
- The **MortgageServiceProvicer** process implements the **SubmitMortgageApp**, **SubmitFinDoc**, or **CancelMortgage** operation. The **SubmitMortgageApp** operation generates and returns the mortgage app id and tentative mortgage rate, based on the customer's credit profile. The **SubmitFinDoc** operation submits the final mortgage documents. The **CancelMortgage** operation cancels the mortgage request.

For details about the conversation, select the **Conversation** tab for each of the following:

- The **SubmitMortgageAppOut** Reply activity for initiating the conversation.
- The **SubmitFinDocIn** Receive activity for joining the conversation.
- The **onEvent** Event Handler for joining the conversation.

A conversation is evaluated to **true** if the value in **Join** matches the **mortgageAppID** generated by the **SubmitMortgageApp** operation for each mortgage request. This way, the **mortgageAppID** is used as a key to process a mortgage request for a customer.

Catching Activity Faults with Scope and Fault Handlers

A scope and fault handler is defined for activities within a process. The handler catches faults that have been triggered within the scope.

If accessing the sample from the **Welcome** page of TIBCO Business Studio™ for BusinessWorks™, go directly to **Step 3**.

Procedure

1. In the **samples** directory, select **core > fault > ScopeAndFaultHandler** and double-

click **tibco.bw.sample.core.fault.ScopeAndFaultHandler.zip**. For more information, see Accessing Samples.

2. In the **Project Explorer**, expand the **tibco.bw.sample.core.fault.ScopeAndFaultHandler** project.
3. Set the default **ApplicationProfile** to match the OS you are running on. For more information, see Setting the Default Application Profile.
4. Fully expand the **Processes** directory and double-click **ScopeFault.bwp**.
5. Click **Run > Debug Configurations**.
6. At the left hand tree of Debug Configuration wizard, expand **BusinessWorks Application** and select **BWApplication**
7. Click the **Applications** tab and then click the **Deselect All** button if you have multiple applications. Select the checkbox next to **tibco.bw.sample.core.fault.ScopeAndFaultHandler.application**.
8. Click **Debug**.

   This runs the sample in Debug mode.
9. Click the **Terminate** icon to stop the process.

Result

The console displays the messages based on the value specified in "INPUT_XML_FILE" Module Property.

- If the correct location is specified for the `MortgageRatesDB.xml` file in "INPUT_XML_FILE" Module Property property, the message, The mortgage rate is 3.5 displays.
- If an incorrect location is specified for the `MortgageRatesDB.xml` file in "INPUT_XML_FILE" Module Property, the following error message displays:

  Caught FNF: < User_Defined_File_Location >\MortgageRatesDB.xml was not found.

  The same message is written in a `output.log` file at `C:\tmp\ScopeAndFaultHandler` location.

Modularity Among Applications with a Shared Module Project

This sample shows how modularity can be achieved among two application module projects and a shared module project.

Two scenarios in this sample show how a process in a shared module can be used by other application modules, without requiring changes to the contents of the shared module. The scenarios are demonstrated in the following projects:

Application Module Projects

- tibco.bw.sample.core.modularity.AppsWithSharedModule.App.HelloWorldConsumer
- tibco.bw.sample.core.modularity.AppsWithSharedModule.App.TimerEventComponent

Shared Module Project

- tibco.bw.sample.core.modularity.AppsWithSharedModule.SharedModule

Procedure

Scenario 1

1. In the **samples** directory, select **core > modularity > AppsWithSharedModule** and double-click **tibco.bw.sample.core.modularity.AppsWithSharedModule.SharedModule**. For more information, see Accessing Samples.
2. In **Project Explorer**, expand the **tibco.bw.sample.core.modularity.AppsWithSharedModule.App.HelloWorldConsumer**.
3. Set the default **ApplicationProfile** to match the OS you are running on. For more information, see Setting the Default Application Profile.
4. Fully expand the **Processes** directory and click **MainProcess.bwp**.
5. Click **Run > Debug Configurations**.
6. At the left hand tree of Debug Configuration wizard, expand **BusinessWorks Application** and select **BWApplication**
7. Click the **Applications** tab and then click the **Deselect All** button if you have multiple applications. Select the checkbox next to

tibco.bw.sample.core.modularity.AppsWithSharedModule.App.HelloWorldConsumer.application.

8. Click **Debug**.

   This runs the sample in Debug mode.

9. Click the **Terminate** icon to stop the process.

Scenario 2

10. In the **AppsWithSharedModule** directory, double-click the **tibco.bw.sample.core.modularity.AppsWithSharedModule.App.TimerEventComponent** project.
11. Click **Run > Debug Configurations**.
12. At the left hand tree of Debug Configuration wizard, expand **BusinessWorks Application** and select **BWApplication**
13. Click the **Applications** tab and then click the **Deselect All** button if you have multiple applications. Select the checkbox next to **tibco.bw.sample.core.modularity.AppsWithSharedModule.App.TimerEventComponent.application**.
14. Click **Debug**.

    This runs the sample in Debug mode.

15. Click the **Terminate** icon to stop the process.

Result

Scenario 1

The following messages are written by the shared module and application module to the `helloWorldServiceConsumer.log` file at `c:\tmp\AppsWithSharedModule` location.

- --> LogMessage from HelloWorldServiceProcess for John
- --> Hello John from HelloWorldServiceProcess

Scenario 2

The following messages are written by the shared module and application module to the `timerEvent.log` file at `c:\tmp\AppsWithSharedModule` location.

- --> LogMessage from TimerEventProcessTimerEventProcess
- --> Hello TimerEventProcess--> from LogMessage from TimerEventProcess

Understanding the Configuration

Scenario 1: The **MainProcess** included in the **tibco.bw.sample.core.modularity.AppsWithSharedModule.App.HelloWorldConsumer** application module, invokes the **HelloWorldServiceProcess** in the **tibco.bw.sample.core.modularity.AppsWithSharedModule.SharedModule**. The **Write To File** activity logs the output from the calling process, **HelloWorldServiceProcess**.

Scenario 2: The **Process** included in the **tibco.bw.sample.core.modularity.AppsWithSharedModule.App.TimerEventComponent** application module invokes the **TimerEventProcess** in the **tibco.bw.sample.core.modularity.AppsWithSharedModule.SharedModule**.

Load Balancing an Application Using an AppSpace

This process demonstrates the concept of an AppSpace for scaling an application.

Before you begin

- TIBCO Enterprise Message must be running.
- To run this sample, you can either use TIBCO ActiveMatrix BusinessWorks™ 6.x client or ActiveMatrix BusinessWorks™ 5.x as a client.
- PostgreSQL Database should be installed and running on the machine. Create tables by using the related scripts for the engine database.
- Sample uses default queue - 'queue.sample' of EMS server running on localhost.

Procedure

Setup the Domain

1. In a terminal, navigate to `TIBCO_HOME\bw\n.n\bin` and type bwadmin.
2. Create a Domain, AppSpace and AppNode:

```
bwadmin[admin]> create domain MyDomain
```

```
bwadmin[admin]> cd MyDomain
bwadmin[admin@MyDomain]> create appspace MyAppSpace
bwadmin[admin@MyDomain]> cd MyAppSpace
bwadmin[admin@MyDomain/MyAppSpace]> create -httpPort 9999 appnode
MyAppNode
```

Create the config.ini File

3. Copy `TIBCO_HOME\bw\n.n\config\appspace_config.ini_template` to a temporary location.
4. Rename the `appspace_config.ini_template` file to `config.ini`.
    a. Uncomment the following property and change the value to group.

    ```
    bw.engine.persistenceMode=group
    ```

    b. Set the datastore configuration with values similar to:

    ```
    # BW Engine Database Driver.
    bw.engine.db.jdbcDriver=org.postgresql.Driver

    # BW Engine Database URL.
    bw.engine.db.url=jdbc:postgresql://localhost:5432/postgres1

    # BW Engine Database User Name.
    bw.engine.db.userName=postgres

    # BW Engine Database User Password.
    bw.engine.db.password=root

    # BW Engine Database Connection Pool Size.
    bw.engine.db.maxConnections=10
    ```

    c. Save the `config.ini` file.

    For the **JMS Receive Message** activity, load balancing is done by setting **Max Jobs** limit and the flow limit. If one instance is down, the other instance processes the messages. Thus avoiding any outage. For the **HTTP Receiver** activity, load balancing is done at the external load balancer level. If one instance is down, all requests are sent to the other instance. However, when you use load balancing with fault tolerance, a database is required.
5. Push the configuration to the AppSpace running the following command in the

bwadmin terminal:

```
bwadmin[admin@MyDomain/MyAppSpace]> config -cf <temporary_
location>/config.ini
```

6. In the bwadmin terminal, upload and deploy the EAR file into the AppSpace:

```
bwadmin[admin@MyDomain/MyAppSpace]>upload
TIBCO_
HOME\bw\x.x\samples\AppSpace\core\admin\ears\samples\tibco.bw.sampl
e.core.scalability.LoadBalancingUsingAppSpace.application_1.0.0.ear
```

```
bwadmin[admin@MyDomain/MyAppSpace]> deploy -as

tibco.bw.sample.core.scalability.LoadBalancingUsingAppSpace.applica
tion_1.0.0.ear
```

7. Start the AppSpace using the following command:

```
bwadmin[admin@MyDomain/MyAppSpace]> start appspace MyAppSpace
```

Configure and Start the Client

8. Start TIBCO Designer and click **New Empty Project**.
9. Click **Project > Import** and import the client project, `TIBCO_HOME\bw\n.n\samples\core\scalability\LoadBalancingUsingAppSpace\bw5\tibco.bw.sample.core.scalability.LoadBalancingUsingAppSpace.client`.
10. Update the JMS shared resource configuration and verify the Host and Port values.
11. In TIBCO Designer, click **Tools > Tester > Start**. Wait for 30 seconds before completing the next steps.

Create a Second AppNode

12. In the bwadmin terminal, create a second AppNode in MyAppSpace. Run the create command to create the AppNode and then the start command to start the AppNode:

    The TIBCO ActiveMatrix BusinessWorks™ JMS application should be deployed to `MySecondAppNode` automatically by the AppSpace.

```
bwadmin[admin@MyDomain/MyAppSpace]> create -httpPort 9996 appnode
```

```
MySecondAppNode
bwadmin[admin@MyDomain/MyAppSpace]> start appnode MySecondAppNode
```

13. Open the `bwappnode.log` files on each node located under `<DomainLocation>/<appnodes>/<appspace>/<nodename>/log [Example: <DomainLocation>/appnodes/betaappspace/betaappnode1/log]` and monitor the messages flow.
14. Stop the second AppNode using the following command:

```
bwadmin[admin@MyDomain/MyAppSpace]> stop appnode MySecondAppNode
```

Result

Before stopping the second AppNode, the application is active on both the nodes. After stopping the second AppNode, the application is active only on the first AppNode. See the `bwappnode.log` file to monitor the messages processed by AppNode1.

Invoking Subprocesses

In this sample, the MakeNoise process calls the Invoke subprocess multiple times which in turn calls another subprocess Log.

Note: If accessing the sample from the Welcome page of TIBCO Business Studio™ for BusinessWorks™, go directly to **Step 3**.

Procedure

1. In the **samples** directory, select **core > soa > CallProcessAndInvoke** and double-click **tibco.bw.sample.core.soa.CallProcessAndInvoke**. For more information, see Accessing Samples.
2. In the **Project Explorer**, expand the **tibco.bw.sample.core.soa.CallProcessAndInvoke** project.
3. Set the default application profile to match the OS you are running on. For more information, see Setting the Default Application Profile.
4. Fully expand the **Processes** directory and double-click **MakeNoise.bwp**.

5. Click **Run > Debug Configurations**.
6. At the left hand tree of **Debug Configuration** wizard, expand **BusinessWorks Application** and select **BWApplication**.
7. Click the **Applications** tab and then click the **Deselect All** button if you have multiple applications. Select the checkbox next to **tibco.bw.sample.core.soa.CallProcessAndInvoke**.
8. Click **Debug**.

   This runs the sample in Debug mode.
9. Click the **Terminate** icon to stop the process.

Result

The `output.log` file is generated in the `C:\tmp\CallProcessAndInvoke` directory. This file contains output from MakeNoise process and Log sub process.

The Console shows the following response:
ECHO: 1:41:56 PM

Echo

ECHO: 1:41:59 PM

Echo

ECHO: 1:42:02 PM

Echo

Understanding the Configuration

This is a hands-on and simple demonstration for the Design time experience with simple Invoke services. The MakeNoise process calls the Invoke process with a single operation and a simple string message. The Invoke process simply returns the Input as the Output.

Dynamically Invoking Subprocesses Using SetEPR Activity

This sample illustrates the Dynamic Invocation of the Subprocess using **Set EPR** activity. The **Set EPR** activity generates the Endpoint Reference and determines the service to be invoked at runtime. In this sample, the Service Consumer and Service provider reside in different applications.

Setup

Import the sample project into the workspace. Ensure that both applications are in the workspace. For more information, see Accessing Samples.

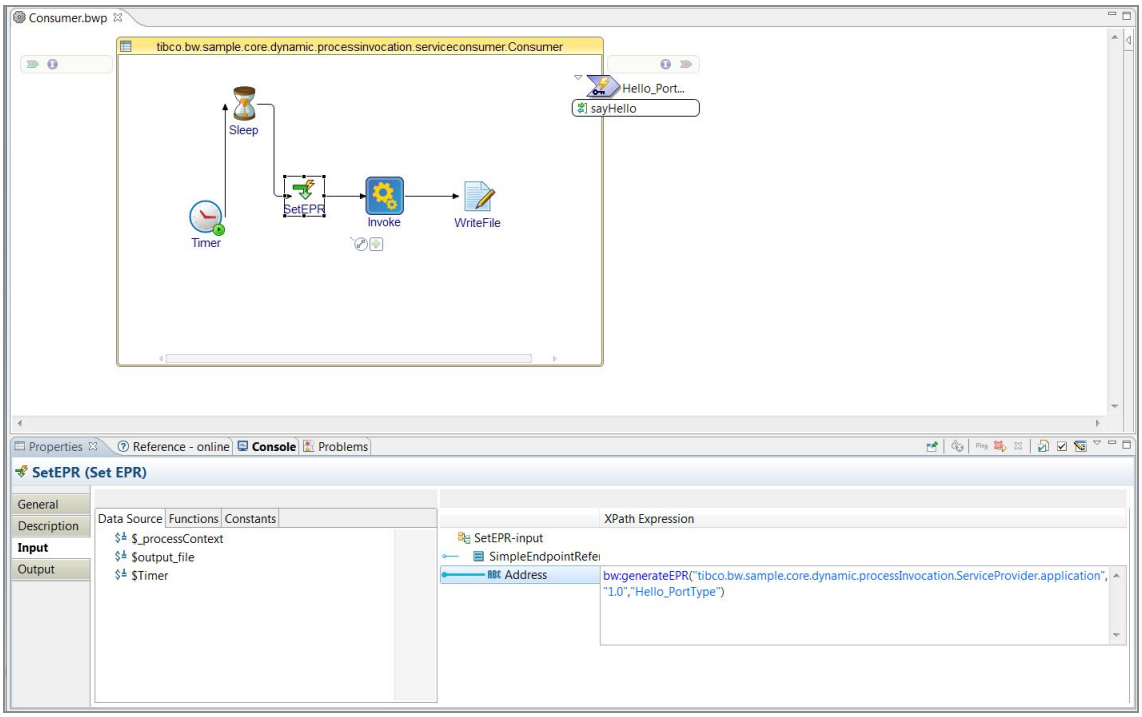
Description of the Project

This sample comprises two applications. Service Provider (**Provider.bwp**) and Service Consumer (**Consumer.bwp**).

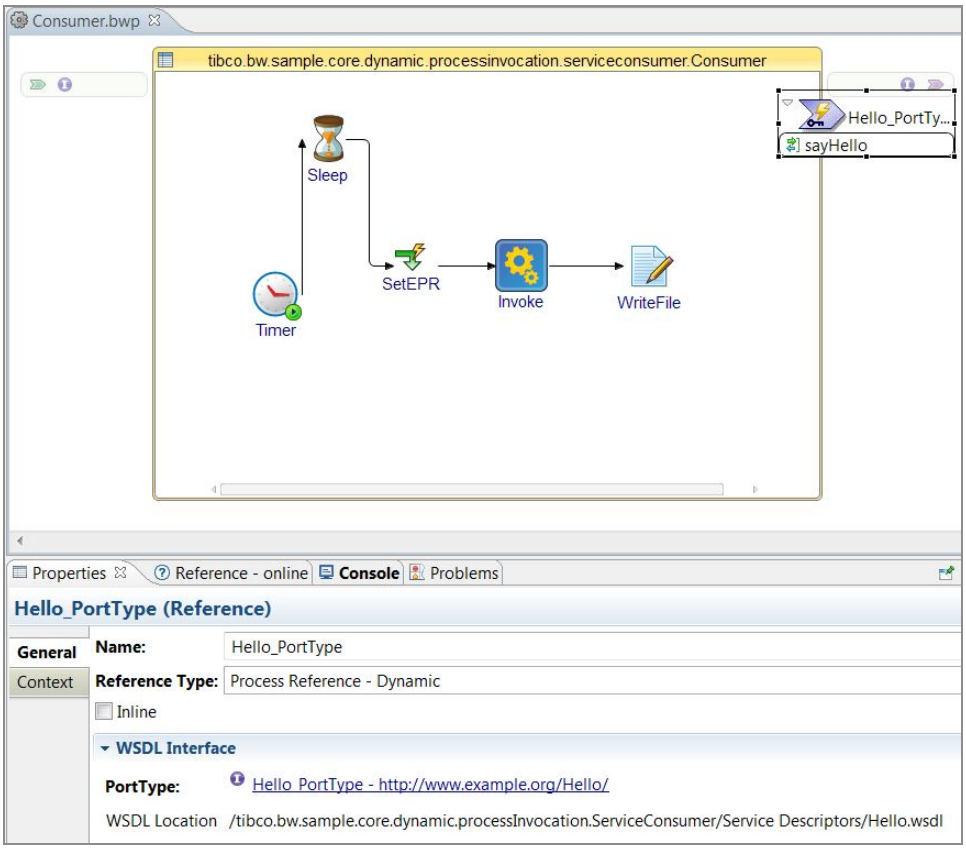
- The Service Consumer invokes the service using Dynamic Process Reference, which is located in the Service Provider application.
- Service Consumer has one process, **Consumer.bwp**. This process has the Dynamic Reference to the Service. The Consumer process contains a **Sleep** activity which ensures that consumer waits to invoke the service, till the service gets deployed and is ready to process the request.
- Service Provider has one process, **Provider.bwp**. This process uses the **Hello.wsdl** file to implement the service and sends a response message to the incoming request.

Note: Every application requires at least one component to deploy the application. In this sample, the **Activator.bwp** process is used in the service provider application for the **Provider.bwp** to be deployed without configuring the binding.

The following image shows the configuration details of the **Set EPR** activity in the **Consumer.bwp**.



The following image shows the details of consumer using the Dynamic Process Reference as the Reference Type.



Procedure

1. In the **samples** directory, select **core > dynamic > processInvocation** and double-click **tibco.bw.sample.core.dynamic.processInvocation.ServiceConsumer**.
2. In **Project Explorer**, expand the **tibco.bw.sample.core.dynamic.processInvocation.ServiceConsumer** project.
3. Fully expand the **Processes** directory and click **Consumer.bwp**.
4. Click **Run > Debug Configurations**.
5. At the left hand tree of Debug Configuration wizard, expand **BusinessWorks Application** and select **BWApplication**
6. Click the **Applications** tab and then click the **Deselect All** button if you have multiple applications. Select **tibco.bw.sample.core.dynamic.processInvocation.ServiceConsumer** and **tibco.bw.sample.core.dynamic.processInvocation.ServiceProvider** applications.
7. Click **Debug**.

This runs the sample in a Debug mode.

8. Click the **Terminate** icon to stop the process.

Troubleshooting

If the service consumer starts before the service provider, then increase the value in the **Sleep** activity in the **Consumer.bwp**.

Calling a Direct Subprocess

In this sample, the MainProcess process uses the **`Call Process`** activity to call two direct subprocesses. One subprocess creates a list of movies, and the other subprocess either uses the genre of the movie to determine how to log it, or returns the name of the current movie.

> **Note:** If accessing the sample from the Welcome page of TIBCO Business Studio™ for BusinessWorks™, go directly to **Step 3**.

Procedure

1. In the **samples** directory, select **core > soa > DirectCallProcess** and double-click **tibco.bw.sample.core.soa.DirectCallProcess**. For more information, see Accessing Samples.
2. In the **Project Explorer** view, expand the **tibco.bw.sample.core.soa.DirectCallProcess.readme** project.
3. Set the default **ApplicationProfile** to match the OS you are running on. For more information, see Setting the Default Application Profile.
4. Fully expand the **Processes** directory and double-click **MainProcess.bwp**.
5. Click **Run > Debug Configurations**.
6. At the left hand tree of Debug Configuration wizard, expand **BusinessWorks Application** and select **BWApplication**.
7. Click the **Applications** tab and then click the **Deselect All** button if you have multiple applications. Select the checkbox next to

tibco.bw.sample.core.soa.DirectCallProcess.

8. Click **Debug**.

   This runs the sample in Debug mode.

9. Click the **Terminate** icon to stop the process.

Result

The `output.log` file is generated in the `C:\tmp\CallProcess\ActionMovie.txt` directory. This file contains output from the ActionMovieList direct subprocess and OtherMoviesList direct subprocess.

Understanding the Configuration

This is a hands-on and simple demonstration for the design time experience with the direct subprocess. The **MainProcess.bwp** process creates a list of movies, then according to the movie genres, calls a direct subprocess to either log outputs to a text file, or to return the name of the current movie.

Collecting Process, Activity, and Transition Statistics

In this sample, use the **tibco.bw.sample.application.execution.event.subscribe** sample to collect statistics for process instances, activity instances, and transitions in the **tibco.bw.sample.palette.http.RequestResponse** sample project.

When working this sample, any of the following application statistics can be collected.

Process Instance Statistics

Statistic	Description
Application Name	Name of the application.

Statistic	Description
Application Version	Version of the application.
Module Name	Name of the module.
Module Version	Version of the module.
Component Process Name	Name of process configured to a component. If the process is a non in-lined sub process, this could be empty.
Job ID	Job ID of the process.
Parent Process Name	If the process is an in-lined sub process, the name of the parent process.
Parent Process ID	If the process is an in-lined sub process, the instance ID of the parent process.
Process Name	Name of the process.
Process Instance ID	Instance ID of the process.
Start Time	Process instance start time.
End Time	Process instance end time.
Elapsed Time	Elapsed time of an activity is the time difference (in milliseconds) between start time and end time of the activity. Between the start and end time, control may get switched with other activities from the other jobs. This is the time taken to run an activity plus all the delays in acquiring resources like engine threads, JDBC connections, network, and so on. The elapsed time includes the execution time plus time taken for evaluating all the forward transitions from that particular activity and getting the next activity ready to run, which includes executing its input mapping if all dependencies are met.

Statistic	Description
Eval Time	The Eval Time for an activity is the actual time (in milliseconds) used by the activity itself to complete while using the engine thread. Asynchronous activities may use other threads not included in this time.
Status	Status of process instance, for example: Completed or Faulted.

Activity Instance Statistics

Statistic	Description
Application Name	Name of the application.
Application Version	Version of the application.
Module Name	Name of the module.
Module Version	Version of the module.
Activity Name	Name of the activity.
Process Name	Name of the process.
Process Instance ID	Instance ID of the process.
Start Time	When the activity instance started.
End Time	When the activity instance ended.
Eval Time	The time between the beginning and end of the evaluation period for the activity. If the activity completes in one step, the evalTime and elapsedTime would be the same. However, some activities, such as **Request**, **Reply**, or **Wait for...** activities typically do not complete in one step.

Statistic	Description
Elapsed Time	Elapsed time of an activity is the time difference (in milliseconds) between start time and end time of the activity. Between the start and end time, control may get switched with other activities from other jobs. This is the time taken to run an activity plus all the delays in acquiring resources like engine threads, JDBC connections, network, and so on. The elapsed time is Eval Time plus the time taken for evaluating all the forward transitions from that particular activity.
Status	Status of activity, for example: Completed, Faulted or Canceled.

Transition Statistics

Statistic	Description
Application Name	Name of the application.
Application Version	Version of the application.
Module Name	Name of the module.
Module Version	Version of the module.
Transition Name	Name of the transition
Process Name	Name of the process.
Process Instance ID	Instance ID of the process.
Component Process Name	Name of process configured to a component. If the process is a non in-lined subprocess, this could be empty.
Target Activity Name	Name of the activity the transition targets.

Procedure

1. Import the sample into TIBCO Business Studio™ for BusinessWorks™ by right-clicking in the **Project Explorer** pane, and selecting **Import > Existing Studio Projects into Workspace**.
2. In the Import Projects window, ensure the option **Select root director** field is selected, and specify the location of the **tibco.bw.sample.application.execution.event.subscribe** sample. The sample is located at *BW_HOME*`/samples/source/event-subscriber/tibco.bw.sample.application.execution.event.subscriber`.
3. Click **Finish** to import the sample project.
4. In the **samples** directory, select **palette > http > RequestResponse** and double-click **tibco.bw.sample.palette.http.RequestResponse.zip**. For more information, see Accessing Samples.
5. In **Project Explorer** expand the **tibco.bw.sample.palette.http.RequestResponse** project.
6. Fully expand the **Processes** directory and double-click **HTTP_Request_Response_Example.bwp**.
7. Click **Run > Debug Configurations**.
8. At the left hand tree of Debug Configuration wizard, expand **BusinessWorks Application** and select **BWApplication**.
9. Click the **Applications** tab and then click the **Deselect All** button if you have multiple applications. Select the checkbox next to **tibco.bw.sample.palette.http.RequestResponse.application**.
10. Click the **Bundles** tab and ensure the following bundles in your workspace are selected:
    - **tibco.bw.sample.application.execution.event.subscriber (1.0.0.qualifer)**
    - **tibco.bw.sample.palette.http.RequestResponse (1.0.0.qualifer)**
    - **tibco.bw.sample.palette.http.RequestResponse.application (1.0.0.qualifer)**
11. Click **Debug**.

