# TIBCO BusinessWorks™ Container Edition

## Maven Plug-in

Version 2.9.2 | August 2024

# Contents

# Installing Maven Plug-in

You can install the Maven plug-in using the GUI, Console, or Silent mode.

### Before you begin

If you want to install TIBCO ActiveMatrix BusinessWorks™ Maven Plug-in, install Apache Maven from https://maven.apache.org/download.cgi and set MAVEN_HOME in the Environment Variables.

For more information on how to install maven plug-in, see the "Installation Modes and Procedures" in the ActiveMatrix BusinessWorks *Installation*.

# Post Installation tasks

Post installation tasks are additional tasks that you might have to perform after installation.

### Before you begin

Complete the installation before running the post installation tasks.

> **Note:** In case Apache Maven is not installed before installing TIBCO BusinessWorks Container Edition Maven Plug-in, install Apache Maven and then run `install.sh/install.bat` from `<TIBCO-HOME>\bwce\2.x\maven` to install Maven Plug-in.

## Maven Plug-in versions compatible with TIBCO BusinessWorks Container Edition

The following table shows the maven plug-in versions compatible with TIBCO BusinessWorks Container Edition.

| TIBCO BusinessWorks Container Edition Version | Maven Plug-in Version |
|---|---|
| 2.7.0 | 2.9.0 |
| 2.7.1 | 2.9.1 |
| 2.7.2 | 2.9.2 |
| 2.7.3 | 2.9.3 |
| 2.8.0 | 2.9.4 |

| TIBCO BusinessWorks Container Edition Version | Maven Plug-in Version |
|---|---|
| 2.8.1 | 2.9.5 |
| 2.8.2 | 2.9.6 |
| 2.8.3 | 2.9.7 |
| 2.9.0 | 2.9.8 |

# Unit Testing

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

**Note:** To get familiarized with the Unit Testing Samples, see "Unit Testing" in the TIBCO BusinessWorks™ Container Edition Samples.

# Running Test Assertions

Unit tests focus on testing small units of work, which in TIBCO BusinessWorks Container Edition maps to individual processes or subprocesses. Ideally this is done in a standalone manner, with no touchpoints or dependencies on other components or interfaces. This is distinct from interface or system testing that would test the service or operation as a whole. Interface tests are run using other tools such as SOAP UI.

# Adding Unit Test Assertions

To add unit test assertions in TIBCO Business Studio for BusinessWorks, follow these steps:

### Before you begin

- The `UnitTestDemo.zip` file must be present in an accessible location.