    This runs the sample in Debug mode.
12. Run lendpoints at the prompt in the **Console** tab to obtain the endpoint for the

application.

13. Copy the endpoint URL of the application.
14. Open a browser window, and paste the endpoint URL into the address bar.
15. Click the **Terminate** icon to stop the process.

Result

Details about process instances, activity instances, and transitions in **tibco.bw.sample.palette.http.RequestResponse** are displayed on the **Console** tab in TIBCO Business Studio for BusinessWorks.

```
<>@BWEclipseAppNode>
ProcessInstance Auditing Event {
       Application
Name:tibco.bw.sample.palette.http.RequestResponse.application
       Application Version:1.0
       Module Name:tibco.bw.sample.palette.http.RequestResponse
       Module Version:1.0.0.qualifier
       ProcessInstanceId:bw0a100
       Process Name:tibco.bw.sample.palette.http.requestresponse.HTTP_
Request_Response_Example
       State:SCHEDULED
}

ProcessInstance Auditing Event {
       Application
Name:tibco.bw.sample.palette.http.RequestResponse.application
       Application Version:1.0
       Module Name:tibco.bw.sample.palette.http.RequestResponse
       Module Version:1.0.0.qualifier
       ProcessInstanceId:bw0a100
       Process Name:tibco.bw.sample.palette.http.requestresponse.HTTP_
Request_Response_Example
       State:STARTED
}

Activity Auditing Event {
       Application
Name:tibco.bw.sample.palette.http.RequestResponse.application
       Application Version:1.0
       Module Name:tibco.bw.sample.palette.http.RequestResponse
       Module Version:1.0.0.qualifier
       ProcessInstanceId:bw0a100
       Process Name:tibco.bw.sample.palette.http.requestresponse.HTTP_
Request_Response_Example
```

```
       Activity Name:Incoming_HTTP_Request
       State:STARTED
}

Transition Auditing Event {
       Application
Name:tibco.bw.sample.palette.http.RequestResponse.application
       Application Version:1.0
       Module Name:tibco.bw.sample.palette.http.RequestResponse
       Module Version:1.0.0.qualifier
       ProcessInstanceId:bw0a100
       Process Name:tibco.bw.sample.palette.http.requestresponse.HTTP_
Request_Response_Example


Activity Auditing Event {
       Application
Name:tibco.bw.sample.palette.http.RequestResponse.application
       Application Version:1.0
       Module Name:tibco.bw.sample.palette.http.RequestResponse
       Module Version:1.0.0.qualifier
       ProcessInstanceId:bw0a100
       Process Name:tibco.bw.sample.palette.http.requestresponse.HTTP_
Request_Response_Example
       Activity Name:Incoming_HTTP_Request
       State:COMPLETED
}

Activity Auditing Event {
       Application
Name:tibco.bw.sample.palette.http.RequestResponse.application
       Application Version:1.0
       Module Name:tibco.bw.sample.palette.http.RequestResponse
       Module Version:1.0.0.qualifier
       ProcessInstanceId:bw0a100
       Process Name:tibco.bw.sample.palette.http.requestresponse.HTTP_
Request_Response_Example
       Activity Name:Log1
       State:STARTED
}
22:25:19.463 INFO  [bwEngThread:In-Memory Process Worker-1]
c.t.b.p.g.L.t.b.s.p.h.R.Log1 - No matching 'NEWS' source found.

Activity Auditing Event {
       Application
Name:tibco.bw.sample.palette.http.RequestResponse.application
```

```
       Application Version:1.0
       Module Name:tibco.bw.sample.palette.http.RequestResponse
       Module Version:1.0.0.qualifier
       ProcessInstanceId:bw0a100
       Process Name:tibco.bw.sample.palette.http.requestresponse.HTTP_
Request_Response_Example
       Activity Name:Log1
       State:COMPLETED
}

ProcessInstance Auditing Event {
       Application
Name:tibco.bw.sample.palette.http.RequestResponse.application
       Application Version:1.0
       Module Name:tibco.bw.sample.palette.http.RequestResponse
       Module Version:1.0.0.qualifier
       ProcessInstanceId:bw0a100
       Process Name:tibco.bw.sample.palette.http.requestresponse.HTTP_
Request_Response_Example
       State:COMPLETED
```

Tip: You can use this sample to build your own application statistics collection tool. Follow these steps to do this:

1. From the **Project Explorer** tab, select **tibco.bw.sample.application.execution.event.subscriber > src > tibco.bw.sample.application.execution.event.subscriber > BWEventSubscriber.java**.
2. Update handleEvent(Event event) method based on your use case.
3. Save your changes to the project.
4. Export the project as a plug-in by right-clicking on **tibco.bw.sample.application.execution.event.subscriber** and selecting **export > Export > Plug-in Development > Deployable plug-ins and fragments**.
5. In the Export wizard, ensure the **tibco.bw.sample.application.execution.event.subscriber** project is selected, and specify a location to export the plug-in.
6. After the project has been exported as a JAR file to the location you specified, locate the JAR file in the plugins folder, and copy and paste it to the shared folder at *BW_HOME*`/system/hotfix/shared`.
7. To load this plug-in into runtime, restart any running AppNodes.

Your application statistics collection tool has been added to your runtime environment.

Policy Feature

This section includes a set of samples that demonstrate how to enforce security policies to services and bindings.

Enforcing Basic Authentication with LDAP Authentication

This sample describes how access to an HTTP service can be managed by enforcing Basic Authentication on a process starter activity. In this sample, the Basic Authentication Policy is associated with the **HTTP Receiver** Activity and configured to verify user credentials using LDAP authentication.

Before you begin

You must be connected to an LDAP server and have user and group information stored in that server.

Procedure

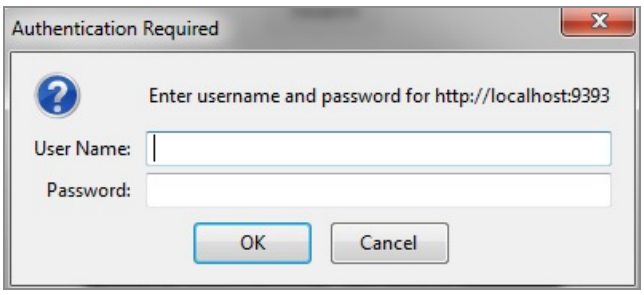
1. In the **samples** directory, select **policy > basicauthentication > HTTPReceiver** and double-click **tibco.bw.sample.policy.basicauthentication.HttpReceiver.zip**. For more information, see Accessing Samples.
2. In the **Project Explorer** view, expand the **tibco.bw.sample.policy.basicauthentication.HttpReceiver.zip** project.
3. In the **Project Explorer** view, click **Resources > tibco.bw.sample.policy.basicauthentication.httpReceiver > defaultLdapProvider.ldapResource**.
4. Fully expand the **Processes** directory and double-click **HttpReceiverProcess.bwp**.
5. In the Resource Editor, configure the following properties in the **LDAP Authentication** section under the **Connection** tab:

- **Server URL**: LDAP Server Location, such as, `ldap://localhost:389`
- **User Search Expression**: search expression for the user, such as `(&(cn={0])(objectclass=user))`
- **User DN Template**: user distinguished name template, such as `cn={0},cn=users,dc=na,DC=tibco,DC=com`

6. Set the default **ApplicationProfile** to match the OS you are running on. For more information, see Setting the Default Application Profile.
7. Fully expand the **Processes** directory and double-click **HttpReceiverProcess.bwp**.
8. Click **Run > Debug Configurations**.
9. At the left hand tree of Debug Configuration wizard, expand **BusinessWorks Application** and select **BWApplication**.
10. Click the **Applications** tab and then click the **Deselect All** button if you have multiple applications. Select the checkbox next to **tibco.bw.sample.policy.basicauthentication.HttpReceiver.application**.
11. Click **Debug**.

    This runs the sample in Debug mode.
12. Launch Google Chrome browser and enter the following URL http://localhost:9393/request.

    The Google Chrome page displays a window asking you to enter your user name and password.



13. Enter your LDAP user name and password and click **OK**.

Result

After the request is authenticated successfully, the following message displays in the browser:

This is from HTTP Receiver Process. The request is successful.

Understanding the Configuration

In the **HttpReceiverProcess.bwp** process, the Basic Authentication policy is configured to verify user credentials using LDAP Authentication and is associated with the **HTTP Receiver** activity.

Enforcing Basic Authentication on a REST Service Binding

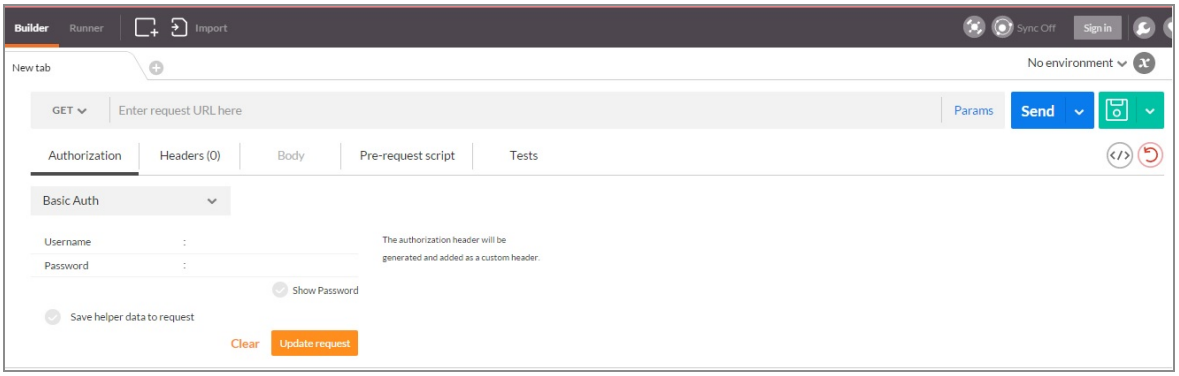
In this sample, the Basic Authentication policy is enforced on a REST service. The REST service uses the GET method to retrieve information from a file stored on your local machine.

Procedure

1. In the **samples** directory, select **policy > basicauthentication > RestBinding** and double-click **tibco.bw.sample.policy.basicauthentication.RestBinding.zip**. For more information, see Accessing Samples.
2. In **Project Explorer** expand the **tibco.bw.sample.policy.basicauthentication.restbinding** project.
3. Fully expand the **Processes** directory and double-click **RestBindingProcess.bwp**.
4. Set the default **ApplicationProfile** to match the OS you are running on. For more information, see Setting the Default Application Profile.
5. Click **Run > Debug Configurations**.
6. At the left hand tree of **Debug Configuration** wizard, expand **BusinessWorks Application** and select **BWApplication**.
7. Click the **Applications** tab and then click the **Deselect All** button if you have multiple applications. Select the checkbox next to **tibco.bw.sample.policy.basicauthentication.RestBinding.application**.
8. Click **Debug**.

   This runs the sample in Debug mode.
9. Launch a REST client, such as Postman Extension, on Chrome, and enter the request URL, for example http://localhost:7777/resources, in the **Enter request URL here**

field.



10. Click the **GET** operation.
11. Enter your user name and password under the **BasicAuth** section.
12. Click **Update Request**.
13. Click **Send** to test your application.

Result

GET books returns an output similar to the following:

```
<?xml version="1.0"?>
<purchaseOrder xmlns="http://tempuri.org/po.xsd" orderDate="1999-10-20">
    <shipTo country="US">
        <name>Alice Smith</name>
        <street>123 Maple Street</street>
        <city>Mill Valley</city>
        <state>CA</state>
        <zip>90952</zip>
    </shipTo>
    <billTo country="US">
        <name>Robert Smith</name>
        <street>8 Oak Avenue</street>
        <city>Old Town</city>
        <state>PA</state>
        <zip>95819</zip>
    </billTo>
    <comment>Hurry, my lawn is going wild!</comment>
    <items>
        <item partNum="872-AA">
            <productName>Lawnmower</productName>
            <quantity>1</quantity>
            <USPrice>148.95</USPrice>
```

```
            <comment>Confirm this is electric</comment>
        </item>
        <item partNum="926-AA">
            <productName>Baby Monitor</productName>
            <quantity>1</quantity>
            <USPrice>39.98</USPrice>
            <shipDate>1999-05-21</shipDate>
        </item>
    </items>
</purchaseOrder>
```

Understanding the Configuration

In this sample, a Basic Authentication policy is associated with the REST service binding and configured to verify user credentials using an XML Authentication resource.

The XML Authentication resource references an XML file that contains information about users, groups, and roles. A user is a person with an authenticated credentials. A group is a collection of users. Roles can be assigned to users to grant them permission to perform a collection of tasks. Authorization, or permission, to access services can be assigned to both users and groups.

In this sample, an XML file is within workspace. The content of the XML file used by the XML Authentication resource is shown below.

```
<?xml version="1.0" encoding="UTF-8"?>
<realm xmlns="http://xsd.tns.tibco.com/trinity/realm/2013"
hashAlgorithm="PBKDF2WithHmacSHA256" repetitionCount="10240"
xsi:schemaLocation="http://xsd.tns.tibco.com/trinity/realm/2013
Realm.xsd" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

  <users>

    <user>
      <name>john</name>
      <plaintext>password</plaintext>
      <!-- attributes are optional -->
      <attribute name="some-name" oid="some:thing">some-
value</attribute>
    </user>
    <user>
      <name>alex</name>
```

```
      <plaintext>password</plaintext>
       </user>

  </users>

  <group-mapping>
    <group-name>managers</group-name>
    <user-name>alex</user-name>
  </group-mapping>
  <group-mapping>

    <group-name>administrators</group-name>
    <user-name>alex</user-name>
    <user-name>john</user-name>
  </group-mapping>

</realm>
```

Enforcing Basic Credential Mapping on an Outbound Request

In this sample, basic credential mapping is enforced when InvokeRestApiProcess.bwp process makes a client call to the RestBindingProcess.bwp process.

Procedure

1. In the **samples** directory, select **policy > basiccredentialmapping > InvokeRestAPI** and double-click **tibco.bw.sample.policy.basiccredentialmapping.InvokeRestAPI.zip**. For more information, see Accessing Samples.
2. In **Project Explorer** expand the **tibco.bw.sample.policy.basiccredentialmapping.InvokeRestAPI** project.
3. Fully expand the **Processes** directory and double-click **InvokeRestApiProcess.bwp** and **RestBindingProcess.bwp**.
4. Set the default **ApplicationProfile** to match the OS you are running on. For more information, see Setting the Default Application Profile.
5. Click **Run > Debug Configurations**.

6. At the left hand tree of **Debug Configuration** wizard, expand **BusinessWorks Application** and select **BWApplication**.
7. Click the **Applications** tab and then click the **Deselect All** button if you have multiple applications. Select the checkbox next to **tibco.bw.sample.policy.basiccredentialmapping.InvokeRestAPI.application**.
8. Click **Debug**.

   This runs the sample in Debug mode.

Result

The client is successfully invoked and the file invokeRest.log is generated at C:\tmp\policy on a Windows system, or at /tmp/policy on a Unix system.

Understanding the Configuration

The following processes interact with each other in this project:

- **RestBindingProcess.bwp**: To ensure the message received from InvokeRestApiProcess is sent by an authenticated user, the service side of this process is associated with a Basic Authentication policy that is configured with XML file authentication. The XML file is included in the sample project and stored in your workspace.
- **InvokeRestApiProcess.bwp**: To ensure credentials are provided in the request message sent to RestBindingProcess, a Basic Credential Mapping policy configured for fixed credential mapping is associated with the **InvokeRESTAPI** activity in this process.

Enforcing SOAP Security to Enable Confidentiality and Integrity on Message Exchanges

This sample describes how WSS Provider and WSS Consumer policies can be enforced on SOAP/HTTP message exchanges to ensure confidentiality and integrity.

In this sample, the following processes communicate with each other:

- **ServiceProvider.bwp**: Provides a SOAP service.
- **ServiceConsumer.bwp**: Consumes the SOAP service provided by the ServiceProvider process.

Also service provider asks to decrypt message from consumer and verify signature of consumer.

Before you begin

Procedure

1. In the **samples** directory, select **policy > confidentialityintegrity > SoapHttp** and double-click **tibco.bw.sample.policy.confidentialityintegrity.SoapHttp.zip**. For more information, see Accessing Samples.
2. In **Project Explorer** expand the **tibco.bw.sample.policy.confidentialityintegrity.SoapHttp** project.
3. Set the default **ApplicationProfile** to match the OS you are running on. For more information, see Setting the Default Application Profile.
4. Click **Run > Debug Configurations**.
5. At the left hand tree of Debug Configuration wizard, expand **BusinessWorks Application** and select **BWApplication**.
6. Click the **Applications** tab and then click the **Deselect All** button if you have multiple applications. Select the checkbox next to **tibco.bw.sample.policy.confidentialityintegrity.SoapHttp**.
7. Click **Debug**.

   This runs the sample in Debug mode.

Result

The ServiceConsumer process successfully calls the ServiceProvider process.

The file ConfidentialityIntegrity.txt is generated in the C:\tmp\policy directory on Windows platform, or /tmp/policy directory on Unix platform.

Open the `ConfidentialityIntegrity.txt` file in a text editor. The default contents of the file are outlined below.

```
The Request From Service Consumer: Bob The Response from Service
Provider:
Welcome you Bob
```

Understanding the Configuration

The following processes interact with each other in this project:

- **ServiceProvider.bwp**: To ensure decryption and signature verification on the incoming request message, a WSS Provider policy configured for confidentiality and integrity is associated with the service side of this process.
- **ServiceConsumer.bwp**: To ensure the outbound request message to ServiceProvider is encrypted and signed, a WSS Consumer policy configured for confidentiality and integrity is associated with the reference side of this process.

Enforcing SOAP Security to Enable SAML Authentication and SAML Credential Mapping

In this sample, SOAP message exchanges are secured with SAML Credential Mapping, user name token-based authentication, and SAML authentication.
The following processes communicate with each other in this sample:

- **ClientProcess.bwp**: Simulates a client call to Process1 to request a response from Process1
- **Process1.bwp**: Calls Process2
- **Process2.bwp**: Returns a response to Process1

Procedure

1. In the **samples** directory, select **policy > samlcredentialmapping > SoapHttp** and double-click **tibco.bw.sample.policy.samlcredentialmapping.SoapHttpBinding.zip**. For more information, see Accessing Samples.

2. In **Project Explorer** expand the **tibco.bw.sample.policy.samlcredentialmapping.SoapHttpBinding** project.
3. Set the default **ApplicationProfile** to match the OS you are running on. For more information, see Setting the Default Application Profile.
4. Click **Run > Debug Configurations**.
5. At the left hand tree of Debug Configuration wizard, expand **BusinessWorks Application** and select **BWApplication**.
6. Click the **Applications** tab and then click the **Deselect All** button if you have multiple applications. Select the checkbox next to **ttibco.bw.sample.policy.samlcredentialmapping.SoapHttpBinding.application**.
7. Click **Debug**.

   This runs the sample in Debug mode.

Result

Process1 call Process2 successfully

The file SamlCredentialMapping.txt is generated in the C:\tmp\policy directory on Windows platform, or /tmp/policy directory on Unix platform.

Open the `SamlCredentialMapping.txt` file in a text editor. The default contents of the file are outlined below.

```
Congratulations! This is response from Process2. SAML request is
successful!
```

Understanding the Configuration

The following processes interact with each other in this project:

- **Process2.bwp**: The service side of Process2 is associated with a WSS provider policy configured for SAML authentication.
- **Process1.bwp**: To ensure the outbound request to Process2 contains the credentials authenticated in Process1, the following policies are enforced on Process1:
  - A WSS Consumer policy configured for SAML credential mapping is associated

on the reference side.

- A WSS Provider policy configured for user name token authentication is associated with the service side.

- **ClientProcess.bwp**: To simulate a client call to Process1, a WSS Consumer policy, configured for user name token credential mapping, is associated with the reference side of ClientProcess.

Exposing Security Context

This sample describes how to expose the security context on SOAP/HTTP. In this sample, the following processes communicate with each other:

- **Provider.bwp**: Provides a SOAP service.
- **Consumer.bwp**: Consumes the SOAP service provided by the Provider process.

The service provider verifies the signature on request.

Procedure

1. In the samples directory, select **policy > exposesecuritycontext > SoapHttp** and double-click **tibco.bw.sample.policy.soap.exposesecuritycontext.module.zip**. For more information, see Accessing Samples.
2. In the Project Explorer pane, expand the **tibco.bw.sample.policy.soap.exposesecuritycontext.module**.
3. Set the default application profile to match the OS you are using. For more information, see Setting the Default Application Profile.
4. Click **Run > Debug Configurations**.
5. At the left tree of Debug Configuration wizard, expand **BusinessWorks Application** and select **BWApplication**.
6. Click the **Applications** tab and click the **Deselect All** button if you have multiple applications.
7. Select the checkbox next to **tibco.bw.sample.policy.soap.exposesecuritycontext.module.application**.
8. Click **Debug**. This runs the sample in the Debug mode.

Result

The **ServiceConsumer** process successfully calls the **ServiceProvider** process.

The file `exposeSecurityContext.txt` is generated in the `c:\tmp\policy` directory on the Windows platform, or `/tmp/policy` directory on the Unix platform.

Open the `exposeSecurityContext.txt` file in a text editor. The default contents of the file are outlined below:

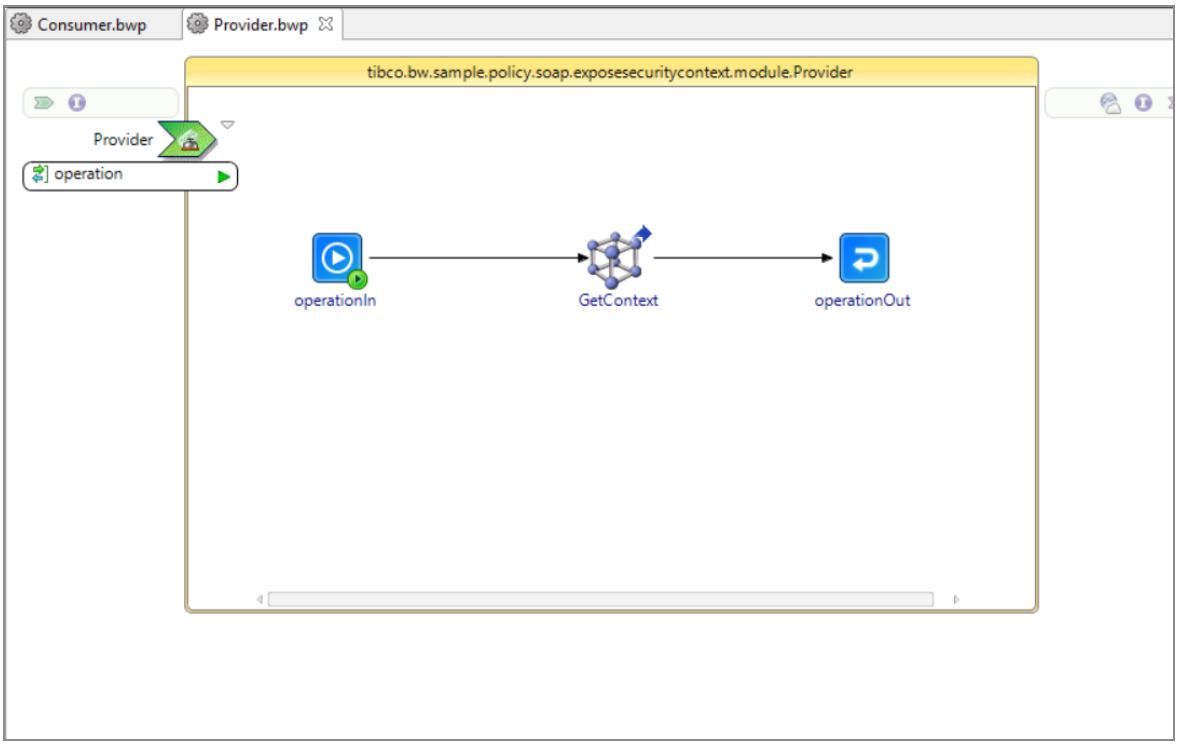
The Response from Service Provider:

```
Welcome you Bob
```

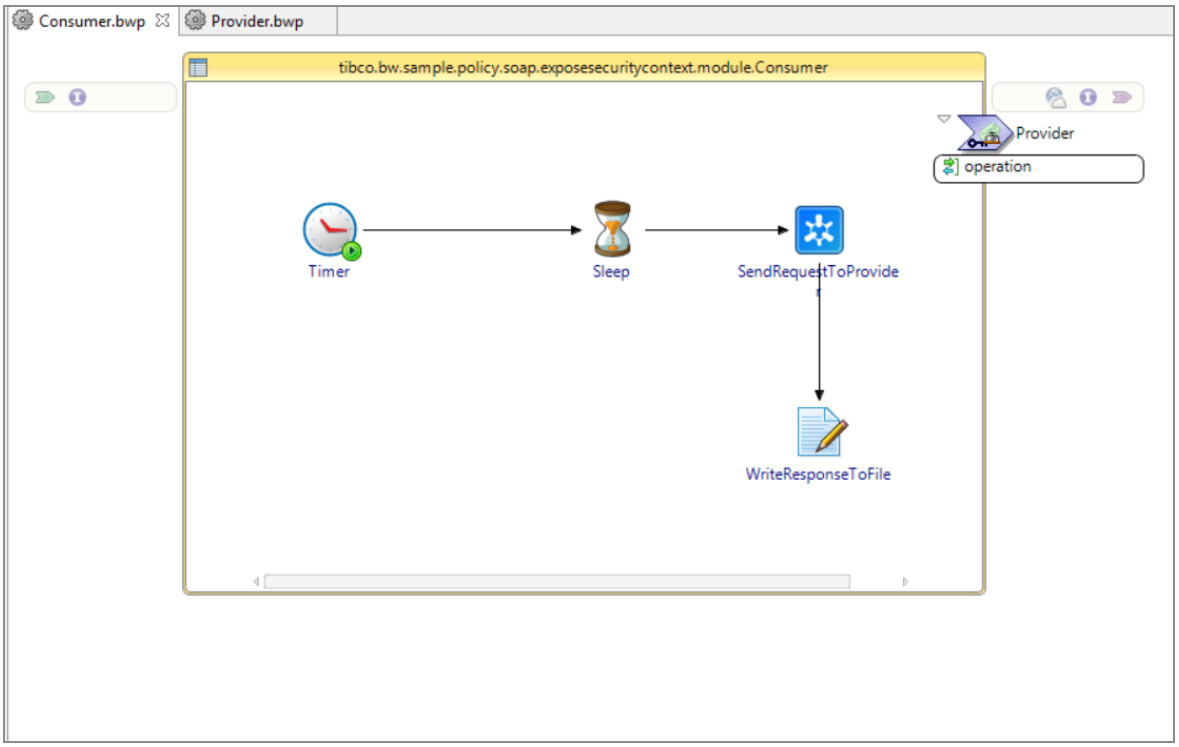
Understanding the Configuration

The following processes interact with each other in this project:

- **Provider.bwp**: To ensure the signature verification in the incoming request message, a WSS Provider policy configured for integrity is associated with the service side of this process. **Get Context** Activity is used to expose Security (WSSE) context parameters.



- **Consumer.bwp**: To ensure the outbound request message to Provider is signed, a WSS Consumer policy configured for integrity is associated with the reference side of this process.


Consumer.bwp
Provider.bwp
tibco.bw.sample.policy.soap.exposesecuritycontext.module.Consumer
Provider
operation
Timer
Sleep
SendRequestToProvide
r
WriteResponseToFile

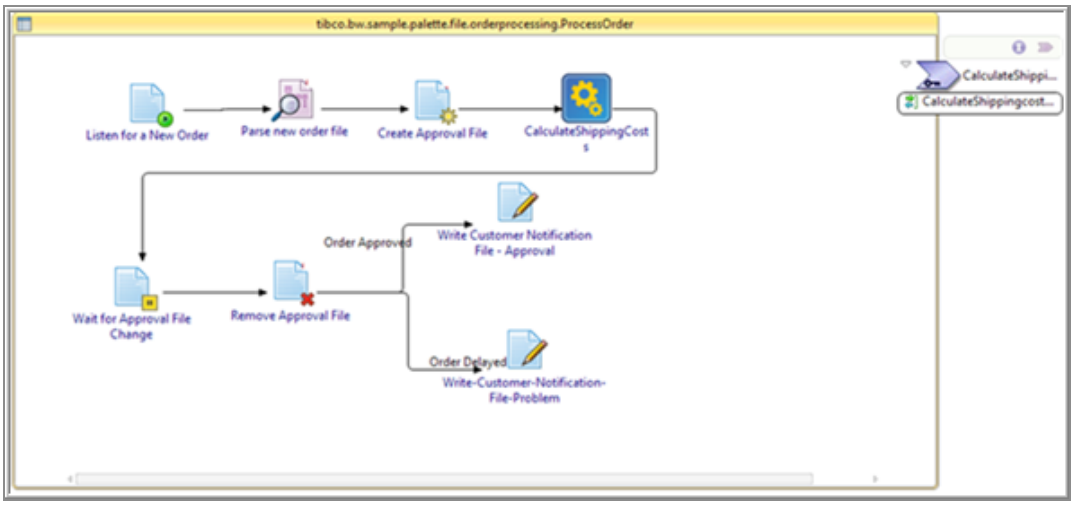
Palettes Features

This section shows the use of palette activities by providing samples for using JAVA HTTP.

Creating an Order Processing Service

This sample describes a simple order processing service where a File Poller activity waits for the `BW_new_order.xml` file to be updated. After an update is detected, the process creates an empty approval file and calls a sub process to calculate the shipping costs. The process then waits for the empty approval file to be updated. Finally, the `BW_customer_notification.txt` file with data in XML format is created in the `c:\tmp\OrderProcessing` folder of the default location, or the path you specified earlier that gives the customer the relevant shipping information.

In a real system, tasks such as database queries, email acknowledgment, and so on would be used to implement this scenario. However, we have chosen to use files for this service to highlight the use of the activities in the file palette.



Note: If you are accessing the sample from the **Welcome** page of TIBCO Business Studio™ for BusinessWorks™, go directly to **Step 3**.

Procedure

1. In the **samples** directory, select **palette > file > OrderProcessing** and double-click **tibco.bw.sample.palette.file.orderprocessing**. For more information, see Accessing Samples.
2. In the **Project Explorer** view, expand the **tibco.bw.sample.palette.file.orderprocessing** project.
3. Set the default **ApplicationProfile** to match the OS you are running on. For more information, see Setting the Default Application Profile.
4. Fully expand the **Processes** directory and double-click **ProcessOrder.bwp**.
5. Click **Run > Debug Configurations**.
6. At the left hand tree of Debug Configuration wizard, expand **BusinessWorks Application** and select **BWApplication**.
7. Click the **Applications** tab and then click the **Deselect All** button if you have multiple applications. Select the checkbox next to **tibco.bw.sample.palette.file.orderprocessing.application**.
8. Click **Debug**.

   This runs the sample in Debug mode.
9. Click the **Terminate** icon to stop the process.

Result

The `BW_customer_notification.txt` output file at `C:\tmp\OrderProcessing` should have the following content:

```
Hank Hill,
                Your order has been approved
                The cost is as follows,
                Purchase cost : $123
                Shipping costs : $40
                Total Cost : $163Hank Hill,
                Your order has been delayed due to problems
                We will contact you at your Arlen address with further
                details.
                Regards,
                TIBCO Software Inc.
```

The `BW_approve_order.txt` file is deleted.

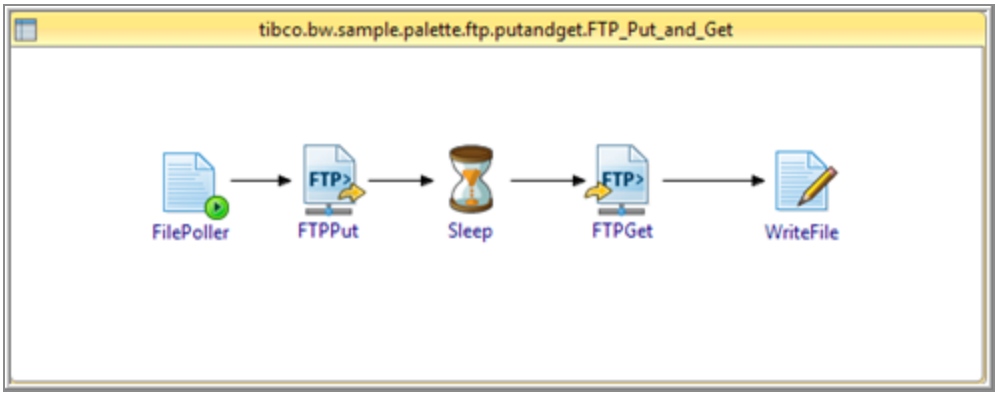
Understanding the Configuration

The following file names are defined as Module Properties and can be changed in the Module Properties dialog.

- `BW_new_order.xml` points to the XML file that contains the new order details and triggers the process initially as **Include Existing Files** checkbox option is selected. For consecutive triggers, the file must be updated (overwritten or modified) each time for the sample process to run or the process does not start.
- `BW_shipping_costs.xml` contains shipping cost information.
- `BW_customer_notification.txt` is created by the process to notify the customer of the complete shipping details.
- `BW_approve_order.txt` is created by the process. The file is deleted during process execution.

Using FTP to Put and Get Files

This sample service invokes the FTP put and get commands. The service reads a file and places it on the specified FTP server. The FTP get command is issued after a short pause and the file contents are written back to the local storage.



Before you begin

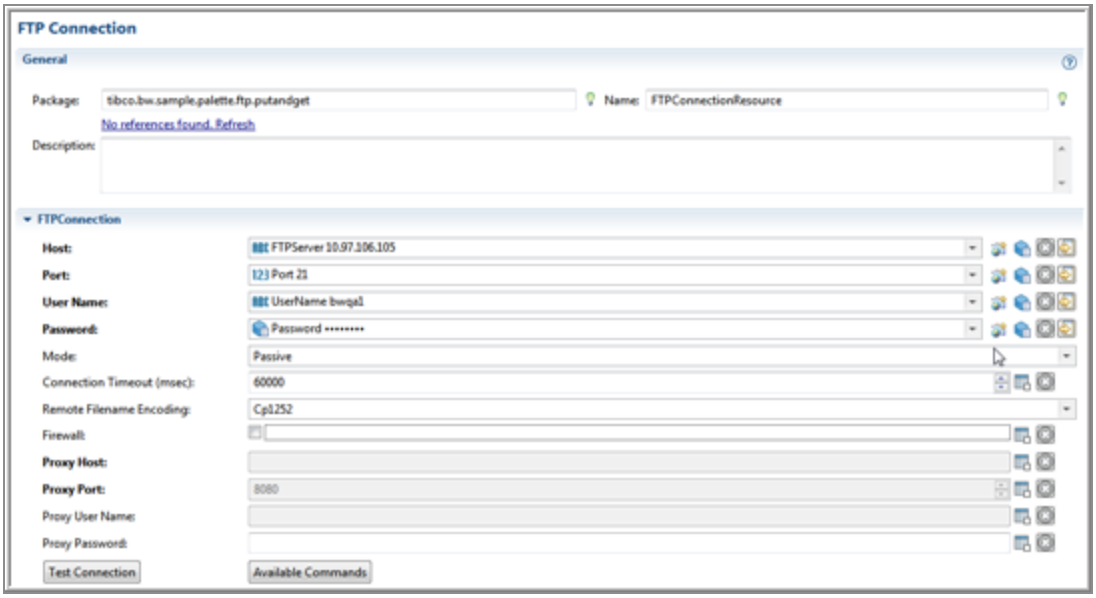
Copy the `ftp_put_file.log` file included in this sample to the `c:\tmp` directory. If the directory is not available, see the steps listed in the Understanding the Configuration section.

Ensure that the following FTP server information is available:

- FTP server name or IP address
- FTP server port (default is 21)
- FTP username
- FTP user password

Procedure

1. In the **samples** directory, select **palette > ftp > PutAndGet** and double-click **tibco.bw.sample.palette.ftp.PutAndGet**. For more information, see Accessing Samples.
2. In **Project Explorer** expand the **tibco.bw.sample.palette.ftp.PutAndGet** project.
3. Set the default **ApplicationProfile** to match the OS you are running on. For more information, see Setting the Default Application Profile.
4. Copy the `ftp_put_file.log` file to `c:\tmp`.
5. Fully expand the **Resources** directory and then double-click **FTPConnectionResource.ftpResource**. Specify valid values for your FTP connection in the **Host**, **Port**, **User Name**, and **Password** fields. Click **Test Connection** to verify your FTP connection.



6. Click **File > Save**.
7. Fully expand the **Processes** directory and double-click **FTP_Put_and_Get.bwp**.

8. Click **Run > Debug Configurations**.
9. At left hand tree of Debug Configuration wizard, expand **BusinessWorks Application** and select **BWApplication**.
10. Click the **Applications** tab and then click the **Deselect All** button if you have multiple applications. Select the checkbox next to **tibco.bw.sample.palette.ftp.PutAndGet.application**.
11. Click **Debug**.

    This runs the sample in Debug mode.
12. Monitor the log/trace and after a minute, verify whether the `PutAndGet.log` file has been created at `c:\tmp\PutAndGet`.
13. Click the **Terminate** icon to stop the process.

Result

- The file specified by the module property **BW_ftp_put_file** is copied to the FTP server.
- The file specified by the module property **BW_ftp_get_file** is written to `c:\tmp\PutAndGet` by the FTP server. The default name is `PutAndGet.log`.

Understanding the Configuration

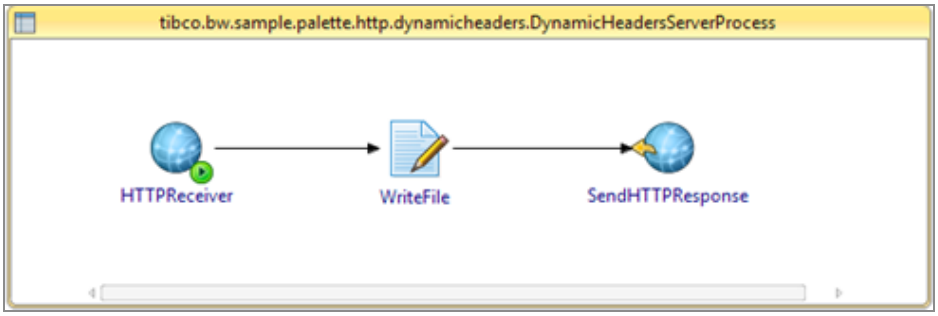
The sample project has default values for the file names. You can change the defaults, if needed for your environment.

1. Under **Module Descriptors**, click the configuration file.
2. Modify **BW_ftp_put_file** to point to the input file.
3. Modify **BW_ftp_get_file** to point to the output file that is written to on disk.

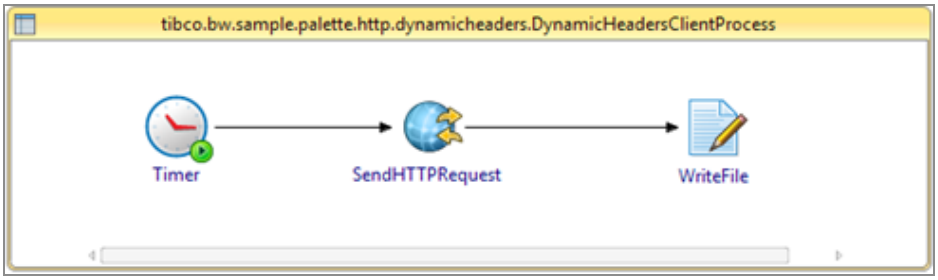
Using HTTP to Send and Receive Dynamic Headers

A request is sent to the server by the client. The server accepts the request and the HTTP connection is then closed by the response activity. Dynamic headers are added on both the client request and server's response.

The server process listens for requests and sends a response from the server.



The client process sends a request to the server.



Procedure

1. In the **samples** directory, select **palette > http > DynamicHeaders** and double-click **tibco.bw.sample.palette.http.DynamicHeaders**. For more information, see Accessing Samples.
2. In **Project Explorer** expand the **tibco.bw.sample.palette.http.DynamicHeaders** project.
3. Fully expand the **Processes** directory and double-click **DynamicHeadersClientProcess.bwp**.
4. Click **Run > Debug Configurations**.
5. At the left hand tree of Debug Configuration wizard, expand **BusinessWorks**

Application and select **BWApplication**.

6. Click the **Applications** tab and then click the **Deselect All** button if you have multiple applications. Select the checkbox next to **tibco.bw.sample.palette.http.DynamicHeaders.application**.
7. Click **Debug**.

   This runs the sample in Debug mode.
8. Click the **Terminate** icon to stop the process.

Result

the following output:

Headers Received at the Server Side are:

content-type:text/plain

content-type1found_ContentType1_firstTime

content-type1found_ContentType1_SecondTime

arraydata2

arraydata1

arraydata3

Headers Received at the Client Side are:

HTTP/1.1 200 OK

Content-Type: text/plain

Content-Length: 12

server: Jetty(8.1.16.v20140903)

testing-mydata: Done Setting data

testing-mydata: Done Setting Dynamic Value

company: TIBCO India Pvt Ltd

company: TIBCO Inc, USA

Understanding the Configuration

The **ListeningHTTPConnection** and **RequestingHTTPConnection** connections are defined to listen on port 13009. To change the port, specify a new Port value in the Module Properties dialog. You can also change the default location of the OUTPUT_FILE file, by specifying it in the **Module Properties** dialog.

The Incoming HTTP Request process starter listens on the connection specified by the **ListeningHTTPConnection** shared resource.

In the **DynamicHeadersClientProcess**, the ArrayData element in the **Input Editor** of the **Send HTTP Request** activity has **Cardinality** defined as **Repeating**. The dynamic header value for the same is provided from the **Input** tab under the **DynamicHeaders** element. The dynamic headers **Content-Type1** and **Content-Type** are defined under the same.

In **DynamicHeadersServerProcess**, the **Testing-MyData** element in the **Input Editor** of the **Send HTTP Response** activity has **Cardinality** defined as **Repeating**. The dynamic header value for the same is provided from the **Input** tab under the **DynamicHeaders** element. The dynamic headers StatusLine and Company are defined under the same.

The **Send HTTP Request** activity sends the request to the server. The **HTTP Receiver** activity accepts the request. The **Send HTTP Response** activity closes the HTTP connection that has been established by the Incoming HTTP Request process starter.

HTTP Sending and Receiving MIMEAttachments

This sample shows how the HTTP Receiver, HTTP Response, and HTTP Request activities can be used to send MIME attachments over HTTP.

The sample proceeds as follows:

- The Client process sends a HTTP Request with three attachments.
- The Server process responds with a HTTP Response with two attachments.

Before you begin

Copy the `TIBCO_HOME\bw\n.n\samples\palette\http\MIMEAttachment\attachment1.gif` file to the `c:\tmp` folder on Windows or the `/tmp` folder on UNIX.

Procedure

1. In the **samples** directory, select **palette > http > MIMEAttachment** and double-click **tibco.bw.sample.palette.http.MIMEAttachment**. For more information, see Accessing Samples.
2. In **Project Explorer** expand the **tibco.bw.sample.palette.http.MIMEAttachment** project.
3. Fully expand the **Processes** directory and double-click **Client.bwp**.
4. Click **Run > Debug Configurations**.
5. At the left hand tree of Debug Configuration wizard, expand **BusinessWorks Application** and select **BWApplication**.
6. Click the **Applications** tab and then click the **Deselect All** button if you have multiple applications. Select the checkbox next to **tibco.bw.sample.palette.http.MIMEAttachment.application**.
7. Click **Debug**.

   This runs the sample in the Debug mode.
8. Click the **Terminate** icon to stop the process.

Result

[Server] Received attachment of content-type: application/octet-stream

[Server] Received attachment of content-type: image/gif

[Server] Received attachment of content-type: text/plain

[Server] Received attachment of content-type: image/gif

Reply sent with attachment

[Client] Received attachment of content-type=: text/plain

[Client] Received attachment of content-type=: image/gif

Understanding the Configuration

The client process sends a HTTP Request with following three attachments:

- textContent
- binaryContent

- fileName

The server process responds with a HTTP Response with following two attachments:

- textContent
- binaryContent

Using HTTP or SOAP to Get Information from MySQL

Several processes are defined that show how to get information from MySQL and Sakila, which is a sample database (included with MySQL) that has default schemas and tables.

Before you begin

Ensure to install JDBC Drivers for Design time and Run time.

For more information about a JDBC Connection shared resource, see the "JDBC Connection" section in the *TIBCO ActiveMatrix BusinessWorks™ Bindings and Palettes Reference* guide.

Procedure

1. In the **samples** directory, select **palette > http > MultiEndpointApp** and double-click **tibco.bw.sample.palette.http.MultiEndpointApp**. For more information, see Accessing Samples.
2. In **Project Explorer** expand the **tibco.bw.sample.palette.http.MultiEndpointApp** project.
3. Fully expand the **Resources** directory and then double-click **HTTPConnectorResource.httpConnResource**. Verify that the **Host** and **Port** values are correct.
4. Double-click **JDBCConnectionResource.jdbcResource** and verify the database connection properties. Click the **Test Connection** button to verify your database configuration.
5. Fully expand the **Processes** directory and double-click **ActorService.bwp**, **FilmService.bwp**, or **SakilaService.bwp**.

6. Click **Run > Debug Configurations**.
7. At the left hand tree of Debug Configuration wizard, expand **BusinessWorks Application** and select **BWApplication**.
8. Click the **Applications** tab and then click the **Deselect All** button if you have multiple applications. Select the checkbox next to **tibco.bw.palette.http.MultiEndpointApp.application**.
9. Click **Debug**.

   This runs the sample in Debug mode.
10. Click the **Terminate** icon to stop the process.

Result

Actor Name: CUBA OLIVIER

Film Title: ALIEN CENTERDescription: A Brilliant Drama of a Cat And a Mad Scientist who must Battle a Feminist in A MySQL Convention CUBA OLIVIER

Understanding the Configuration

Three processes are defined:

- **ActorService Process**: contains the HTTP Receiver requests the input query to retrieve actor information using JDBC Query Activity. The response is provided back to HTTP Response activity.
- **FilmService Process**: this process has logic similar to the ActorServer process to fetch film information using the JDBC Query activity.
- **SakilaService Process**: returns the actor information using the SOAP service, which invokes a JDBC Query and returns the result.

Using the HTTP Persistent Connection Feature

The **PersistentConnectionClientProcess** sends five simultaneous HTTP requests. Each **SendHttpRequest** activity in the **PersistentConnectionClientProcess** is configured with a **ClientPCMConnection** shared resource. The **HttpReceiver** in **ServerProcess** listens on connection specified by **ServerConnection** shared resource.

Procedure

1. In the **samples** directory, select **palette > http > PersistentConnection** and double-click **tibco.bw.sample.palette.http.PersistentConnection**. For more information, see Accessing Samples .
2. In **Project Explorer** expand the **tibco.bw.sample.palette.http.PersistentConnection** project.
3. Fully expand the **Processes** directory and double-click **PersistentConnectionClientProcess.bwp**.
4. Click **Run > Debug Configurations**.
5. At the left hand tree of Debug Configuration wizard, expand **BusinessWorks Application** and select **BWApplication**.
6. Click the **Applications** tab and then click the **Deselect All** button if you have multiple applications. Select the checkbox next to **tibco.bw.sample.palette.http.PersistentConnection.application**.
7. Click **Debug**.

   This runs the sample in Debug mode.
8. Click the **Terminate** icon to stop the process.

Result

is similar to the following:

Response received for Request2 at Wed, 11 Jun 2014 20:56:06 GMT

Response received for Request1 at Wed, 11 Jun 2014 20:56:06 GMT

Response received for Request4 at Wed, 11 Jun 2014 20:56:16 GMT

Response received for Request3 at Wed, 11 Jun 2014 20:56:16 GMT

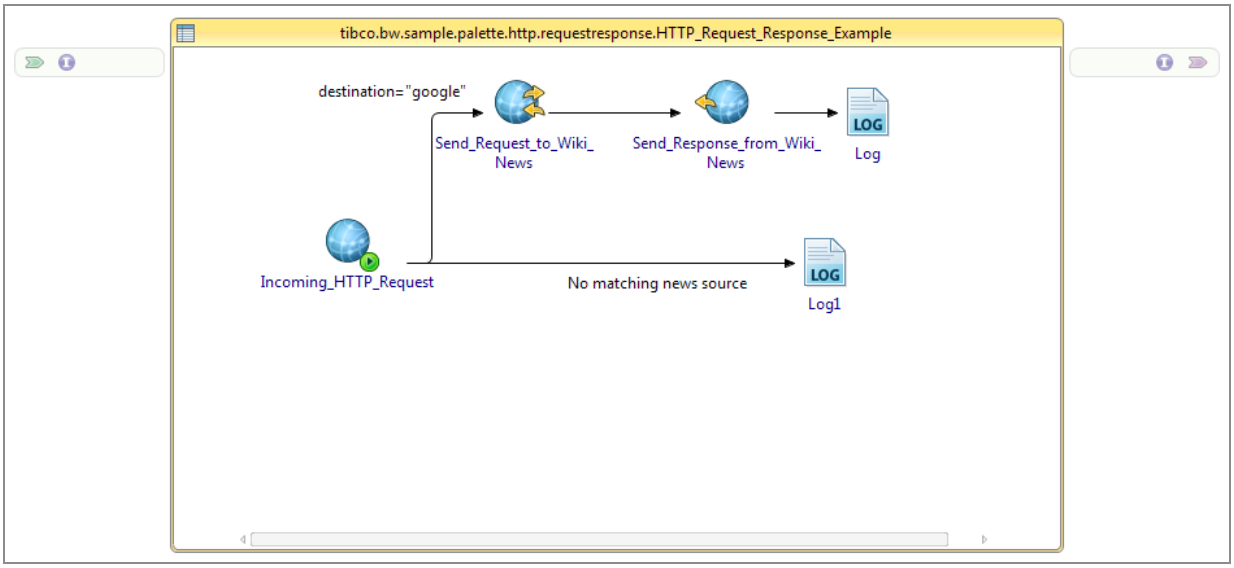
Response received for Request3 at Wed, 11 Jun 2014 20:56:26 GMT

Sending a Request and Getting a Response from a Website

Using the HTTP palette activities, you can configure requests to a web server and manage the response.

Before you begin

Your computer must be connected to the Internet.



Procedure

1. In the **samples** directory in the **File Explorer** view, expand **palette > http > RequestResponse** and double-click . For more information, see Accessing Samples.
2. In the **Project Explorer** view, expand the .
3. Expand the folder.
4. Fully expand the **Processes** directory and double-click to open it in the **Process Editor** pane.
5. Click **Run > Debug Configurations**.
6. At the left-hand tree of the Debug Configuration wizard, expand **BusinessWorks Application** and select **BWApplication**.

7. Click the **Applications** tab and then click the **Deselect All** button if you have multiple applications. Select the checkbox next to .
8. Click **Debug**.

   This runs the sample in Debug mode.
9. Run lendpoints at the prompt in the Console to obtain the endpoint for the application.
10. Copy the hostname from the endpoint.
11. In **File Explorer**, right-click the **RequestResponse** folder and select **Open Location** to access the RequestResponse folder in your file system.
12. Right-click `request_news.html` in your file system's folder and open it in a text editor.
13. Replace <*hostname*> with the actual hostname of your machine that you copied from the endpoint and save the file.
14. In **File Explorer**, under `samples\palette\http\RequestResponse`, double-click the `request_news.html` file to open it in TIBCO Business Studio™ for BusinessWorks™.
15. Click the **Get News from Google!** button to request headlines from the associated web page.
16. Click the **Terminate** icon to stop the process.

    The Google News web page displays in your default browser. The Response sent successfully! Message appears in the TIBCO Business Studio™ for BusinessWorks™ console.

Understanding the Configuration

The Incoming HTTP Request process starter listens on the connection specified in `ListeningHTTPConnection.httpConnResource`. The `request_news.html` file contains a form and clicking the **Get News from Google!** button sends the corresponding text string (Google) to the Incoming HTTP Request activity.

The conditional transition routes the request to the **Send HTTP Request** activity, which sends the request to the host using **ListeningHTTPConnection**. Finally, when the response comes from the remote site, the **Send HTTP Response** activity closes the HTTP connection established by the Incoming HTTP request process starter.

An internet connection is required for the sample to connect to Google News. The `ListeningHTTPConnection` listens on port 13008. Change the `LISTENING_PORT` value in the

Module Properties dialog. Also change the Port defined in the `request_news.html` file to the same value.

Sending a Secured Request and Getting a Secured Response from a Web Site

A secured HTTP request is sent to a web server. Mutual authentication takes place between the web server and client. After the authentication is successful, the web server sends valid response to the client completing the handshake.

Before you begin

Your computer must be connected to the Internet.

Copy the certificates to the `c:/tmp` folder on Windows and `/tmp` folder on Unix by unzipping the JKS folder packaged with the sample.

Procedure

1. In the **samples** directory, select **palette > http > SecuredRequestResponse** and double-click **tibco.bw.sample.palette.http.SecuredRequestResponse**. For more information, see Accessing Samples.
2. In the **Project Explorer** view, expand the **tibco.bw.sample.palette.http.SecuredRequestResponse** project.
3. Fully expand the **Processes** directory and double-click **ClientRequest.bwp**.
4. Click **Run > Debug Configurations**.
5. At the left hand tree of Debug Configuration wizard, expand **BusinessWorks Application** and select **BWApplication**.
6. Click the **Applications** tab and then click the **Deselect All** button if you have multiple applications. Select the checkbox next to **tibco.bw.sample.palette.http.SecuredRequestResponse.application**.
7. Click **Debug**.

   This runs the sample in Debug mode.
8. Click the **Terminate** icon to stop the process.

Result

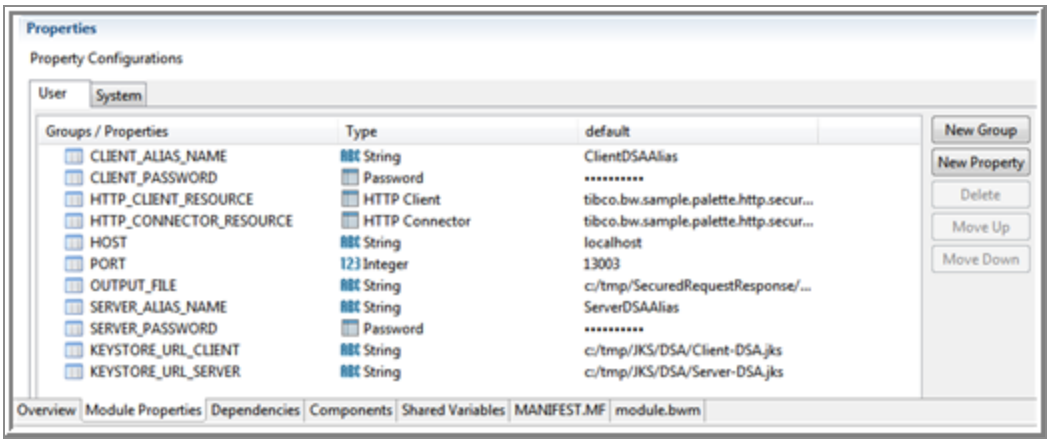
The following is the expected output:

Response received from BW HTTPS Server!!!

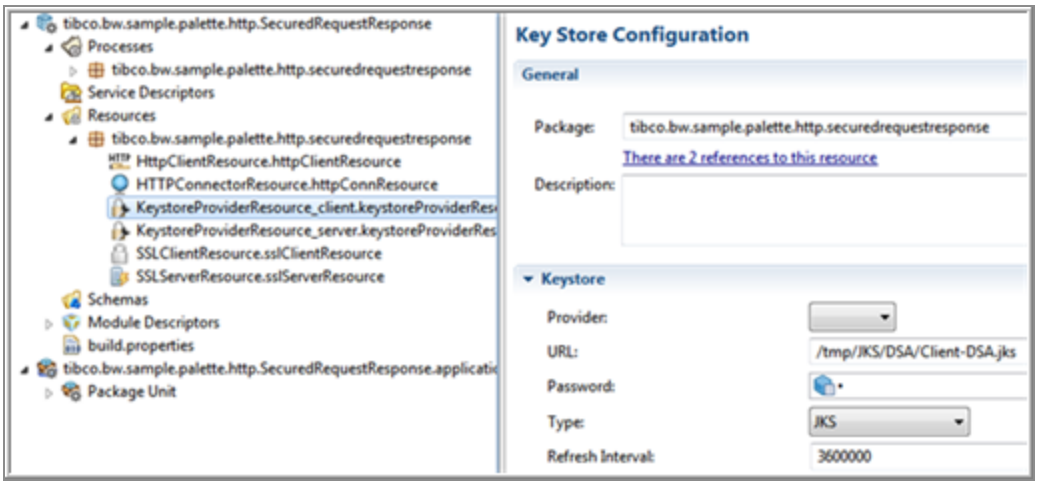
Understanding the Configuration

The **HttpClientResource** and the **HttpConnectorResource** is defined to listen on port 13003. To change the port, specify a new Port value in the **Module Property** dialog. If you want to change the default location of the OUTPUT_FILE, KEYSTORE_URL_CLIENT, KEYSTORE_URL_SERVER and HOST, specify it in the **Module Property** dialog.

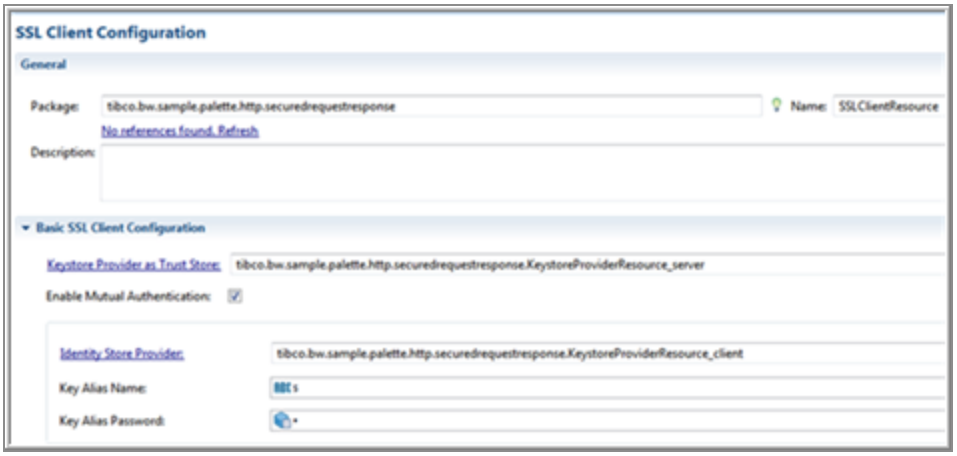
The **HTTP Receiver** process starter listens on the connection specified by the **HTTPConnectorResource** shared resource. Verify the path of the client and server certificates defined in the Module Properties dialog.



For the **SSLClientResource** and the **SSLServerResource** make sure you have provided correct entries for all the fields as shown in the following image.



The secured HTTP request is sent to the server. Mutual authentication happens between the Server and Client. After a successful authentication, the Server sends a valid response to Client, completing the handshake.

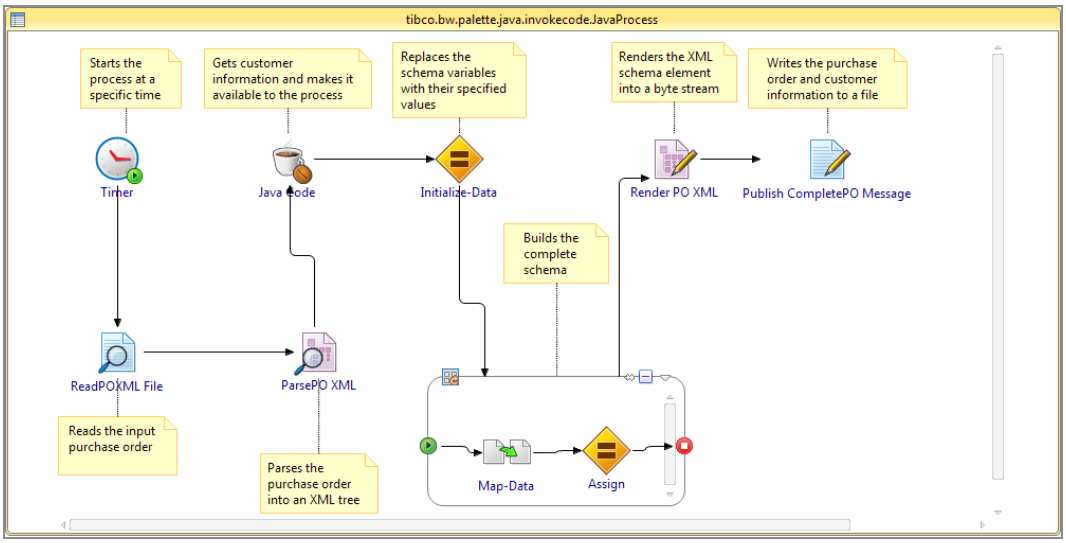


Invoking Custom Code to Complete a Purchase Order

The Java Invoke activity can be used to write custom code to get information from a legacy system, retrieve binary data, and so on.

Assume, for example, information from an ERP system is sent to a business partner using a B2B XML protocol such as CXML or RosettaNet. If some fields required by the protocol are not available from the ERP system, the missing information can be retrieved using the Java Invoke activity. The activity can access, for example, a CSV file. You can then combine the

information coming from the ERP system and the CSV file and send it to the business partner.



Note: If you are accessing the sample from the Welcome page of TIBCO Business Studio™ for BusinessWorks™, go directly to **Step 3**.

Procedure

1. In the **samples** directory, select **palette > java > InvokeCode** and double-click **tibco.bw.sample.palette.java.InvokeCode**. For more information, see Accessing Samples.
2. In the **Project Explorer** view, expand the **tibco.bw.sample.palette.java.InvokeCode** project.
3. Set the default **ApplicationProfile** to match the OS you are running on. For more information, see Setting the Default Application Profile.
4. Fully expand the **Processes** directory and double-click **JavaProcess.bwp**.
5. Click **Run > Debug Configurations**.
6. At the left hand tree of Debug Configuration wizard, expand **BusinessWorks Application** and select **BWApplication**.
7. Click the **Applications** tab and then click the **Deselect All** button. Select the checkbox next to **tibco.bw.sample.palette.java.InvokeCode.application**.
8. Click **Debug**.

This runs the sample in Debug mode.

9. Click the **Terminate** icon to stop the process.

Result

The process combines the purchase order information in the `PO.xml` file and customer information in `csvFile.log` file present in `BW_HOME\samples\palette\java\InvokeCode` folder and writes it to the `C:\tmp\InvokeCode\CompletePo.xml` file.

Understanding the Configuration

The sample uses the **Read File** activity to read and parse an XML file into an XML tree. The **Java Invoke** activity gets customer information from the `csvFile.log` file and makes it available to the process. A Map activity combines the **Read File** and **Java Invoke** output to build a complete schema, which contains the purchase order and customer information. The combined information is passed to the **RenderPOXML** activity that converts the information to XML string, using the specified schema. The process then publishes the XML string by writing it to the `CompletePo.xml` file in `C:\tmp\InvokeCode`.

The example requires the following schema files:

- **PurchaseOrderSchema** used by ParsePO XML, which parses the PO.
- **CustomerSchema** referenced by CompletePO, which contains customer information.
- **CompletePO** used by Render PO XML and has information on both the purchase order and the customer. CompletePO is a combination of **PurchaseOrderSchema** and **CustomerSchema**.

To view the Java source, fully expand the **src** directory and double-click **JavaProcessJavaCode.java**.

Troubleshooting

If any problem markers are visible in the project, close and re-open the project or import the project in a clean new workspace.

Using Java Activities to Publish Information

Using several Java activities, you can invoke a method on a Java object and translate the resulting data into an XML document. The data can then be made available to other activities.

Procedure

1. In the **samples** directory of the **File Explorer** view, expand **palette > java > InvokeMethod** and double-click to expand it in the Project Explorer. For more information, see Accessing Samples.
2. In **Project Explorer** expand the project.
3. Fully expand the **Processes** directory and double-click **PublishAllBalances.bwp** to open it in the Process Editor pane.
4. Click **Run > Debug Configurations**.
5. At the left-hand tree of the Debug Configuration wizard, expand **BusinessWorks Application** and select **BWApplication**.
6. Click the **Applications** tab and then click the **Deselect All** button if you have multiple applications. Select the checkbox next to .
7. Click **Debug**.

   This runs the sample in Debug mode.
8. Click the **Terminate** icon to stop the process.

   A log is displayed in the **Console** view.

Understanding the Configuration

The following three Java activities are used in this sample.

- The **Java Invoke** activity invokes a method on a Java object.
- The **Java To XML** activity translates data from a Java object into an XML document that can be later used in other activities.
- The **XML To Java** activity supplies data to an XML schema to construct an instance of the desired Java object.

The source code for the Java objects, LocalBank, Account, Balance, and SystemProperties are in the `src` directory in the project.

This project has the following three processes:

- PublishBalance
- PublishAllBalances
- PublishJavaInfo

The **PublishBalance** process runs as follows:

1. The **CreateLocalBank** activity is a **Java Invoke** activity that constructs an instance of the LocalBank object. This object contains the `getBalance(...)` method that the process later invokes to retrieve the balance.
2. The **CreateAccount** activity is an **XML To Java** activity that creates an instance of the Account object and sets the object's data using an XML schema.
3. The **getBalance** activity is a **Java Invoke** activity that invokes the `getBalance(...)` method on the LocalBank object created by the **CreateLocalBank** activity. The Account object created by the **CreateAccount** activity is also passed as a parameter to the `getBalance(...)` method. The method returns an instance of the Balance object that contains the balances for the specified account.
4. If the **getBalance** activity returns an exception, the error is published by the **PublishException** (if the exception is InvalidAccount) or **PublishError** activity (in the event of any other exceptions).
5. If the **getBalance** activity runs without any exception, the **Balance** activity is a **Java To XML** activity that takes the Balance object produced by the **getBalance** activity and converts the object's data to an XML document.
6. The **PublishBalance** activity takes the data produced by the Balance activity and publishes the account and balance information by writing it to

The **PublishAllBalances** process runs as follows:

1. The **getAllBalance** activity is a **Java Invoke** activity that constructs an instance of the LocalBank object and invokes the `getAllBalance()` method on that object. The output of the method is an instance of the `java.util.ArrayList` class that contains a collection of all balances.
2. The **CollectionToArray** activity invokes the toArray method of `java.util.ArrayList` to convert the collection to an array of objects.
3. The group then iterates over each object in the array. The **Balance** activity takes the current object in the array and converts its data to an XML document. The

PublishBalance activity publishes the data from the output of the **Balance** activity.

The **PublishJavaInfo** process allows a **Java To XML** activity to make the well-known Java system properties available as well as the host name and address. This process runs as follows:

1. The **SystemProperties** activity is a **Java To XML** activity that constructs an instance of the SystemProperties object and provides the data in the class as an XML schema.
2. The **PublishJavaInfo** activity uses the data produced by the SystemProperties activity and publishes the Java information by writing it to

Troubleshooting

- If the **Output** tab of **Java To XML** activity is not populated, then press the **Reload** button next to the **Class Name** in **Java To XML** activity.
- If any problem markers are visible in the project, close and reopen the project or import the project in a clean new workspace.

Using Java Invoke that Makes a REST call to Search Twitter

In this sample, Java Invoke makes a REST call to Twitter to search for a specified term. The application module contains the user-written code, user jars, and processes. There is no dependency on a Shared Module.

Before you begin

1. Create new app on - https://apps.twitter.com/ and generate Access Token, Access Token Secret, Consumer Key, Consumer Secret for the app created and use it in the parameters for SearchTwitter Activity.
2. To create a token:
    a. Login to https://apps.twitter.com/ with your Twitter account.
    b. Click **Create New App**.
    c. Provide **Name** and **Description** and create your Twitter application. Your application is successfully created.

d. Go to **API Keys** Tab and click **Create my access token**. Your application access token is successfully generated.

e. Click **Test OAuth**. You should see the OAuth Settings page which has the following tokens generated - Consumer key, Consumer secret, Access token, Access token secret.

Procedure

1. Browse the **samples** directory in File Explorer and select **palette > java > RESTInvokeToTwitter**. Double-click to import the sample to your workspace. For more information, see Accessing Samples.
2. In the **Project Explorer** view, expand the project.
3. Fully expand the **Processes** directory and double-click **RESTInvokeToTwitter.bwp**.
4. Click **Run > Debug Configurations**.
5. At the left hand tree of Debug Configuration wizard, expand **BusinessWorks Application** and select **BWApplication**.
6. Click the **Applications** tab and then click the **Deselect All** button if you have multiple applications. Select the checkbox next to .
7. Click **Debug**.

   This runs the sample in Debug mode.
8. Click the **Terminate** icon to stop the process.

Understanding the Configuration

The project contains the following process:

RESTInvokeToTwitter Process: This process has two Java Invoke activities: **OpenTwitterConnection** and **SearchTwitter**. The source code for these activities is located under `src` directory under `com.example.TwitterConnection` package in the same project. **OpenTwitterConnection** activity creates a TwitterConnection object which is then passed to **SearchTwitter** activity.

SearchTwitter activity requires the following 5 input parameters:

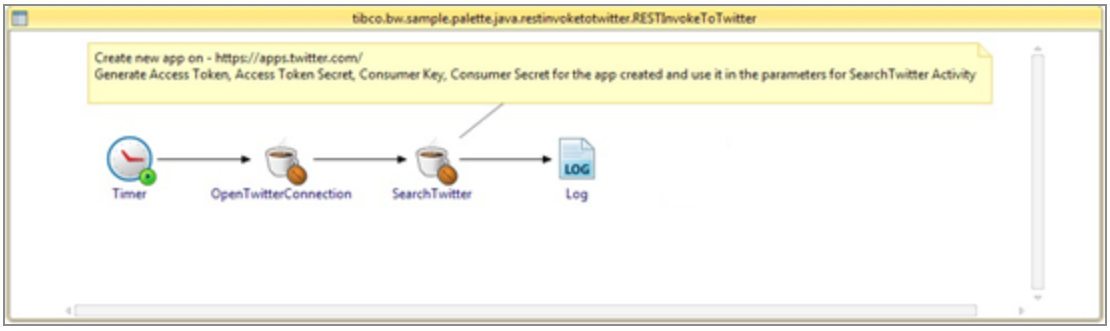
- `SearchString`

- `AccessToken`
- `AccessTokenSecret`
- `ConsumerKey`
- `ConsumerSecret`

`SearchString` is the string for which user wants to search tweets. In this project the SearchString is "Lady Gaga".

`AccessToken`, `AccessTokenSecret`, `ConsumerKey`, and `ConsumerSecret` are required for authentication to the Twitter Account. The steps to generate these are mentioned in the **Prerequisite** section.

The Java invoke uses Twitter API v1.1 which requires this request to be authenticated using **OAuth**.



Troubleshooting

If you receive an error message such as:

401:Authentication credentials (https://dev.twitter.com/pages/auth) were missing or incorrect. Ensure that you have set valid consumer key/secret, access token/secret, and the system clock is in sync.

Then, ensure that you have generated Access Token, Access Token Secret, Consumer Key, and Consumer Secret as per the steps specified in **Prerequisites** section.

Using a Custom Process Starter in a Project That Gets Information about a Checking Account

Java Event Source allows you to create a custom process starter written in Java.

Before you begin

Copy the `AccountInformation.txt` file from `TIBCO_HOME\bw\n.n\samples\palette\java\JavaEventSource` to `c:\tmp` directory on Windows and `/tmp` on UNIX.

Procedure

1. In the **samples** directory, select **palette > java > JavaEventSource** and double-click **tibco.bw.sample.palette.java.JavaEventSource**. For more information, see Accessing Samples.
2. In **Project Explorer** expand the **tibco.bw.sample.palette.java.JavaEventSource** project.
3. Set the default **ApplicationProfile** to match the OS you are running on. For more information, see Setting the Default Application Profile.
4. Fully expand the **Processes** directory and double-click **PublishAllBalances.bwp**.
5. Click **Run > Debug Configurations**.
6. At the left hand tree of Debug Configuration wizard, expand **BusinessWorks Application** and select **BWApplication**.
7. Click the **Applications** tab and then click the **Deselect All** button if you have multiple applications. Select the checkbox next to **tibco.bw.sample.palette.java.JavaEventSource.application**.
8. Click **Debug**.

   This runs the sample in Debug mode.
9. Click the **Terminate** icon to stop the process.

Result

The process writes the customer last name, account ID, and checking balance to the `PublishEvnSourceBalance.log` file at `c:\tmp\JavaEventSource`.

Understanding the Configuration

This sample uses the Java objects such as, LocalBank, Account, Balance, and SystemProperties. The source for these objects is in the `src` directory.

The process runs as follows:

1. The **Java Event Source** activity reads balances from the `AccountInformation.txt` and outputs an instance of the `java.util.LinkedList` class that contains a collection of all balances.
2. The **CollectionToArray** activity invokes the `java.util.LinkedListtoArray` method to convert the collection to an array of objects.
3. The group then iterates over each object in the array.
4. The **Balance** activity takes the current object in the array and converts its data to an XML document.
5. The **PublishBalance** activity publishes the data from the output of the **Balance** activity.

Troubleshooting

- If the **Output** tab of **Java To XML** activity is not populated, then press the **Reload** button next to the **Class Name** in **Java To XML** activity.
- If any problem markers are visible in the project, close and re-open the project or import the project in a clean new workspace.

Using JDBC Query - Process in Subsets and JDBC Update - Batch Updates

This sample shows how to use the **Advanced** tab options of **JDBC Query** and **JDBC Update** activities.

Before you begin

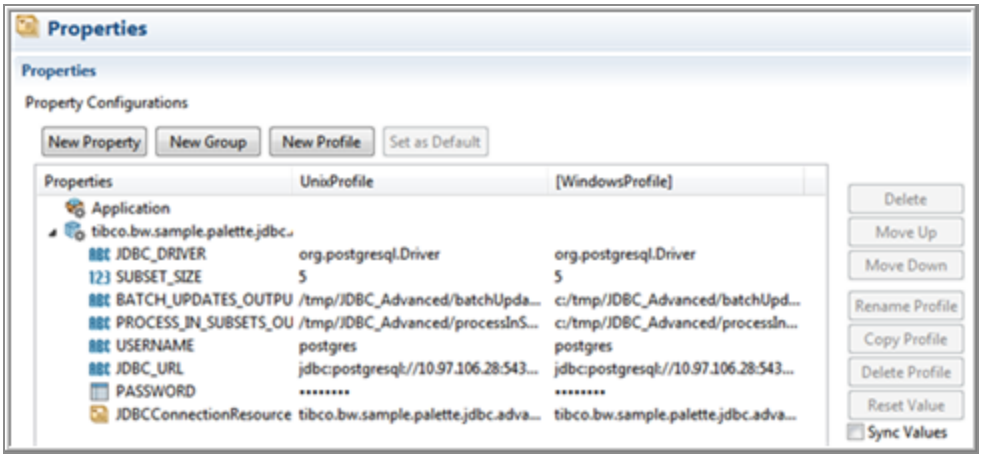
Install the PostgreSQL Database:

1.

Run `createTestTables.sql` on the PostgreSQL database to create test tables. The script is located in the samples directory under the path `TIBCO_HOME\bw\n.n\sample\palette\jdbc\Advanced`

Procedure

1. In the **samples** directory, select **palette > jdbc > Advanced** and double-click **tibco.bw.sample.palette.jdbc.Advanced**. For more information, see Accessing Samples.
2. In the **Project Explorer** view, expand the **tibco.bw.sample.palette.jdbc.Advanced** project.
3. Fully expand **Module Descriptors** under **tibco.bw.sample.palette.jdbc.Advanced** and double-click **Module Properties**. The JDBC properties defined for the application are defined in the dialog. Provide a valid username, password, and database URL to connect to your PostgreSQL database.



4. Update the values under Module properties and save your project.

   The values in `Advanded.application` properties are updated and the connection to the database is successful.

5. Verify your JDBC connection.

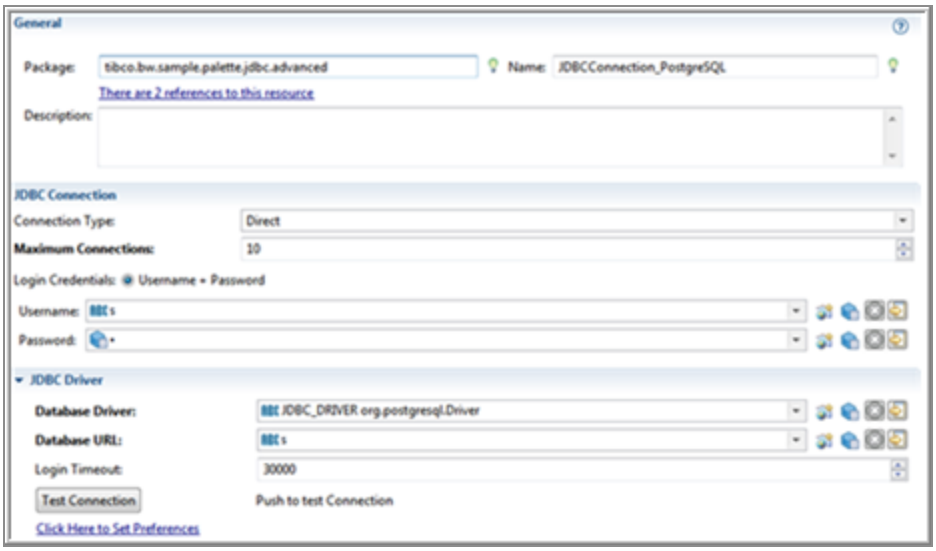
   a. Fully expand the **Resources** directory.

   b. Double-click **JDBCConnection_PostgreSQL.jdbcResource**.

   c. In **JDBC Driver**, click **Click here to set preferences**.

   d. Set the JDBC driver folder directory preference and click **Apply**. Click **OK**.

   e. Click the **Test Connection** button to verify the connection.



6. Click **File > Save** to save the project.

7. Fully expand the **Processes** directory and double-click **ProcessInSubsets.bwp** and **BatchUpdates.bwp**.

8. Click **Run > Debug Configurations**.

9. At the left hand tree of Debug Configuration wizard, expand **BusinessWorks Application** and select **BWApplication**.

10. Click the **Applications** tab and then click the **Deselect All** button if you have multiple applications. Select the checkbox next to **tibco.bw.sample.palette.jdbc.Advanced.application**.

11. Click **Debug**.

    The sample runs in Debug mode.

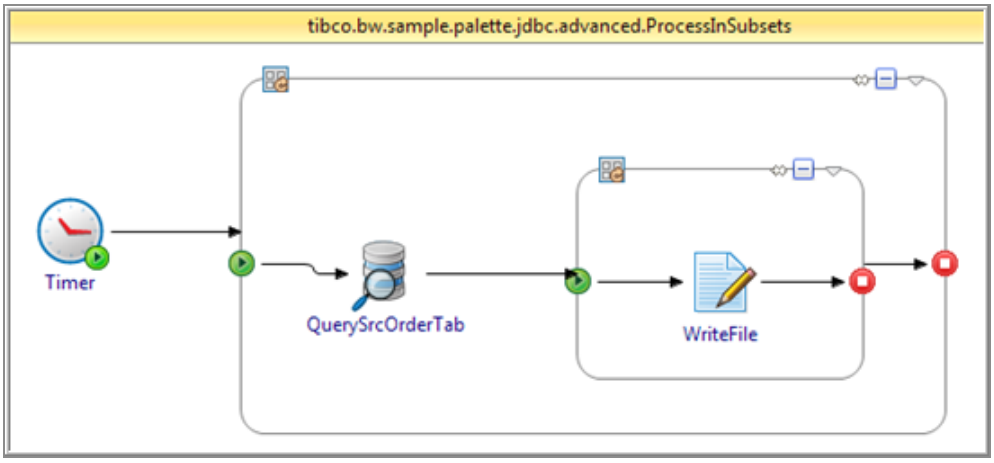
12. Click the **Terminate** icon to stop the process.

Understanding the Configuration

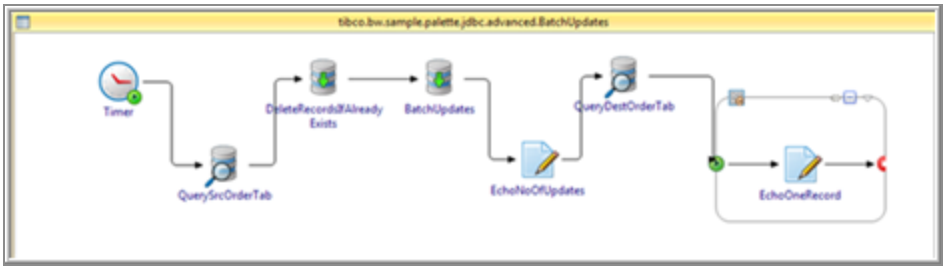
This sample shows:

- **JDBC Query** - Process in Subsets: This activity allows users to process the result set in smaller subsets, rather than processing the entire result set at once. It uses the ProcessInSubsets process.
- **JDBC Update** - Batch Updates: This activity is configured to perform multiple statements in one batch. It uses the BatchUpdates process.

The **ProcessInSubsets** process is configured to process query in subsets.



The **BatchUpdates** process is configured to perform multiple insert statements in one batch.



ProcessInSubsets runs as follows:

- QuerySrcOrderTab is a **JDBC Query** activity configured to retrieve subsets of the result set of SRC_ORDER_TABLE table
- Check the **Process In Subsets** field on the **Advanced** tab.
- Set the subsetSize input item to the number of records you want to process for each execution. In this sample, it is set as a module property named SUBSET_SIZE.

To retrieve subsets of the result set, you must use a **Repeat** group to iterate until the entire result set is processed. For this, both **QuerySrcOrderTab** and **ProcessSubsets** are grouped into **processInSubsets** Repeat group with the exit condition: `$QuerySrcOrderTab/lastSubset = true()`.

BatchUpdates runs as follows:

- **QuerySrcOrderTab** is a **JDBC Query** activity to retrieve the entire result set of SRC_ORDER_TABLE table in one batch.
- The **DeleteRecordsIfAlreadyExists** activity deletes the records that are already in the database.
- **BatchUpdates** is a **JDBC Update** activity configured to perform multiple insertions in one batch. It inserts all the records of the SRC_ORDER_TABLE table to the DEST_ORDER_TABLE table.
    - Check the **Batch Update** field on the **Advanced** tab.
    - Map the `$QuerySrcOrderTab/Record` to the Record input repeating element.
- **QueryDestOrderTab** retrieves the entire result set of DEST_ORDER_TABLE table at once.

Setting Up a JDBC Connection to Query and Update Tables

After setting up a JDBC connection, a process runs the SELECT and UPDATE queries.

Before you begin

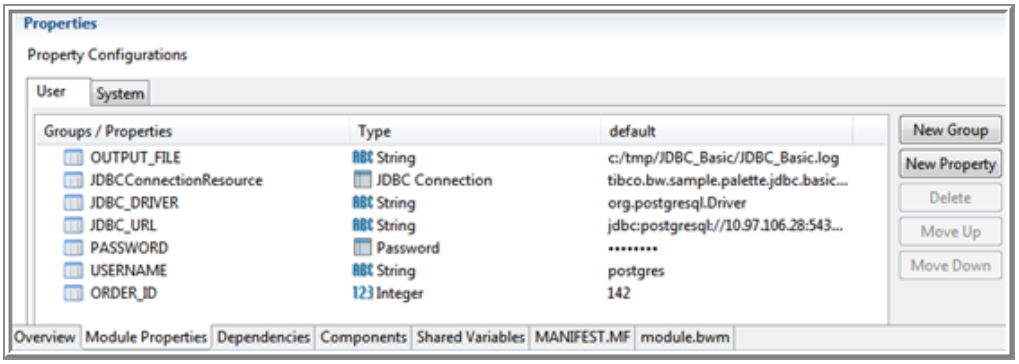
Install the PostgreSQL database:

1.

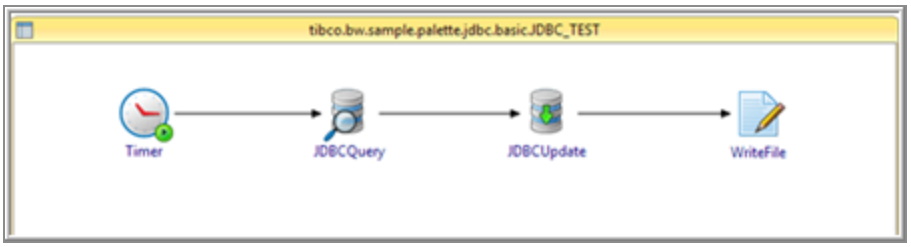
Run `JDBC.sql` on the PostgreSQL database to create test tables. The script is located in the `samples` directory under the path `TIBCO_HOME\bw\n.n\samples\palette\jdbc\Basic`.

Procedure

1. In the **samples** directory, select **palette > jdbc > Basic** and double-click **tibco.bw.sample.palette.jdbc.Basic**. For more information, see Accessing Samples.
2. In the **Project Explorer** view, expand the **tibco.bw.sample.palette.jdbc.Basic** project.
3. Fully expand **Module Descriptors** under **tibco.bw.sample.palette.jdbc.Basic** and double-click **Module Properties**. The JDBC properties defined for the application are defined in the dialog. Provide a valid username, password, and database URL to connect to your PostgreSQL database.

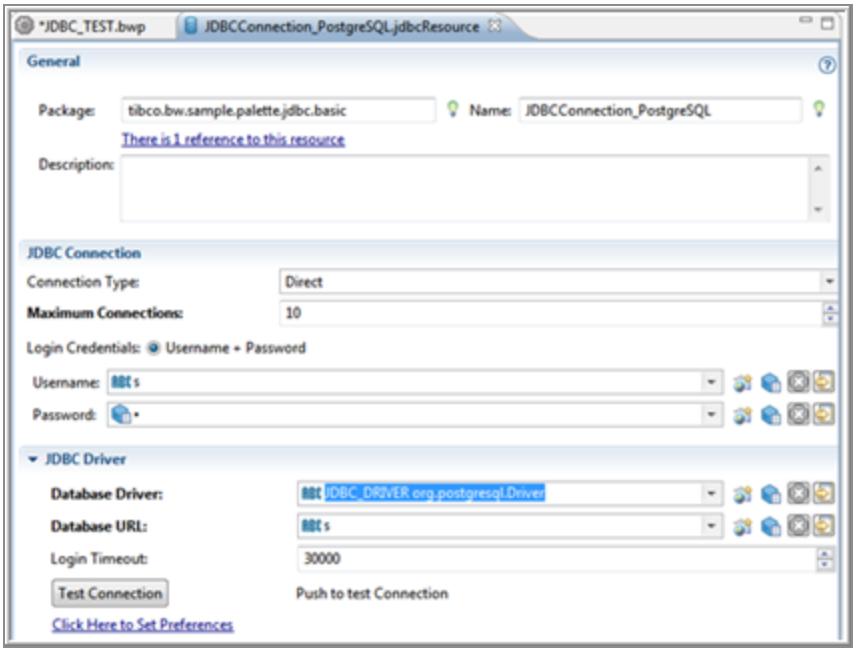


4. Fully expand the **Processes** directory and double-click **JDBC_TEST.bwp**.



5. Verify your JDBC connection.
   a. Fully expand the **Resources** directory.
   b. Double-click **JDBCConnection_PostgreSQL.jdbcResource**.
   c. In **JDBC Driver**, click **Click Here to Set Preferences**.
   d. Set the JDBC driver folder directory preference and click **Apply**. Click **OK**.

e. Click the **Test Connection** button to verify the connection.



6. Click **File > Save** to save the project.
7. Click **Run > Debug Configurations**.
8. At the left hand tree of Debug Configuration wizard, expand **BusinessWorks Application** and select **BWApplication**.
9. Click the **Applications** tab and then click the **Deselect All** button if you have multiple applications. Select the checkbox next to **tibco.bw.sample.palette.jdbc.Basic.application**.
10. Click **Debug**.

    This runs the sample in Debug mode.
11. Click the **Terminate** icon to stop the process.

Result

- The order_description record in the sub_order table is updated to TESTING JDBC CONNECTION.

Understanding the Configuration

The **JDBC_TEST.bwp** process uses the JDBC Connection in the **Resources** folder to establish a connection to the database and run SELECT and UPDATE queries. First it queries the table for a specific record using the **JDBC_Query** activity and then it updates another table using the **JDBC_Update** activity. Finally it updates a file indicating the number of records written.

Both activities use prepared parameters as input. This shows the ability to run the same statement with multiple values by caching the statement at runtime.

Using Oracle Objects and Collections in JDBC Call Procedure and JDBC Query Activities

The JDBC Call Procedure activity calls a database procedure. The JDBC Query activity performs the specified SQL SELECT statement.

This sample has the following processes:

- The **CreateCustomer** and **SelectCustomer** processes show how to use Oracle Objects.
- The **CreateCustomerMultiAddress** and **SelectCustomerMultiAddress** processes show how to use Oracle Collections.

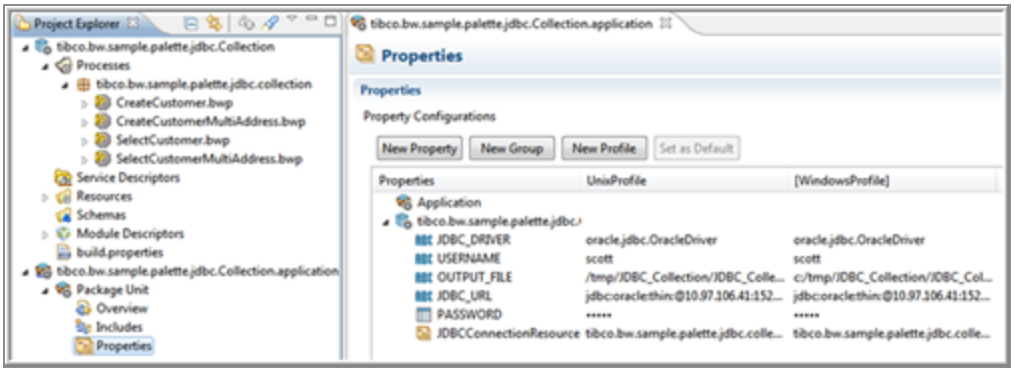
Before you begin

- You must have credentials that allow you to run scripts against your Oracle database.
- Run the `jdbc_object_collection.sql` file against your Oracle database to create the test objects. The file is located in the samples directory under the path `TIBCO_HOME\bw\n.n\samples\palette\jdbc\Collection`.
- Ensure to install Drivers for Design time and Run time.

> **Note:** For more information about installing Drivers, see "JDBC Connection" section in the *TIBCO ActiveMatrix BusinessWorks™ Bindings and Palettes Reference* guide.

Procedure

1. In the **samples** directory, select **palette > jdbc > Collection** and double-click **tibco.bw.sample.palette.jdbc.Collection**. For more information, see Accessing Samples.
2. In the **Project Explorer** view, expand the **tibco.bw.sample.palette.jdbc.Collection** project.
3. Fully expand **tibco.bw.sample.palette.jdbc.Collection.application** and double-click **Properties**. The JDBC properties defined for the application are shown next. Provide a valid username, password, and database URL to connect to the Oracle database in your environment.



4. Verify your JDBC connection:
   a. Fully expand the **Resources** directory.
   b. Double-click **OracleConnection-OracleThinDriver.jdbcResource**.
   c. In JDBC Driver, click **Click Here to Set Preferences**.
   d. Set the JDBC driver folder directory preference and click **Apply**. Click **OK**.
   e. Click the **Test Connection** button to verify the connection.
5. Click **File > Save** to save the project.
6. Click **Run > Debug Configurations**.
7. At the left hand tree of Debug Configuration wizard, expand **BusinessWorks Application** and select **BWApplication**.
8. Click the **Applications** tab and then click the **Deselect All** button if you have multiple applications. Select the checkbox next to **tibco.bw.sample.palette.jdbc.Collection.application**.

9. Click **Debug**.

   This runs the sample in Debug mode.

10. Click the **Terminate** icon to stop the process.

Result

- On successful completion, the **CreateCustomer** and **CreateCustomerMultiAddress** processes insert a row each in the database.
- On successful completion, the **SelectCustomer** and **SelectCustomerMultiAddress** processes display the data that was inserted by each process in the **Output** tab.

Understanding the Configuration

The sample consists of the following processes:

- **CreateCustomer**: In this process, the **JDBCCallProcedure** activity ADD_SINGLE_ADDRESS_CUSTOMER inserts a record containing Oracle Objects into the database.
- **CreateCustomerMultiAddress**: In this process, the **JDBCCallProcedure** activity ADD_MULTI_ADDRESS_CUSTOMER inserts a record containing Oracle Collections into the database.
- **SelectCustomer**: In this process, the **JDBCQuery** activity queries the database for the record added using the CreateCustomer process and displays the record in the **Output** tab.
- **SelectCustomerMultiAddress**: In this process, the **JDBCQuery** activity queries the database for the record added using the **CreateCustomerMultiAddress** process and displays the record in the **Output** tab.

Setting Up and Executing JDBC Activities with Eclipse Using DTP

Using JDBC palette activities, you can setup JDBC connections and run JDBC activities to run queries and display the results. This sample shows how to run SQL calls and edit data with Eclipse DTP at Design time.

Before you begin

Install the Sakila Sample Database.

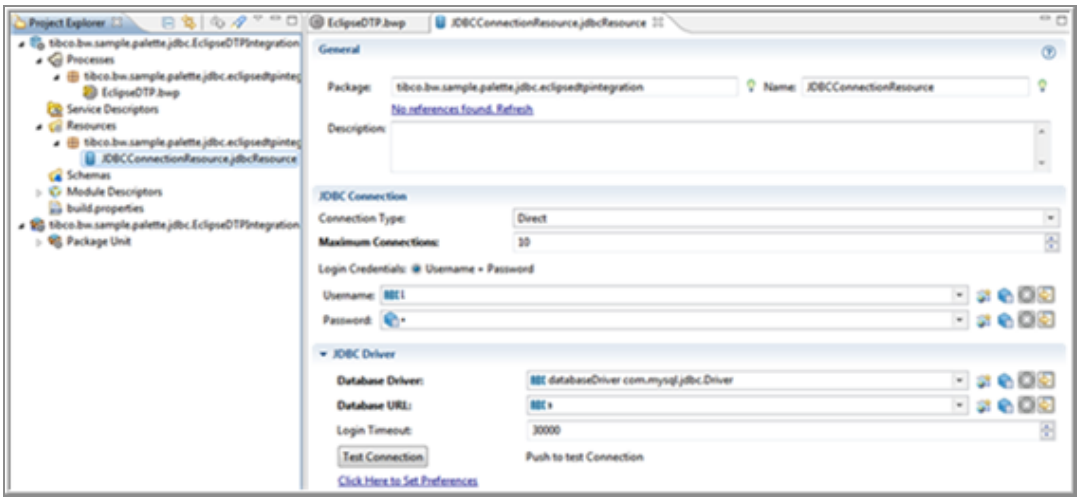
- Download and install the Sakila sample database from http://dev.mysql.com/doc/index-other.html.
- Installation information is available at http://dev.mysql.com/doc/sakila/en/sakila-installation.html.
- Ensure to install Drivers for Design time and Run time.

  For more information about installing drivers, see the "JDBC Connection" section of *TIBCO ActiveMatrix BusinessWorks™ Bindings and Palettes Reference* guide.

Procedure

1. In the **samples** directory, select **palette > jdbc > EclipseDTPIntegration** and double-click **tibco.bw.sample.palette.jdbc.EclipseDTPIntegration**. For more information, see Accessing Samples.
2. In the **Project Explorer** view, expand the **tibco.bw.sample.palette.jdbc.EclipseDTPIntegration** project.
3. Verify your JDBC connection.

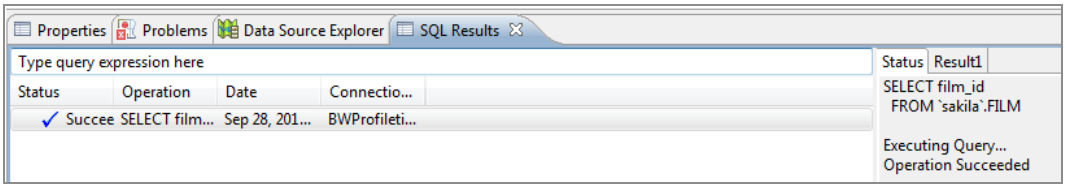
   a.



4. Fully expand the **Processes** directory and double-click **EclipseDTP.bwp**.
5. In the Process Editor, click **JDBCQuery** and then click **Properties > Statement** tab.
6. Click **SQL**. This launches the SQL Query Builder. In SQL Query Builder, update or edit

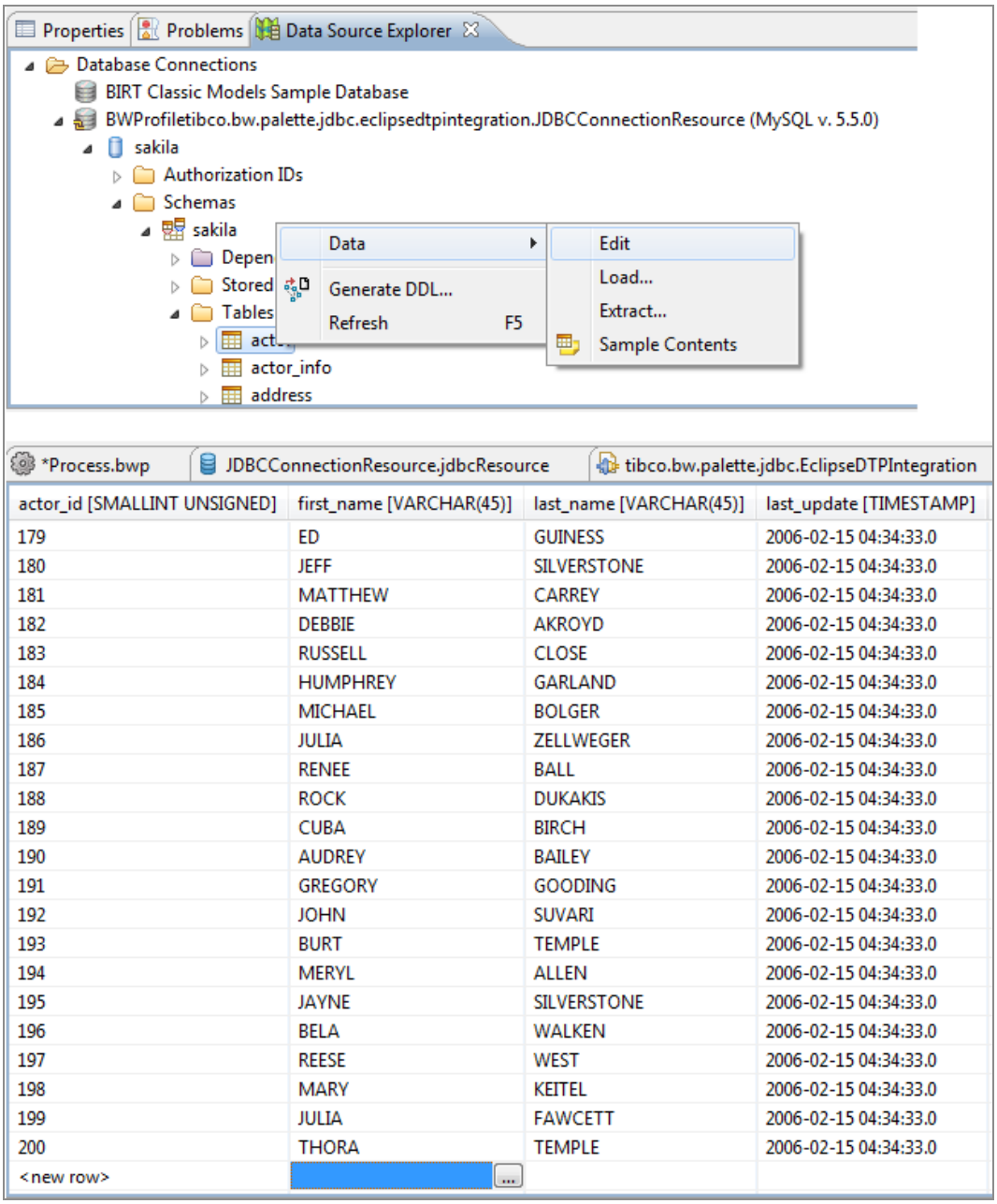
the statement, run it, and view the result.

a. Modify the statement by removing `"where title=?"` and click **Run the query** to run the statement at Design time by using DTP.



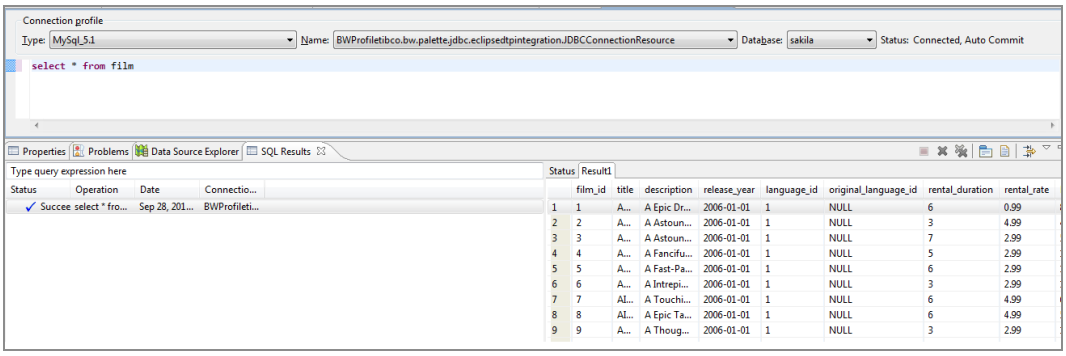
In the **SQL Results** tab, the history for all operations displays at the left and Status, Result, and Parameters display at the right.

b. Click the **Data Source Explorer** tab to view the connected database, **BWProfiletibco.bw.palette.jdbc.eclipsedtpintegration.JDBCConnectionResource**. Right-click the table and select **Data > Edit** to edit the data.



c. Open the scrapbook to edit and run the SQL statements by clicking the icon available in the **Data Source Explorer** view.

d. The entry for any SQL for default tables is available in the Sakila schema. Right-click and select **Execute All**. See the results in the **SQL Results** view.



e. Double-click **JDBCQuery**. Click **Statement** in the **Properties** tab. Modify the statement by adding "where title=?" at the end in statement.

7. Click **File > Save** to save the project.
8. Click **Run > Debug Configurations**.
9. At the left hand tree of Debug Configuration wizard, expand **BusinessWorks Application** and select **BWApplication**.
10. Click the **Applications** tab and then click the **Deselect All** button if you have multiple applications. Select the checkbox next to **tibco.bw.sample.palette.jdbc.EclipseDTPIntegration.application**.
11. Click **Debug**.

    This runs the sample in Debug mode.
12. Click the **Terminate** icon to stop the process.

Result

The console log contains messages similar to: INFO c.t.b.p.g.L.t.b.s.p.j.E.Log - FILM_ID-10001: Inventory of film id [17] in store id = [1] is 2.

Understanding the Configuration

The `EclipseDTP.bwp` process uses the **JDBC Connection** in the **Resources** folder to establish a connection to the database and runs SELECT and STORED PROCEDURE queries.

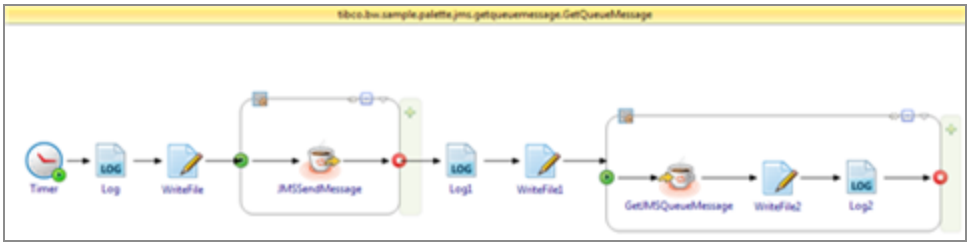
It queries the table for a specific record using the **JDBC Query** activity and then returns the available copies of a specified film in stock using the **JDBC Call Procedure** activity.

Finally it displays a number of records with MessageID in console.

Both activities use prepared parameters as Input. This shows the ability to run the same statement with multiple values by caching the statement at runtime.

Sending and Getting JMS Messages

A process can send JMS messages and get JMS messages from a queue in a loop.



Before you begin

TIBCO Enterprise Message Service must be running.

Procedure

1. In the **samples** directory, select **palette > jms > GetQueueMessage** and double-click **tibco.bw.sample.palette.jms.GetQueueMessage**. For more information, see Accessing Samples.
2. In the **Project Explorer** view, expand the **tibco.bw.sample.palette.jms.GetQueueMessage** project.
3. Verify your TIBCO Enterprise Message Service connection.
    a. Fully expand the **Resources** directory.
    b. Double-click **JMSConnectionResource.jmsConnResource**.
    c. In the Basic Configuration dialog, click the **Test Connection** button to verify the connection.
4. Fully expand the **Processes** directory and double-click **GetQueueMessage.bwp**.
5. Click **Run > Debug Configurations**.
6. At the left hand tree of Debug Configuration wizard, expand **BusinessWorks Application** and select **BWApplication**.
7. Click the **Applications** tab and then click the **Deselect All** button if you have multiple

applications. Select the checkbox next to **tibco.bw.sample.palette.jms.GetQueueMessage.application**.

8. Click **Debug**.

   This runs the sample in Debug mode.

9. Click the **Terminate** icon to stop the process.

Result

The Console displays messages similar to the following:

16:08:26.242 [PVM:In-Memory STWorkProcessor:1] INFO c.t.b.p.g.L.t.b.p.j.G.Log - Message=Sending 5 Queue Messages

16:08:26.292 [PVM:In-Memory STWorkProcessor:1] INFO c.t.b.p.g.L.t.b.p.j.G.Log1 - Message=Finished sending 5 Queue messages. Getting 5 Queue messages...

16:08:26.303 [PVM:In-Memory STWorkProcessor:3] INFO c.t.b.p.g.L.t.b.p.j.G.Log2 - Message=Received message number 1. Message = This is message number 1

16:08:26.307 [PVM:In-Memory STWorkProcessor:5] INFO c.t.b.p.g.L.t.b.p.j.G.Log2 - Message=Received message number 2. Message = This is message number 2

16:08:26.311 [PVM:In-Memory STWorkProcessor:1] INFO c.t.b.p.g.L.t.b.p.j.G.Log2 - Message=Received message number 3. Message = This is message number 3

16:08:26.315 [PVM:In-Memory STWorkProcessor:3] INFO c.t.b.p.g.L.t.b.p.j.G.Log2 - Message=Received message number 4. Message = This is message number 4

16:08:26.318 [PVM:In-Memory STWorkProcessor:5] INFO c.t.b.p.g.L.t.b.p.j.G.Log2 - Message=Received message number 5. Message = This is message number 5

Understanding the Configuration

The **JMS Send Message** activity sends the messages with the message style set to **Queue**.

The Get JMS Queue Message activity receives the JMS messages from a **Queue** in a loop. The activity is placed in a group and the group action is set to **Repeat**.

The condition loops five times and receives five messages.

Using JMS Message Selector

The **JMS Send Message** activity sends five queue messages and sets an application property with the index. The **Get JMS Queue Message** activity selects the third message using message selector.

Before you begin

TIBCO Enterprise Message Service must be running.

Procedure

1. In the **samples** directory, select **palette > jms > MessageSelector** and double-click **tibco.bw.sample.palette.jms.MessageSelector**. For more information, see Accessing Samples.
2. In the **Project Explorer** view, expand the **tibco.bw.sample.palette.jms.MessageSelector** project.
3. Verify your TIBCO Enterprise Message Service connection.
    a. Fully expand the **Resources** directory.
    b. Double-click **JMSConnectionResource.jmsConnResource**.
    c. In the Basic Configuration dialog, click the **Test Connection** button to verify the connection.
4. Fully expand the **Processes** directory and double-click **MessageSelector.bwp**.
5. Click **Run > Debug Configurations**.
6. At the left hand tree of Debug Configuration wizard, expand **BusinessWorks Application** and select **BWApplication**.
7. Click the **Applications** tab and then click the **Deselect All** button if you have multiple applications. Select the checkbox next to **tibco.bw.sample.palette.jms.MessageSelector.application**.
8. Click **Debug**.

    This runs the sample in Debug mode.
9. Click the **Terminate** icon to stop the process.

Result

The Console displays a message similar to the following:

11:54:01.283 INFO [bwThread:In-Memory STWorkProcessor-1] c.t.b.p.g.L.t.b.s.p.j.M.Log - Sending 5 Queue Messages

11:54:01.409 INFO [bwThread:In-Memory STWorkProcessor-1] c.t.b.p.g.L.t.b.s.p.j.M.Log1 - Finished sending 5 Queue messages. Selecting 3rd Message...

11:54:01.475 INFO [bwThread:In-Memory STWorkProcessor-2] c.t.b.p.g.L.t.b.s.p.j.M.Log2 - Selected 3rd message,. Message = This is message number 3

Understanding the Configuration

The process sends five Queue messages and selects one message using the selectors. This process definition runs as follows:

- The **JMS Send Message** activity sends five messages in a loop. It also sets the index of the loop to an application property called **myselector**.
- **The Get JMS Queue Message** activity uses the message selector **myselector='3'** to pick the third message.

Receiving a JMS Message from a Queue and Responding to the Message

The **JMS Queue Receiver** and **Reply to JMS Message** activities can be configured to receive and respond to JMS messages.

Before you begin

TIBCO Enterprise Message Service must be running.

Procedure

1. In the **samples** directory, select **palette > jms > QueueReceiver** and double-click **tibco.bw.sample.palette.jms.QueueReceiver**. For more information, see Accessing

Samples.

2. In the **Project Explorer** view, expand the **tibco.bw.sample.palette.jms.QueueReceiver** project.
3. Verify your TIBCO Enterprise Message Service connection.
    a. Fully expand the **Resources** directory.
    b. Double-click **JMSConnectionResource.jmsConnResource**.
    c. In the Basic Configuration dialog, click the **Test Connection** button to verify the connection.
4. Fully expand the **Processes** directory and double-click **QueueReceiver.bwp**.
5. Click **Run > Debug Configurations**.
6. At the left hand tree of Debug Configuration wizard, expand **BusinessWorks Application** and select **BWApplication**.
7. Click the **Applications** tab and then click the **Deselect All** button if you have multiple applications. Select the checkbox next to **tibco.bw.sample.palette.jms.QueueReceiver.application**.
8. Trigger the application by sending a request. You can use the **tibco.bw.sample.palette.jms.SenderAndSignalIn** sample to send the request. For more information, see Sending a JMS Message to a Queue and Waiting for a Response). The applications can run in the same or separate TIBCO Business Studio™ for BusinessWorks™ instance. If you run the applications in the same instance, load the application before clicking **Debug**.
9. Click **Debug**.

    This runs the sample in Debug mode.
10. Click the **Terminate** icon to stop the process.

Result

When both applications are running in the same TIBCO Business Studio for BusinessWorks instance, information similar to the following is displayed in the console:

14:32:46.402 [bwThread:In-Memory STWorkProcessor-2] INFO c.t.b.p.g.L.t.b.s.p.j.Q.Log - Received a Queue Message. Request = This is a test request. JMSMessageID = ID:EMS-SERVER.25FC52F2890A46:6. JobId [bw0a101], ProcessInstanceId [bw0a101], Activity [Log], Process [tibco.bw.sample.palette.jms.queuereceiver.Process], Module

[tibco.bw.sample.palette.jms.QueueReceiver:1.0.0.qualifier], Application [tibco.bw.sample.palette.jms.QueueReceiver.application:1.0].

14:32:46.419 [bwThread:In-Memory STWorkProcessor-3] INFO c.t.b.p.g.L.t.b.s.p.j.S.Log1 - Received the response. Response = This is a test response. JMSCorrelationID = ID:EMS-SERVER.25FC52F2890A46:6. JobId [bw0a100], ProcessInstanceId [bw0a100], Activity [Log1], Process [tibco.bw.sample.palette.jms.senderandsignalin.Process], Module [tibco.bw.sample.palette.jms.SenderAndSignalIn:1.0.0.qualifier], Application [tibco.bw.sample.palette.jms.SenderAndSignalIn.application:1.0].

Understanding the Configuration

The sample uses the **JMS Queue Receiver** activity to get messages off the queue. The **Reply to JMS Message** activity responds to the incoming JMS message and automatically uses the **JMSReplyTo** header to get the reply destination.

Sending a JMS Message and Receiving a Response in a Process

The **JMS Request Reply** activity sends a request to a JMS queue and waits for the response. After receiving the response, the activity makes the output available.

Before you begin

TIBCO Enterprise Message Service must be running.

Procedure

1. In the **samples** directory, select **palette > jms > RequestReply** and double-click **tibco.bw.sample.palette.jms.RequestReply**. For more information, see Accessing Samples.
2. In the **Project Explorer** view, expand the **tibco.bw.sample.palette.jms.RequestReply** project.
3. Verify your TIBCO Enterprise Message Service connection.
    a. Fully expand the **Resources** directory.

b. Double-click **JMSConnectionResource.jmsConnResource**.

c. In the Basic Configuration dialog, click the **Test Connection** button to verify the connection.

4. Fully expand the **Processes** directory and double-click **RequestReply.bwp**.
5. Click **Run > Debug Configurations**.
6. At the left hand tree of Debug Configuration wizard, expand **BusinessWorks Application** and select **BWApplication**.
7. Click the **Applications** tab and then click the **Deselect All** button if you have multiple applications. Select the checkbox next to **tibco.bw.sample.palette.jms.RequestReply.application**.
8. Trigger the application by sending a request. You can use the tibco.bw.sample.palette.jms.QueueReceiver sample to send the request. For more information, see Receiving a JMS Message from a Queue and Responding to the Message. The applications can run in the same or separate TIBCO Business Studio™ for BusinessWorks™ instance. If you run the applications in the same instance, load the application before clicking **Debug**.
9. Click **Debug**.

   This runs the sample in Debug mode.
10. Click the **Terminate** icon to stop the process.

Result

When both applications are run in the same TIBCO Business Studio for BusinessWorks instance, information similar to the following is displayed in the console:

14:41:00.532 [bwThread:In-Memory STWorkProcessor-2] INFO c.t.b.p.g.L.t.b.s.p.j.Q.Log - Received a Queue Message. Request = This is a test request. JMSMessageID = ID:EMS-SERVER.25FC52F2890A4F:5. JobId [bw0a101], ProcessInstanceId [bw0a101], Activity [Log], Process [tibco.bw.sample.palette.jms.queuereceiver.Process], Module [tibco.bw.sample.palette.jms.QueueReceiver:1.0.0.qualifier], Application [tibco.bw.sample.palette.jms.QueueReceiver.application:1.0].

14:41:00.548 [bwThread:In-Memory STWorkProcessor-3] INFO c.t.b.p.g.L.t.b.s.p.j.R.Log1 - Received the response. Response = This is a test response. JobId [bw0a100], ProcessInstanceId [bw0a100], Activity [Log1], Process [tibco.bw.sample.palette.jms.requestreply.Process], Module

[tibco.bw.sample.palette.jms.RequestReply:1.0.0.qualifier], Application [tibco.bw.sample.palette.jms.RequestReply.application:1.0].

Sending a JMS Message to a Queue and Waiting for a Response

After sending a JMS message, the JMSSignalIn activity gets the response and filters it as required.

Before you begin

TIBCO Enterprise Message Service must be running.

Procedure

1. In the **samples** directory, select **palette > jms > SenderAndSignalIn** and double-click **tibco.bw.sample.palette.jms.SenderAndSignalIn**. For more information, see Accessing Samples.
2. In the **Project Explorer** view, expand the **tibco.bw.sample.palette.jms.SenderAndSignalIn** project.
3. Verify your TIBCO Enterprise Message Service connection.
    a. Fully expand the **Resources** directory.
    b. Double-click **JMSConnectionResource.jmsConnResource**.
    c. In the **Basic Configuration** dialog, click the **Test Connection** button to verify the connection.
4. Fully expand the **Processes** directory and double-click **QueueSenderWait.bwp**.
5. Click **Run > Debug Configurations**.
6. At the left hand tree of Debug Configuration wizard, expand **BusinessWorks Application** and select **BWApplication**.
7. Click the **Applications** tab and then click the **Deselect All** button if you have multiple applications. Select the checkbox next to **tibco.bw.sample.palette.jms.SenderAndSignalIn.application**.
8. Trigger the application by sending a request. You can use the

tibco.bw.sample.palette.jms.QueueReceiver sample to send the request. The applications can run in the same or separate TIBCO Business Studio™ for BusinessWorks™ instance. If you run the applications in the same instance, load the application before clicking **Debug**.

9. Click **Debug**.

   This runs the sample in Debug mode.

10. Click the **Terminate** icon to stop the process.

Result

When running both applications in the same TIBCO Business Studio for BusinessWorks instance, the **Console** view displays the information similar to the following:

14:46:23.347 [bwThread:In-Memory STWorkProcessor-2] INFO c.t.b.p.g.L.t.b.s.p.j.Q.Log - Received a Queue Message. Request = This is a test request. JMSMessageID = ID:EMS-SERVER.25FC52F2890A57:6. JobId [bw0a101], ProcessInstanceId [bw0a101], Activity [Log], Process [tibco.bw.sample.palette.jms.queuereceiver.QueueReceiver], Module [tibco.bw.sample.palette.jms.QueueReceiver:1.0.0.qualifier], Application [tibco.bw.sample.palette.jms.QueueReceiver.application:1.0].

14:46:23.365 [bwThread:In-Memory STWorkProcessor-3] INFO c.t.b.p.g.L.t.b.s.p.j.S.Log1 - Received the response. Response = This is a test response. JMSCorrelationID = ID:EMS-SERVER.25FC52F2890A57:6. JobId [bw0a100], ProcessInstanceId [bw0a100], Activity [Log1], Process [tibco.bw.sample.palette.jms.SenderAndSignalIn.QueueSenderWait], Module [tibco.bw.sample.palette.jms.SenderAndSignalIn:1.0.0.qualifier], Application [tibco.bw.sample.palette.jms.SenderAndSignalIn.application:1.0].

Understanding the Configuration

The **QueueSenderWait** process sends a Queue message and listens for a response. This process definition runs as follows:

1. The **JMS Send Message** activity sends a Queue message. It also sets a static reply destination in the advanced tab. The responder uses this reply destination to reply the message.
2. The **JMS SignalIn** activity uses Conversation mechanism wherein a conversation is initiated by the Correlation Key, which is JMSMessageID sent by the **JMS Queue Sender** activity. Conversation "Join" uses JMSCorrelationID to filter the correct

response with the correct job.

Sending a JMS Message and Receiving a Durable Response

The **JMS Receiver** Event Source activity can be configured as a Durable Subscriber. If the subscriber is offline, the subscriber's messages are stored by the JMS server and delivered when the subscriber is online.

Before you begin

TIBCO Enterprise Message Service must be running.

Procedure

1. In the **samples** directory, select **palette > jms > TopicDurableSubscriber** and double-click **tibco.bw.sample.palette.jms.TopicDurableSubscriber**. For more information, see Accessing Samples.
2. In the **Project Explorer** view, expand the **tibco.bw.sample.palette.jms.TopicDurableSubscriber** project.
3. Verify your TIBCO Enterprise Message Service connection.
    a. Fully expand the **Resources** directory.
    b. Double-click **JMSConnectionResource.jmsConnResource**.
    c. In the Basic Configuration dialog, click the **Test Connection** button to verify the connection.
4. Fully expand the **Processes** directory and double-click **TopicPublisher.bwp**.
5. Click **Run > Debug Configurations**.
6. At the left hand tree of Debug Configuration wizard, expand **BusinessWorks Application** and select **BWApplication**.
7. Click the **Applications** tab and then click the **Deselect All** button if you have multiple applications. Select the checkbox next to **tibco.bw.sample.palette.jms.TopicDurableSubscriber.application**.
8. Click **Debug**.

The sample runs in Debug mode.

9. Click the **Terminate** icon to stop the process.

Result

The Console displays a message similar to the following:

15:46:34.903 [PVM:In-Memory STWorkProcessor:1] INFO c.t.b.p.g.L.t.b.p.j.D.Log - Message=Publishing a Topic Message, MessageId=

15:46:35.043 [PVM:In-Memory STWorkProcessor:2] INFO c.t.b.p.g.L.t.b.p.j.D.Log - Message=Received a Topic Message. Message = This is a test Topic Message, MessageId=.

Understanding the Configuration

The **JMS Send Message** activity sends a message to the `topic.sample` topic. The **JMS Receive Message** activity subscribes to the `topic.sample` topic. Additionally, the **JMS Receive Message** activity is configured as durable, allowing messages to be received later if the receiver is offline.

Sending Mail Using HTML

A process sends an email formatted in HTML and includes embedded images as an attachment.

Before you begin

You must have the account settings for your mail server.

Procedure

1. In the **samples** directory, select **palette > mail > HtmlMailWithImage** and double-click . For more information, see Accessing Samples.
2. In the **Project Explorer** view, expand the project.
3. Fully expand the **Processes** directory and double-click **Sender.bwp**.
4. Click **Run > Debug Configurations**.

5. At the left hand tree of Debug Configuration wizard, expand **BusinessWorks Application** and select **BWApplication**.
6. Click the **Applications** tab and then click the **Deselect All** button if you have multiple applications. Select the checkbox next to .
7. Click **Debug**.

   This runs the sample in Debug mode.
8. Click the **Terminate** icon to stop the process.

Result

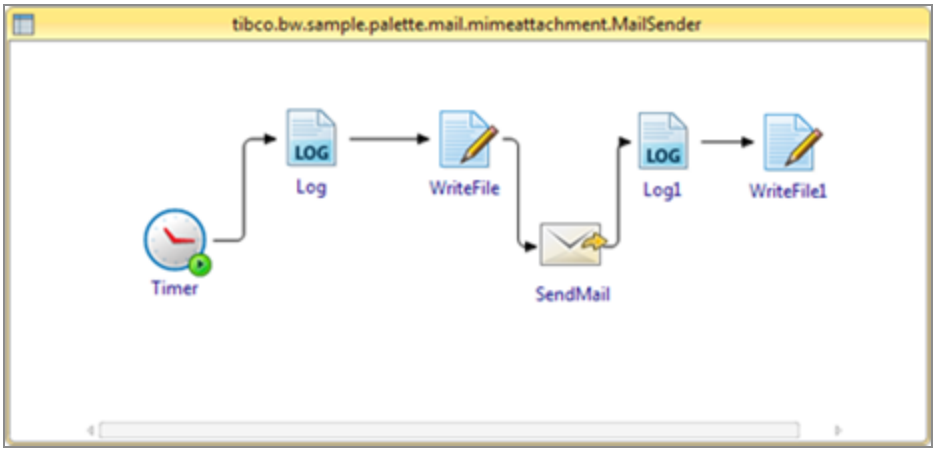
Your mail client receives an email showing the attachments, embedded in an HTML formatted message.

Sending Mail Using MIME

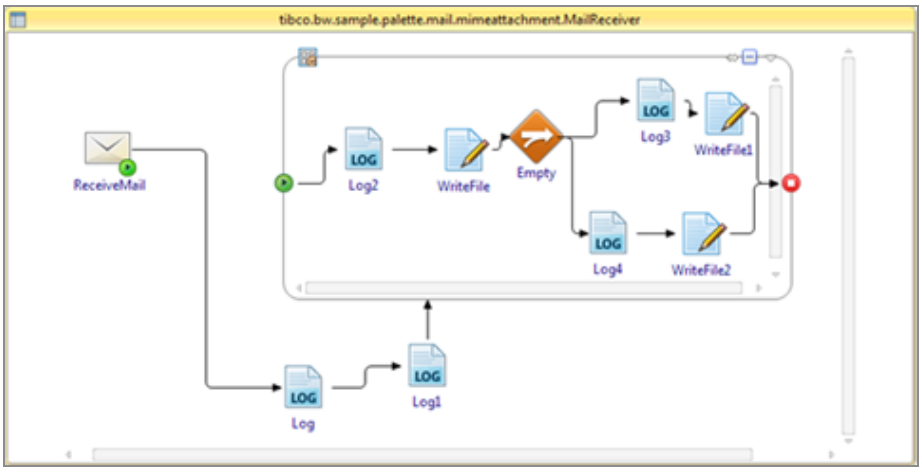
This sample shows how to send mail with attachments using MIME format.

The **MailSender.bwp** process sends mail with attachments and the **MailReceiver.bwp** process receives the mail.

MailSender Process



MailReceiver Process



Before you begin

You must have the account settings for your mail server.

On Windows, copy the `input` folder from `$TIBCO_HOME\bw\n.n\samples\palette\mail\attachment_files` folder to `C:\tmp`.

On UNIX, copy the `input` folder to `/tmp`.

Procedure

1. In the **samples** directory, select **palette > mail > MIMEAttachment** and double-click **tibco.bw.sample.palette.mail.MIMEAttachment**. For more information, see Accessing Samples.
2. In the **Project Explorer** view, expand the **tibco.bw.sample.palette.mail.MIMEAttachment** project.
3. Set the default **ApplicationProfile** to match the OS you are running on. For more information, see Setting the Default Application Profile.
4. Fully expand the **Processes** directory and double-click **MailSender.bwp** and **MailReceiver.bwp**.
5. Click **Run > Debug Configurations**.
6. At the left hand tree of Debug Configuration wizard, expand **BusinessWorks Application** and select **BWApplication**.
7. Click the **Applications** tab and then click the **Deselect All** button if you have multiple applications. Select the checkbox next to **tibco.bw.sample.palette.mail.MIMEAttachment.application**.

8. Click **Debug**.

   This runs the sample in Debug mode.

9. Click the **Terminate** icon to stop the process.

Result

The MailSender process sends the mail and the MailReceiver process receives the mail.

On Windows, the `MIMEAttachment.log` output file is created at `c:/tmp/MIMEAttachment`.

On UNIX, the output file is created at `\tmp\MIMEAttachment`.

The **Console** view shows the messages similar to:

13:25:01.277 INFO [bwThread:In-Memory STWorkProcessor-1] c.t.b.p.g.L.t.b.s.p.m.M.Log - Sending the mail with attachment

13:25:03.984 INFO [bwThread:In-Memory STWorkProcessor-1] c.t.b.p.g.L.t.b.s.p.m.M.Log1 - Mail has been sent with attachment

13:25:13.471 INFO [bwThread:In-Memory STWorkProcessor-2] c.t.b.p.g.L.t.b.s.p.m.M.Log - Received Mail with subject: Test Mail With Attachment

13:25:13.476 INFO [bwThread:In-Memory STWorkProcessor-2] c.t.b.p.g.L.t.b.s.p.m.M.Log1 - Mail Body: Hello World

13:25:13.538 INFO [bwThread:In-Memory STWorkProcessor-2] c.t.b.p.g.L.t.b.s.p.m.M.Log2 - Attachment [1] data:

13:25:13.591 INFO [bwThread:In-Memory STWorkProcessor-2] c.t.b.p.g.L.t.b.s.p.m.M.Log3 - Test Attachment Data

13:25:13.631 INFO [bwThread:In-Memory STWorkProcessor-2] c.t.b.p.g.L.t.b.s.p.m.M.Log2 - Attachment [2] data:

13:25:13.656 INFO [bwThread:In-Memory STWorkProcessor-2] c.t.b.p.g.L.t.b.s.p.m.M.Log4 - R0lGODlheQA2APcAAAAAAP///wBtpQ8KCxALDBMODxYREhcSExoVFhAMDRMPEBQQERgUFRkVFhoWFxANDhEODxUSExYTFBcUFRkWFx0aGyAdHiMgISYjJCUiIykmJxgTFRURExsXGS0rLDAuLzg2NzQyMz89Pjw6O0hGR0RCQ09NTlxaWyckJionKSIeITIwMjAuMDg2ODUzNfr2+jY1NlRTVEJBQlxbXGlnajMyNEhHSUVERkRDRUNCRFJRU1BPUUpJS2NiZGBfYV5dX1hXWbSztXRzdu3r8z8/Qk1NUEdHSVJSVPX1+pKSlV5eYFpaXFlZW1VVV3Z2eHJydGlpa2ZmaGVlZ5mZm5GRk4qKjIeHiYWFh4KChIGBg35+gHFxcm1tbmFhYldXWP39/vv7/Pr6+/n5+vb29/X19u/v8Nzc3dHR0piYmerr9O3u9unr82tsb4GChZSVmLu8v93i7uPn8evu9ebq8ltcXmFiZHd4enR1d3N0dm5vcY2OkIiJi7q7vbO0tq2usKytr6mqrKipq6Slp6KjpaCho56foZydn5ucnsTFx7/AwszX6JianfHz9pav0L7O4qe/2XScxISoy

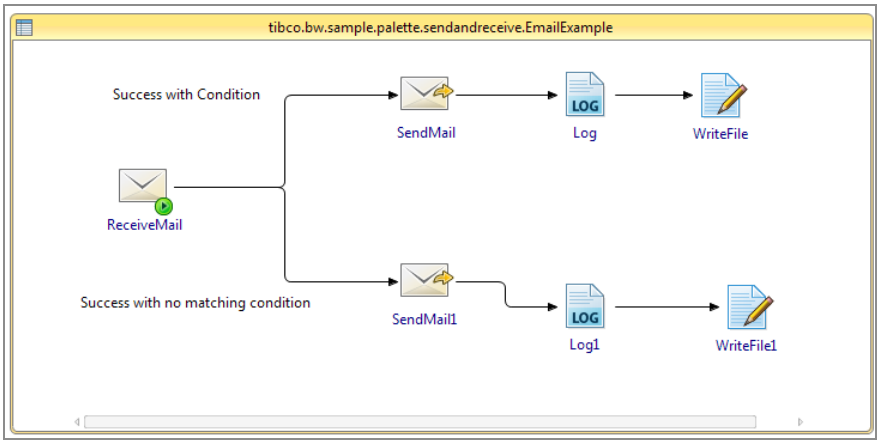
46uzo6QkouNj4WHiYKEhpaYmkhJSq+xs19gYXx9fnl6e2prbKeoqZmam5WWl5SVloyNjoWGh/X
29/Hy8/Dx8uvs7enq6+bn6OXm5+Pk5eLj5N/g4d7f4N3e39zd3trb3Nna29fY2dbX2NTV1tPU1dLT
1M/Q0c7P0M3Oz8rLzMjJysbHyMPExcLDxLu8vbm6u7e4uba3uLKztLCxsqusra3F3KC+2GKXwW
aawmmcw4200ZS300mLuV6XwHuqy4OwzwBjnwBfnSJ4rC58rzqGtVOTvQBnogBoogBooQBm
oQBloANpogVqow5vpxFwpxdzqSl+sABspQBrpABqo7y+v7S2t97g4drc3cfJygBxqO/x8f7///z9/fv
8/Pj5+ff4+PT19fP09PDx8e3u7ujp6czNzfj6+fHy8QwICP7+/vv7+/Pz8+vr6////yH5BAEAAP8ALAAA
AAB5ADYAAAj/AAMIHEiwoMGDCBMqXMiwocOHECNKnEixosWLGDNq3Mixo8ePIEOKHEmypM
mTKFOqXMmypcuXMGPKnPlwmc2bNpU9avSMmaNlynAmcyQw2CYsSJMqxbLpj8BZnAQNRH
Uli9Ikw1YpLDNs0w4PGD7w2EKIDMFWhaSMyPChSZV7Xwia8dPHXsVx2fLqzYYNXKRq05xFAictL
zZpzARmIXCg8QEGDTY4PjCgk0BhCWYMjGVhgwTHCiS4EBXv4KASEhS0/XEkxIEOZgSKmbLigQ
MZSopkIKCCSyqB9iYdcuOm4joByJMjL1cNGTVu1x6J0yZg3LZmAlv14sX9zKYIec7s/+KOSJZAYB
R6DISF4oO58b4yOblwoErBeVgmWNgSzJ6Yd2Kc4osb/gTwhRb6CbHLPF/gIwsVLuijAysB4OKHI
IQQYpxyHG4T3TfbWOOIM9sIcB1ChySwB0LoqScQeyGUUhAxGWhg3kBaPDBCIgsd0oAG5xwUS
wkJABFGKnu84UklG3KY3DjcQBNJOdpYo4gz2GCD3UFTPHAFi+mth0IIZRT0jg8H+DEQIA18kMt
CtXywwSEJ0QKCBIAE0EsggNTSpJPIkROOItaQg42V1EizpUFdfnlQi2KSaRAnC2gYQD0ydBAIQ1c
M0EUYCoGigA5mueOORccBmpw2yEDDjQDcRP9yjKIINQqmiwHAWCZBX0RhQJ4B9MGADvMsZ
AoPDgSpECshVMAjRqmqKkA2kiAjjQDlhONINYlx6eWtke46kC4fYHCLQJw8AApDsXiwgioMCaFA
cdBKi9w2xzRDnQCIQRJNrd8+GuaLY5oy0BeJuPAAFgKR0oSzDAVxgBcNEfKAHRlFC2g21cThTD
n3LtPIv946ahCkL2qggR1abCLEESnYUIhZAZwiAwZ+LoRGAU80FAwHuKIqLTndHPMIyMhJA40k
JDMa8MkD55qCBQ4g0EADDOxHoUCruJDCKwzpocAmDfXSAMX1qqqlIt8gPY43jlzTdEG2Cowre
yPwcosuvWT/ooUHGkzRTgD2jGAjQ0kUgDFDiUzwQ8aqejiENdkkh801cIDTrdMmF4RyrhiUMEZ
BuYzgADEBHKsCLwx5okDQCRlTQB6QO3moGsmUmBw5ijQzzaJ0P+151LBgIEOxBRkjQRNxScG
BMQzpooIIMi5kRQErpq1cOdTAwQw241j+TByJAk9Q3VDfbTzyBMnCAghagZLAFgyhIsIF6ixEBg8
RqKn9cuBQhCO8QY7kZAsO0NiGlgDWOYJ8rnjHM4grXBACVwTgFhr4wI0U0okBNNAg5+hAC+D
1P7wgYw7UQBpysIEMOHyDHCcqGbhetD6D/MICOGCHQLZAAB+QYiHCqIAH/3aRkFIw4QFW0Ei
0oBQNNVjjVcnJhjNecI1tWMd8A0Hf8NRXAnkUhAxdGADDBJKKHDxgE6NLiDugQAAekLAgX+BE
AVxgwdqZKBkveIbulkONODwCfFdk4AxBl4NStMMdpXhFMXago1YQ5B4jUMAR+MA+gYCBFv0IQ
Cow8QAbCOMdBLkFFAyggWFsZFrZ4IYzkACNwuyFG82QAzW2kRdaHQQU+hijQYKhgMcJJBYqu
IAJdmACGXhgAg5gwiwMIgsmMKADN8jCJwoxBSs0gQWxCYAsgGAAB8RAD4RAQycwkIAQ9IEj4
hAHNZwhB0VUI53wpIYyAiAJasTzEQgRhAeSgP+QX9TgDgNxRQ5GAAMXuGAEOrjCL+CBkHjwQ
QorUIEDJNABFXwACvUYyBg8oQMMVEACDrjACPbwRo2kYQ1zYEQ71BCHOazhpSh9wRdcCtM5I
AEh+UBHRg8yhlYUSCBhYMUqhsoKe4CKIe5oBS+GUQxh8KIVRyWIPGxhjmIMYxenoIlWtRqP6m
01APuoBSq0WohLlNUQpkgE2RpShkK4IRA7JckUeIAIgryCEBltBeog4gtB5M8ixRiECEKRCTIMgw
sGIog7ojqQW+DAE1GIwlHfAUqwHmQflqzsQPhhkC9EdQwxICJBzgEBSgRAGCQYyD44a9mD5II
SbriEIy0SBk3/1BEYR6BEJ/Y6Cy1sYVMEyUUUAhALI8DrD3nIgylZMQkxrKITtgiAG85FjCc8IX/zqIId
QlGQXzyBC4UIwDuSEAI26IIgfVhCD1hhDhMI5B5PyEMS3gGGKtAiAFqQSh/8cIZJDGIRsLjIPGaQD
oHwYQSy+MURyuCOJQRCFUzoBUFoUQJRNKFniLCBK2hhhHu8QwmuIAYRCNGPGNjDFUuQhT
nowI8ywAANNBPIK0jwi1M0QSqv0EEwfjgQY2QhE6NAxA4CQI8m+IEUUaCTFQphCjrgIQBPCEYA

ggEIPmBkwAUOwDlooEkmpMIeIrhCEvLwLIHQ4gaTMEY+AhAKJwgE/wpoCIAoLjEJQGghE2RLR
BHcQAk70MMfR5jtQIShBM5OArHxmEF0CWIMJ7QDCpU4wQU1URpjMCEAiMhCH/5ACV/YoUxf6
EdcBAyEWAikD4hdRRNUEQ8eZCIAZRBDcH3AUIEM4whkKMMOpMwLHrShHpw4wi8qpANHlsk
eQNAKQWzBA7BFgZ/yAMIZCgII+gVjBRSzRxPepAX7nCIPUmiFMehwiY7IYwuzTUQWNMkGeAGj
B06whLgCQIs7lEYg/BAFGzoxhbjEgwmGCIAgiAVWKvRgC+H9dkkFMogobCELmSTDE5Y52iR+AQ
9SEEgxevCEO+xKFHXQ5g0k/NWSm/zkKBtPucpXzvKWu/zlMI+5zGdO85rb/OY4zzlHAgIAOw==

Sending and Receiving Mail

A process sends an email to a user and another process receives the email, by either sending a response to the sender, or writing it to a log in the console.

This process contains **Send Mail** and **Receive Mail** activities.



Before you begin

You must have the account settings for your mail server.

Procedure

1. In the **samples** directory, select **palette > mail > SendAndReceive** and double-click **tibco.bw.sample.palette.mail.SendAndReceive**. For more information, see Accessing Samples.
2. In the **Project Explorer** view, expand the **tibco.bw.sample.palette.mail.SendAndReceive** project.
3. Set the default **ApplicationProfile** to match the OS you are running on. For more information, see Setting the Default Application Profile.

4. Fully expand the **Processes** directory and double-click **EmailExample.bwp** and **MailSendReceive.bwp**.
5. Click **Run > Debug Configurations**.
6. At left hand tree of Debug Configuration wizard, expand **BusinessWorks Application** and select **BWApplication**.
7. Click the **Applications** tab and then click the **Deselect All** button if you have multiple applications. Select the checkbox next to **tibco.bw.sample.palette.mail.SendAndReceive.application**.
8. Click **Debug**.

   This runs the sample in the Debug mode.
9. Click the **Terminate** icon to stop the process.

Result

The `SendAndReceive.log` output file is created in **SendAndReceive** folder at `c:/tmp`

Depending on your actions:

- If userA sends an email to the user configured in the **Receive Incoming Email** task with the subject "Send Response from BW", the mail account specified by the **reply_email_address** module property receives an email and the following message is visible in the console:

13:49:35.053 INFO [bwThread:In-Memory STWorkProcessor-2] c.t.b.p.g.L.t.b.s.p.m.S.Log

- Mail sent to user. 13:49:39.613 INFO [bwThread:In-Memory STWorkProcessor-3] c.t.b.p.g.L.t.b.s.p.m.S.Log
- Response sent to user1.
- If userA sends an email to the user configured in the **Receive Incoming Email** task with another subject, the console shows the following message:

13:49:35.053 INFO [bwThread:In-Memory STWorkProcessor-2] c.t.b.p.g.L.t.b.s.p.m.S.Log

- Mail sent to user. 13:49:39.613 INFO [bwThread:In-Memory STWorkProcessor-3] c.t.b.p.g.L.t.b.s.p.m.S.Log -No subject match

Troubleshooting

If the process does not receive the mail, verify the following whether:

- The email server and user information are correctly configured for your system.
- The reply_email_address module property refers to the mail account you are checking.

If the process after receiving the mail did not send a response, verify the following whether:

- The subject used in the email matches the incoming_email_subject module property. If it does not match, then find an appropriate message in the console.
- You have specified the Host correctly. If not, then it generates an error.

Publishing and Subscribing to TIBCO Rendezvous Messages

Rendezvous programs use messages as the common currency to exchange data. The Rendezvous palette includes activities that allow you to easily setup subjects to send and receive messages. This sample shows how to use Rendezvous activities.

Before you begin

TIBCO Rendezvous™ must be running on the machine.

To run this sample you can either use TIBCO ActiveMatrix BusinessWorks™ 6.x client or ActiveMatrix BusinessWorks™ 5.x as a client.

For more information about how to configure client in TIBCO ActiveMatrix BusinessWorks 5.x, see Configure the Client in TIBCO Designer section.

Procedure

1. In the **samples** directory, select **palette > rendezvous > RPC** and double-click **tibco.bw.sample.palette.rendezvous.RPC**. For more information, see Accessing Samples.
2. In the **Project Explorer** view, expand the **tibco.bw.sample.palette.rendezvous.RPC** project.

3. Set the default **ApplicationProfile** to match the OS you are running on. For more information, see Setting the Default Application Profile.
4. Fully expand the **Processes** directory and double-click **PublishRVmessageProcess.bwp**, **RendezvousRPC.bwp**, and **SubscribeRVMessageProcess.bwp**.
5. Click **Run > Debug Configurations**.
6. At the left hand tree of Debug Configuration wizard, expand **BusinessWorks Application** and select **BWApplication**.
7. Click the **Applications** tab and then click the **Deselect All** button if you have multiple applications. Select the checkbox next to **tibco.bw.sample.palette.rendezvous.rpc.application**.
8. Click **Debug**.

   This runs the sample in Debug mode.

   Configure the Client in TIBCO Designer
9. In TIBCO Designer, import the client project, `TIBCO_HOME\bw\n.n\samples\palette\rv\RPC\bw5\tibco.bw.sample.palette.rendezvous.RPCClient`.
10. Click **Global Variables** and select the **OUTPUT_FILE** variables to get the location of the output file.
11. Click **Tools > Tester > Start**.
12. In TIBCO Business Studio™ for BusinessWorks™, click the **Terminate** icon to stop the process.

Result

The **Console** view displays the message similar to the following:

10:49:55.611 INFO [bwThread:In-Memory STWorkProcessor-1] c.t.b.p.g.L.t.b.s.p.r.rpc.Log - Message Published to BW.Start

10:49:57.745 INFO [bwThread:In-Memory STWorkProcessor-7] c.t.b.p.g.L.t.b.s.p.r.rpc.Log - TestCase Completed

10:50:02.759 INFO [bwThread:In-Memory STWorkProcessor-5] c.t.b.p.g.L.t.b.s.p.r.rpc.Log - Recieved Response HELLO WORLD

The `output.log` file is generated at `C:\tmp\RPC` that contains the following data:

Message Published to BW.Start

Subscriber to Bw.Start invoked and Message Published on BW.Results

Process completed.Final output is--->>HELLO WORLD

Understanding the Configuration

The **Subscribe to BW.Start** Rendezvous Subscriber activity listens on the **BW.Start** subject. On receiving the message on that subject, the process starts the:

- **Wait_for_BW.Request** activity that is listening to the BW.Request subject.
- **Reply_to_BW.Request** activity that is listening on the **BW.Request** subject and **BW.Reply** subject.

The **Wait for Rendezvous Request** activity receives the message from the **Send Rendezvous Request** activity and sends a reply message to the **Rendezvous Request** activity. A final result message is delivered by the publisher on the subject BW.Result.

Managing TCP Connections

The TCP activities are designed to work with custom applications that communicate with other applications via TCP. This sample shows how to perform common tasks, such as opening a connection, writing data to it, and closing the connection.

Procedure

1. In the **samples** directory, select **palette > tcp > Basic** and double-click **tibco.bw.sample.palette.tcp.Basic**. For more information, see Accessing Samples.
2. In the **Project Explorer** view, expand the **tibco.bw.sample.palette.tcp.Basic** project.
3. Set the default **ApplicationProfile** to match the OS you are running on. For more information, see Setting the Default Application Profile.
4. Fully expand the **Processes** directory. See multiple sample sub packages, each comprising a **Client.bwp** and **Server.bwp** process. Double-click each process to view the activities defined.

5. Click **Run > Debug Configurations**.
6. At the left hand tree of Debug Configuration wizard, expand **BusinessWorks Application** and select **BWApplication**.
7. Click the **Applications** tab and then click the **Deselect All** button if you have multiple applications. Select the checkbox next to **tibco.bw.sample.palette.tcp.Basic.application**.
8. Click **Debug**.

   This runs the sample in Debug mode.
9. Click the **Terminate** icon to stop the process.

Result

The `TCP_Basic.log` file is generated at `C:\tmp\TCP_Basic` contains the following data.

Test Request Data by iterationLoop- 1

Test Request Data by iterationLoop- 2

Test Request Data by iterationLoop- 3

Test Request Data by iterationLoop- 4

Test Request Data by noSeparator

Test Reply Data by noSeparator

Test Request Data by preDefinedSeparator

Test Reply Data by preDefinedSeparator

Test Request Data by userDefinedSeparator

Test Reply Data by userDefinedSeparator

Test Request Data by binarySeparator

Test Reply Data by binarySeparator

The Console view displays the output similar to the following.

10:40:20.505 INFO [bwThread:In-Memory STWorkProcessor-1]
c.t.b.p.g.L.t.b.s.p.t.B.PrintRequest - Test Request Data by iterationLoop- 1

10:40:20.545 INFO [bwThread:In-Memory STWorkProcessor-5]
c.t.b.p.g.L.t.b.s.p.t.B.PrintRequest - Test Request Data by iterationLoop- 2

10:40:20.568 INFO [bwThread:In-Memory STWorkProcessor-7]
c.t.b.p.g.L.t.b.s.p.t.B.PrintRequest - Test Request Data by iterationLoop- 3

10:40:20.593 INFO [bwThread:In-Memory STWorkProcessor-8]
c.t.b.p.g.L.t.b.s.p.t.B.PrintRequest - Test Request Data by iterationLoop- 4

10:40:20.618 INFO [bwThread:In-Memory STWorkProcessor-2]
c.t.b.p.g.L.t.b.s.p.t.B.PrintRequest -

10:40:21.447 INFO [bwThread:In-Memory STWorkProcessor-1]
c.t.b.p.g.L.t.b.s.p.t.B.PrintRequest - Test Request Data by noSeparator

10:40:21.491 INFO [bwThread:In-Memory STWorkProcessor-7]
c.t.b.p.g.L.t.b.s.p.t.B.PrintReply - Test Reply Data by noSeparator

10:40:22.428 INFO [bwThread:In-Memory STWorkProcessor-4]
c.t.b.p.g.L.t.b.s.p.t.B.PrintRequest - Test Request Data by preDefinedSeparator

10:40:22.460 INFO [bwThread:In-Memory STWorkProcessor-3]
c.t.b.p.g.L.t.b.s.p.t.B.PrintReply - Test Reply Data by preDefinedSeparator

10:40:23.429 INFO [bwThread:In-Memory STWorkProcessor-2]
c.t.b.p.g.L.t.b.s.p.t.B.PrintRequest - Test Request Data by userDefinedSeparator

10:40:23.445 INFO [bwThread:In-Memory STWorkProcessor-6]
c.t.b.p.g.L.t.b.s.p.t.B.PrintReply - Test Reply Data by userDefinedSeparator

10:40:24.429 INFO [bwThread:In-Memory STWorkProcessor-7]
c.t.b.p.g.L.t.b.s.p.t.B.PrintRequest - Test Request Data by binarySeparator

10:40:24.467 INFO [bwThread:In-Memory STWorkProcessor-8]
c.t.b.p.g.L.t.b.s.p.t.B.PrintReply - Test Reply Data by binarySeparator

Understanding the Configuration

The following TCP palette activities are used in this sample:

- **TCP Receiver**: The process starter starts a new process, when a client requests a TCP connection.
- **Wait for TCP Request**: The activity waits for a TCP client connection request.
- **TCP Open Connection**: The activity opens a connection to a TCP server.

- **TCP Close Connection**: The activity closes a TCP connection opened by a previously executed activity or process starter.
- **Write TCP Data**: The activity sends data on the specified TCP connection.
- **Read TCP Data**: The activity reads data from an open TCP connection.

The processes show the use of **Binary** and **Text** modes.

Binary mode:

- The **binarySeparator** sub-package shows the use of user-defined binary separator in **binary** mode.
- The **iterationLoop** sub-package shows the use of an iteration loop.
- The **noSeparator** sub-package shows the use of no separator in **binary** mode.

Text mode:

- The **preDefinedSeparator** sub-package shows the use of a pre-defined text separator in **text** mode.
- The **userDefinedSeparator** sub-package shows the use of a user-defined text separator in **text** mode.

Sharing a TCP Connection across Multiple Processes

A TCP connection can be shared across multiple processes or jobs. This simple sample shows the connection sharing across jobs. In a real-world scenario, the application implementation would handle the ordering or synchronization of request and responses when the connection is shared.

The process contains:

- **Server**: Receives multiple requests from the client processes and logs the request.
- **Initialize Connection**: Opens a TCP Connection to the server and sets the connection to a Shared variable.
- **Client A/Client B/Client C**: Client processes that use the shared connection to send requests.

Procedure

1. In the **samples** directory, select **palette > tcp > SharedTCPConnection** and double-click **tibco.bw.sample.palette.tcp.SharedTCPConnection**. For more information, see Accessing Samples.
2. In the **Project Explorer** view, expand the **tibco.bw.sample.palette.tcp.SharedTCPConnection** project.
3. Set the default **ApplicationProfile** to match the OS you are running on. For more information, see Setting the Default Application Profile.
4. Fully expand the **Processes** directory. Two sub-packages are available, one containing the server and another containing three client processes and the initialize TCP connection process. Explore each process by double-clicking it.
5. Click **Run > Debug Configurations**.
6. At the left hand tree of Debug Configuration wizard, expand **BusinessWorks Application** and select **BWApplication**.
7. Click the **Applications** tab and then click the **Deselect All** button if you have multiple applications. Select the checkbox next to **tibco.bw.sample.palette.tcp.SharedTCPConnection.application**.
8. Click **Debug**.

   This runs the sample in Debug mode.
9. Click the **Terminate** icon to stop the process.

Result

The result, similar to the following, is written to the `SharedTCPConnection.log` file at `C:\tmp\SharedTCPConnection`.

Text 1 from Client A

Text 2 from Client B

Text 3 from Client C

Understanding the Configuration

The following properties are defined and the default values can be changed:

- The **TCPConnectionProperty** that points to the TCP shared resource connection.
- The **FilenameProperty** that points to the Output_File. This file is created by the sample process and contains the output. The default value is `C:\tmp\SharedTCPConnection\SharedTCPConnection.log`.

Inter Process Communication using Wait for Notification and Notify Activities

This sample explains inter-process communication using the **Notify** and the **Wait for Notification** activities in different processes. One process notifies the other waiting process using these activities. These activities are co-related using the key configured on the **General** tab and the same **Notify Configuration** resource configured for both the activities.

Procedure

1. In the **samples** directory, select **palette > generalactivities > WaitNotify** and double-click **tibco.bw.sample.palette.generalactivities.WaitNotify**. For more information, see Accessing Samples.
2. In the **Project Explorer** view, expand the **tibco.bw.sample.palette.generalactivities.WaitNotify** project.
3. Set the default **ApplicationProfile** to match the OS you are running on. For more information, see Setting the Default Application Profile.
4. Fully expand the **Processes** directory.
5. Click **Run > Debug Configurations**.
6. At the left hand tree of **Debug Configuration** wizard, expand **BusinessWorks Application** and select **BWApplication**.
7. Click the **Applications** tab and then click the **Deselect All** button if you have multiple applications. Select the checkbox next to **tibco.bw.sample.palette.generalactivities.WaitNotify.application**.
8. Click **Debug**.

   This runs the sample in Debug mode.
9. Click the **Terminate** icon to stop the process.

Result

A `waitnotify.log` file is created in the `C:\tmp\WaitNotify` folder of the default location, or the path you specified earlier. This file contains the information similar to the following:

Main Process: Started the main process

Main Process: Spawning non-inline sub process

Main Process: Activity Waiting for the notification

Sub Process : Non-inline sub process started

Sub Process : Notification sent for the waiting activity

Main Process: Notification Received from the other process with Data: Results sent from the notifying process

Understanding the Configuration

The scenario depicts a simple inter process communication use case.

- The main process is triggered through a **Timer** activity.
- The main process spawns a non-inline sub process.
- The main process then starts waiting, using a **Wait for Notification** activity.
- The spawned sub process sends the notification for the waiting activity using a **Notify** activity and proceeds further.
- The **Wait for Notification** activity in the main process receives the notification and proceeds further.

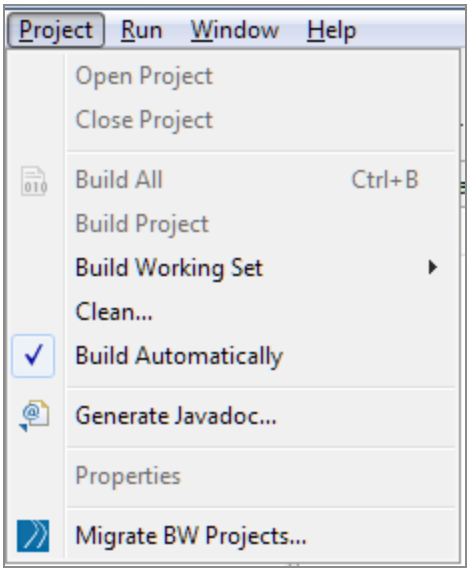
Migrating Projects

ActiveMatrix BusinessWorks 6.x provides an easy way to migrate ActiveMatrix BusinessWorks 5.x projects by using the **Migrate BW Projects** menu option in TIBCO Business Studio for BusinessWorks. You can also use the Eclipse import mechanism to migrate the projects.

Migrating the Project Using TIBCO Business Studio for BusinessWorks

Procedure

1. In TIBCO Business Studio for BusinessWorks, go to **Project > Migrate BW Projects...**.



2. In the BusinessWorks Migration Tool wizard, specify the relevant information in the following fields.
    a. Click **Browse** in the **BusinessWorks 5 Project Folder** field and provide the location of the ActiveMatrix BusinessWorks 5.x project that you want to migrate. The tool scans the ActiveMatrix BusinessWorks 5.x projects in the specified folder location and displays the list of projects.
    b. In the **Migrated Project Folder** field, click **Browse** and provide the location where you want to migrate the project.

c. Select the **Add External JARs** checkbox if the ActiveMatrix BusinessWorks 5.x project is using external JAR files in context with Java activities.

   It is a best practice to place the external JAR files in a single location. These JAR files can then be in-lined in the migrated ActiveMatrix BusinessWorks 6.x project.

d. Click **Browse** in the **Jar Location** field and provide the location of the JAR files on disk.

3. In the **BusinessWorks 5 Projects** area, select the projects you want to migrate and click **Migrate**.

   The migrated projects are visible in the **Migration Output** area.

4. Click **Close** after the migration is complete.

Note: You can alternatively migrate the projects using the **File > Import** menu option. Select **Migrate BW Projects... > Next** in the Import dialog.

Manual Task during Migration

Some of the samples require certain manual steps to be performed while migrating the samples. You can refer to the steps mentioned in the listed samples.

HTTP Basic Sample

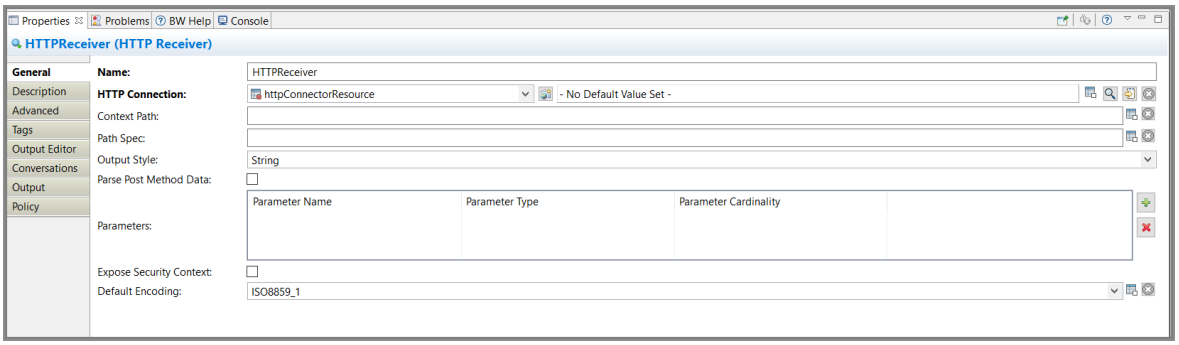
This section describes how to migrate an HTTP Basic sample.

Before you begin

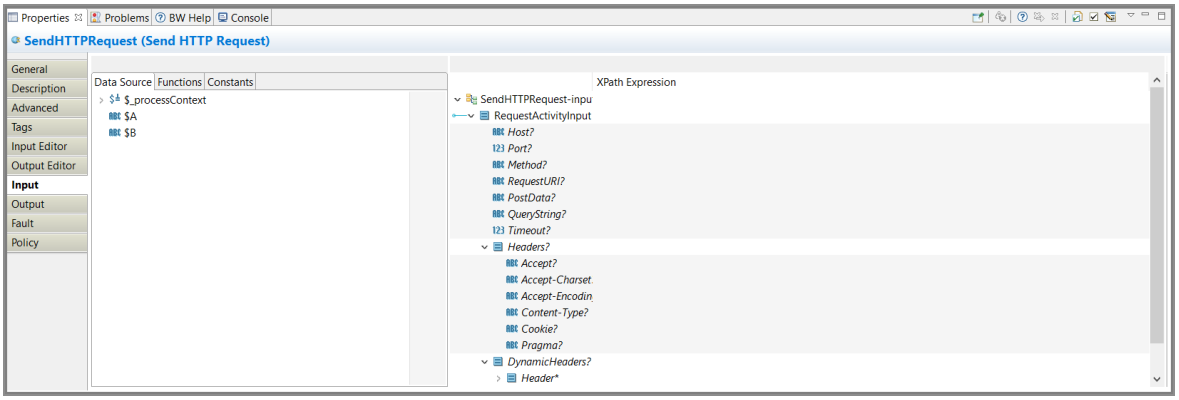
Migrate the ActiveMatrix BusinessWorks 5.x project.

Procedure

1. Click the **Incoming-HTTP-request (HTTP Receiver)** activity.
2. Click the **General** tab.
3. In the **Parameter** table, change the **Parameter Cardinality** from **Required** to **Optional**.

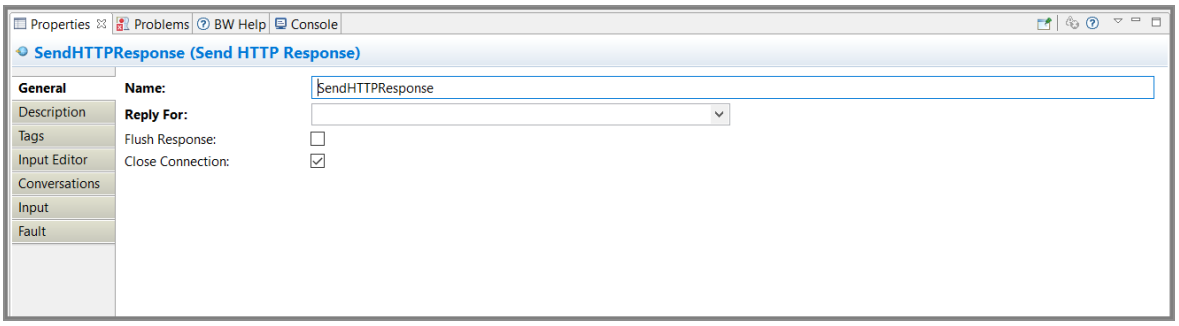


4. Click the **Send-Request-to-Wiki-News (Send HTTP Request)** activity and specify the following three values on the **Input** tab.



Input Type	Values
RequestURI	"/wiki/Main_Page"
Accept-Encoding	"text/*, text/html, text/html;level=1"
Content-Type	"text/html;charset=UTF-8"

5. Click the **Send-Response-from-Wiki-News (Send HTTP Response)** activity and specify the value for **Content-Type** on the **Input** tab.



HTTP Persistent Connection

This topic describes migrating this project from ActiveMatrix BusinessWorks 5.x to ActiveMatrix BusinessWorks 6.x.

Before you begin

Migrate the ActiveMatrix BusinessWorks 5.x project.

Procedure

1. In the **HTTP Client** shared resource, **Client.ClientProcess-Send-HTTP-Request-HttpClientResource**, provide the specified details in the following fields.

Field	Value
Maximum Total Connections	3
Maximum Total Connections Per Host	2
Connection Timeout (ms)	20000

2. Provide the same **HTTP Client** shared resource **Client.ClientProcess-Send-HTTP-Request-HttpClientResource** to the following activities in the process, `ClientProcess.bwp`.
    a. Send-HTTP-Request-1
    b. Send-HTTP-Request-1-1
    c. Send-HTTP-Request-1-2

d. Send-HTTP-Request-1-3

JAVA Method Project

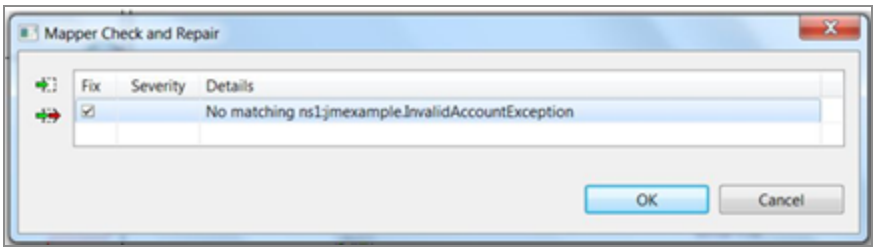
This topic describes migrating this project from ActiveMatrix BusinessWorks 5.x to ActiveMatrix BusinessWorks 6.x.

Before you begin

Migrate the ActiveMatrix BusinessWorks project.

Procedure

1. Go to **PublishBalance.bwp** and perform **Check and Repair** for the **InvalidAcctExp** and **PublishException** activities.
2. In the Mapper Check and Repair wizard, check for jmexample.InvalidAccountException.



3. Run the sample.

JMS Message Selector Project

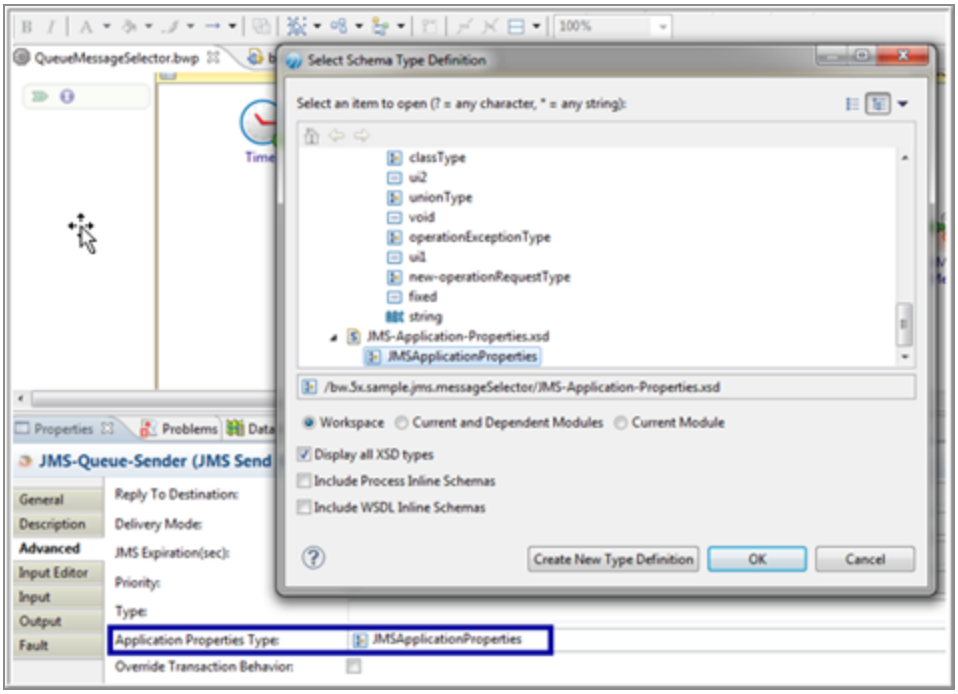
This topic describes migrating this project from ActiveMatrix BusinessWorks 5.x to ActiveMatrix BusinessWorks 6.x.

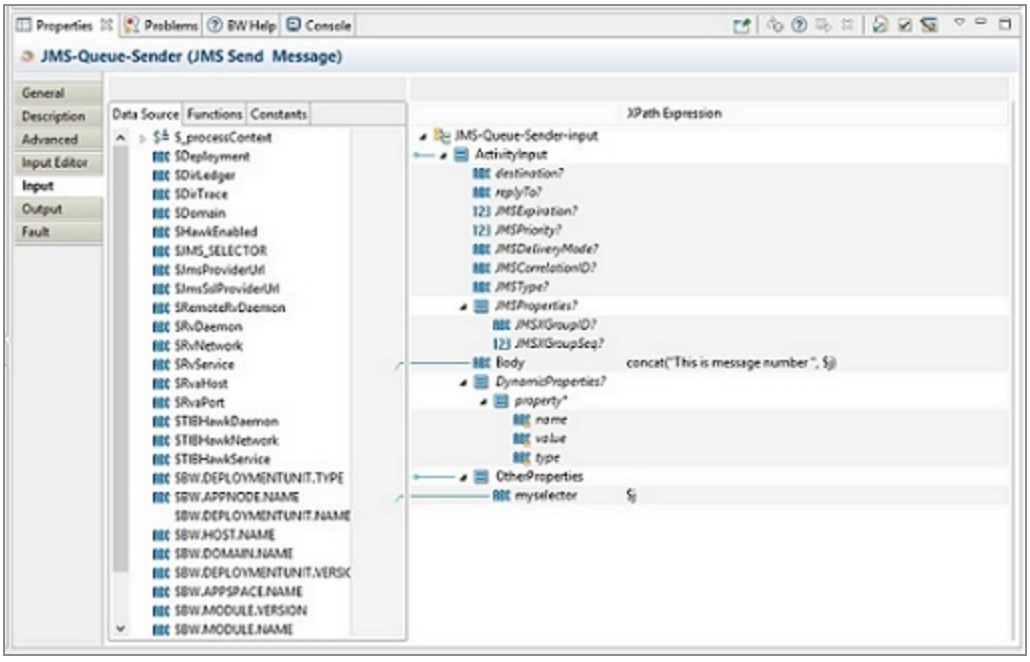
Before you begin

Migrate the ActiveMatrix BusinessWorks 5.x project.

Procedure

1. Open **QueueMessageSelector.bwp**, click the **JMS-Queue-Sender** activity and then click the **Advanced** tab.
2. Click **Application Properties Type**, open the Select Schema Type Definition wizard and select **JMSApplicationProperties**.





3. Run the process.

HTML Mail with Image

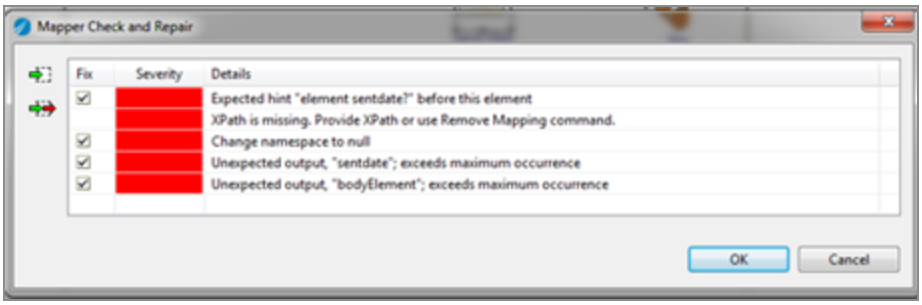
Migrating this project from ActiveMatrix BusinessWorks 5.x to ActiveMatrix BusinessWorks 6.x involves certain manual steps to be performed. This topic provides information about the manual steps involved for a successful migration.

Before you begin

Migrate the ActiveMatrix BusinessWorks 5.x project.

Procedure

1. Go to **Sender.bwp** and click **Send-Mail** activity.
2. Do **Check and Repair**.



Mail with Simple Attachment

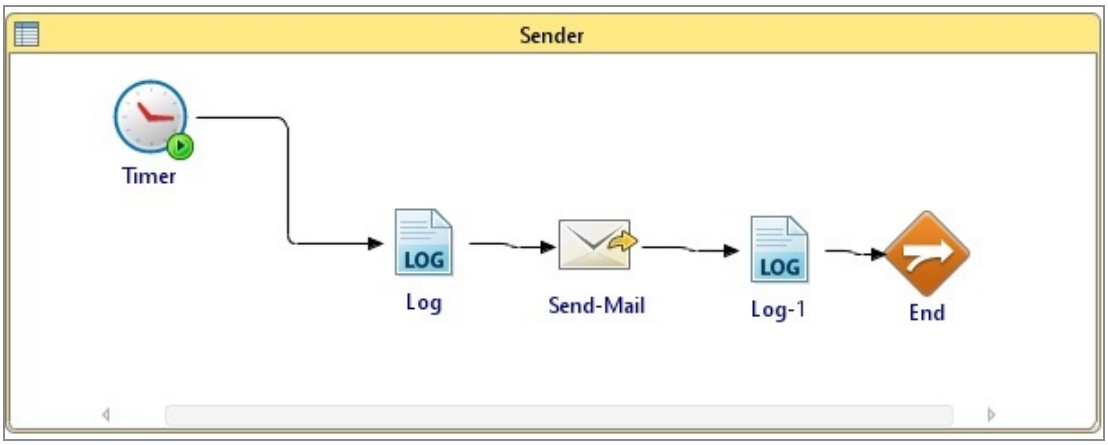
This topic describes migrating this project from ActiveMatrix BusinessWorks 5.x to ActiveMatrix BusinessWorks 6.x.

Before you begin

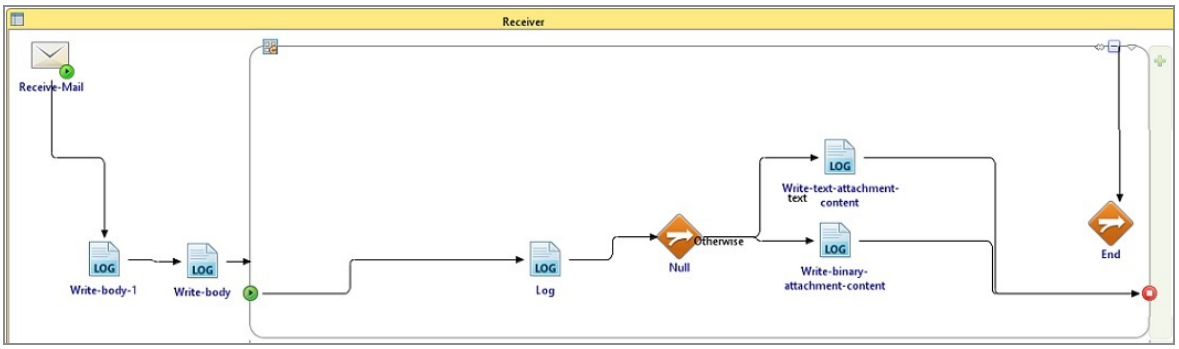
Migrate the ActiveMatrix BusinessWorks 5.x project.

Procedure

1. After migrating the project, select the **Sender.bwp** process.



2. Select the **Receiver.bwp** process.



TIBCO Documentation and Support Services

For information about this product, you can read the documentation, contact TIBCO Support, and join TIBCO Community.

How to Access TIBCO Documentation

Documentation for TIBCO products is available on the Product Documentation website, mainly in HTML and PDF formats.

The Product Documentation website is updated frequently and is more current than any other documentation included with the product.

Product-Specific Documentation

The following documentation for this product is available on the TIBCO ActiveMatrix BusinessWorks™ page:

- *TIBCO ActiveMatrix BusinessWorks™ Release Notes*
- *TIBCO ActiveMatrix BusinessWorks™ Installation*
- *TIBCO ActiveMatrix BusinessWorks™ Application Development*
- *TIBCO ActiveMatrix BusinessWorks™ Bindings and Palettes Reference*
- *TIBCO ActiveMatrix BusinessWorks™ Concepts*
- *TIBCO ActiveMatrix BusinessWorks™ Error Codes*
- *TIBCO ActiveMatrix BusinessWorks™ Getting Started*
- *TIBCO ActiveMatrix BusinessWorks™ Migration*
- *TIBCO ActiveMatrix BusinessWorks™ Performance Benchmarking and Tuning*
- *TIBCO ActiveMatrix BusinessWorks™ REST Implementation*
- *TIBCO ActiveMatrix BusinessWorks™ Refactoring Best Practices*
- *TIBCO ActiveMatrix BusinessWorks™ Samples*

How to Contact Support for TIBCO Products

You can contact the Support team in the following ways:

- To access the Support Knowledge Base and getting personalized content about products you are interested in, visit our product Support website.
- To create a Support case, you must have a valid maintenance or support contract with a Cloud Software Group entity. You also need a username and password to log in to the product Support website. If you do not have a username, you can request one by clicking **Register** on the website.

How to Join TIBCO Community

TIBCO Community is the official channel for TIBCO customers, partners, and employee subject matter experts to share and access their collective experience. TIBCO Community offers access to Q&A forums, product wikis, and best practices. It also offers access to extensions, adapters, solution accelerators, and tools that extend and enable customers to gain full value from TIBCO products. In addition, users can submit and vote on feature requests from within the TIBCO Ideas Portal. For a free registration, go to TIBCO Community.

Legal and Third-Party Notices

SOME CLOUD SOFTWARE GROUP, INC. (“CLOUD SG”) SOFTWARE AND CLOUD SERVICES EMBED, BUNDLE, OR OTHERWISE INCLUDE OTHER SOFTWARE, INCLUDING OTHER CLOUD SG SOFTWARE (COLLECTIVELY, “INCLUDED SOFTWARE”). USE OF INCLUDED SOFTWARE IS SOLELY TO ENABLE THE FUNCTIONALITY (OR PROVIDE LIMITED ADD-ON FUNCTIONALITY) OF THE LICENSED CLOUD SG SOFTWARE AND/OR CLOUD SERVICES. THE INCLUDED SOFTWARE IS NOT LICENSED TO BE USED OR ACCESSED BY ANY OTHER CLOUD SG SOFTWARE AND/OR CLOUD SERVICES OR FOR ANY OTHER PURPOSE.

USE OF CLOUD SG SOFTWARE AND CLOUD SERVICES IS SUBJECT TO THE TERMS AND CONDITIONS OF AN AGREEMENT FOUND IN EITHER A SEPARATELY EXECUTED AGREEMENT, OR, IF THERE IS NO SUCH SEPARATE AGREEMENT, THE CLICKWRAP END USER AGREEMENT WHICH IS DISPLAYED WHEN ACCESSING, DOWNLOADING, OR INSTALLING THE SOFTWARE OR CLOUD SERVICES (AND WHICH IS DUPLICATED IN THE LICENSE FILE) OR IF THERE IS NO SUCH LICENSE AGREEMENT OR CLICKWRAP END USER AGREEMENT, THE LICENSE(S) LOCATED IN THE “LICENSE” FILE(S) OF THE SOFTWARE. USE OF THIS DOCUMENT IS SUBJECT TO THOSE SAME TERMS AND CONDITIONS, AND YOUR USE HEREOF SHALL CONSTITUTE ACCEPTANCE OF AND AN AGREEMENT TO BE BOUND BY THE SAME.

This document is subject to U.S. and international copyright laws and treaties. No part of this document may be reproduced in any form without the written authorization of Cloud Software Group, Inc.

TIBCO, the TIBCO logo, the TIBCO O logo, ActiveMatrix BusinessWorks, ActiveSpaces, Business Studio, TIBCO Business Studio, TIBCO Designer, TIBCO Enterprise Administrator, Enterprise Message Service, Rendezvous, and TIBCO Runtime Agent are either registered trademarks or trademarks of Cloud Software Group, Inc. in the United States and/or other countries.

Java and all Java based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

All other product and company names and marks mentioned in this document are the property of their respective owners and are mentioned for identification purposes only. You acknowledge that all rights to these third party marks are the exclusive property of their respective owners. Please refer to Cloud SG’s Third Party Trademark Notices (https://www.cloud.com/legal) for more information.

This document includes fonts that are licensed under the SIL Open Font License, Version 1.1, which is available at: https://scripts.sil.org/OFL

Copyright (c) Paul D. Hunt, with Reserved Font Name Source Sans Pro and Source Code Pro.

Cloud SG software may be available on multiple operating systems. However, not all operating system platforms for a specific software version are released at the same time. See the “readme” file for the availability of a specific version of Cloud SG software on a specific operating system platform.

THIS DOCUMENT IS PROVIDED “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT.

THIS DOCUMENT COULD INCLUDE TECHNICAL INACCURACIES OR TYPOGRAPHICAL ERRORS. CHANGES ARE PERIODICALLY ADDED TO THE INFORMATION HEREIN; THESE CHANGES WILL BE INCORPORATED IN NEW EDITIONS OF THIS DOCUMENT. CLOUD SG MAY MAKE IMPROVEMENTS AND/OR CHANGES IN THE PRODUCT(S), THE PROGRAM(S), AND/OR THE SERVICES DESCRIBED IN THIS DOCUMENT AT ANY TIME WITHOUT NOTICE.

THE CONTENTS OF THIS DOCUMENT MAY BE MODIFIED AND/OR QUALIFIED, DIRECTLY OR INDIRECTLY, BY OTHER DOCUMENTATION WHICH ACCOMPANIES THIS SOFTWARE, INCLUDING BUT NOT LIMITED TO ANY RELEASE NOTES AND "README" FILES.

This and other products of Cloud SG may be covered by registered patents. For details, please refer to the Virtual Patent Marking document located at https://www.tibco.com/patents.

Copyright © 2015-2023. Cloud Software Group, Inc. All Rights Reserved.