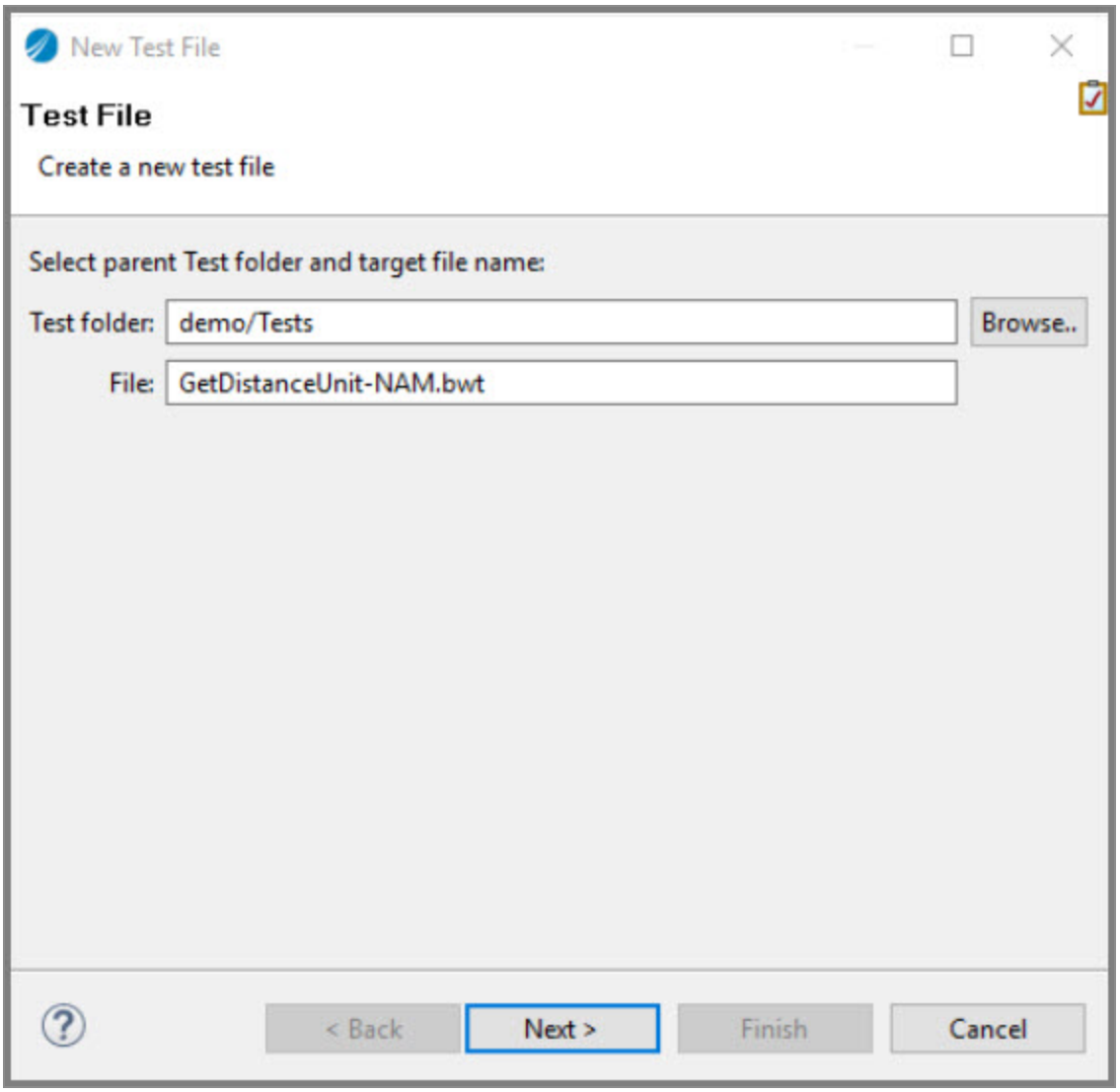
### Procedure

1. In TIBCO Business Studio for BusinessWorks, on the demo project, right-click the Tests folder and select **New > Add Test File**.

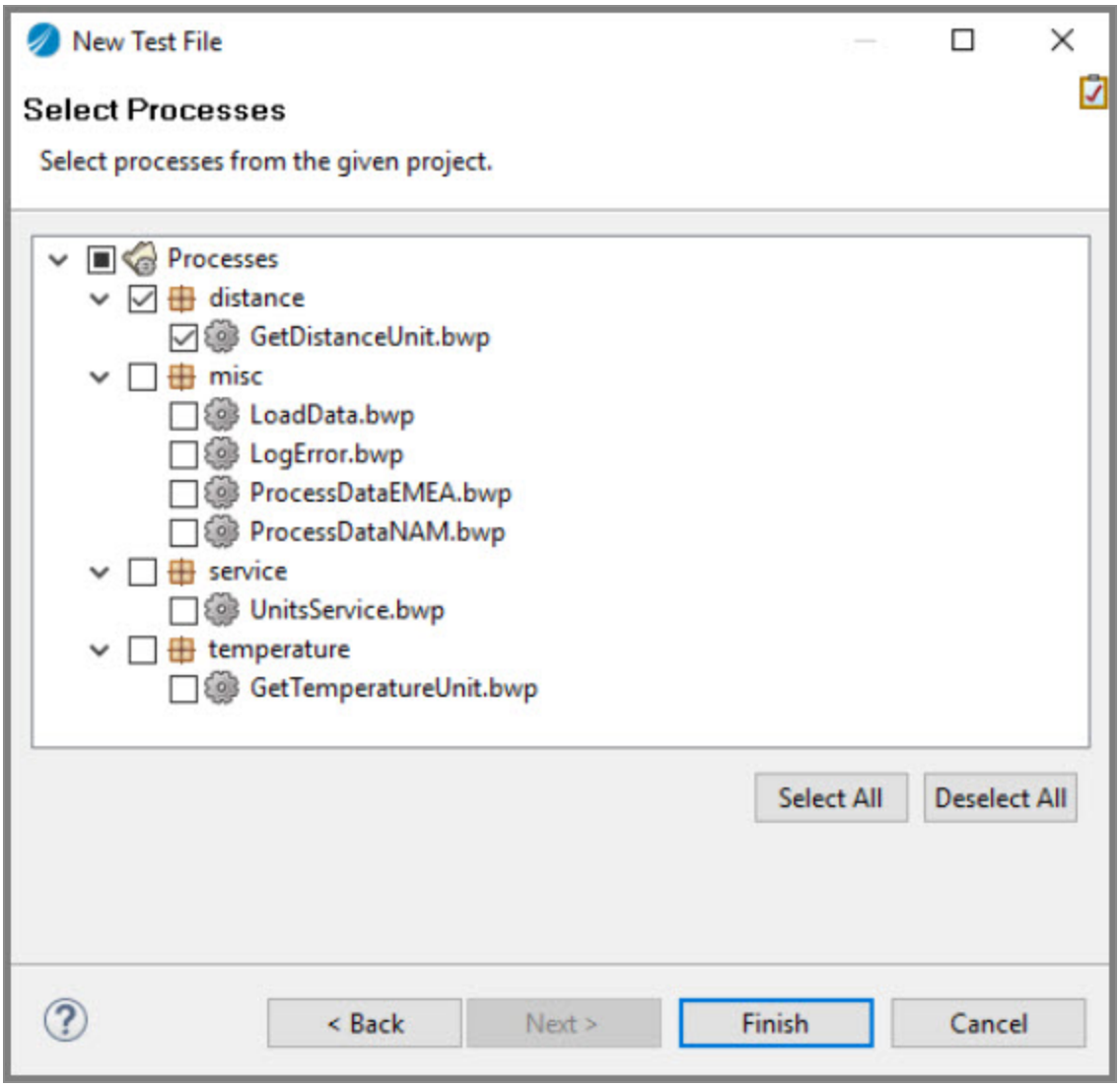
The **New Test File** wizard displays with the **Test File** page.

> **Note:** You can also create a Test file in the subfolder created under the Tests folder.

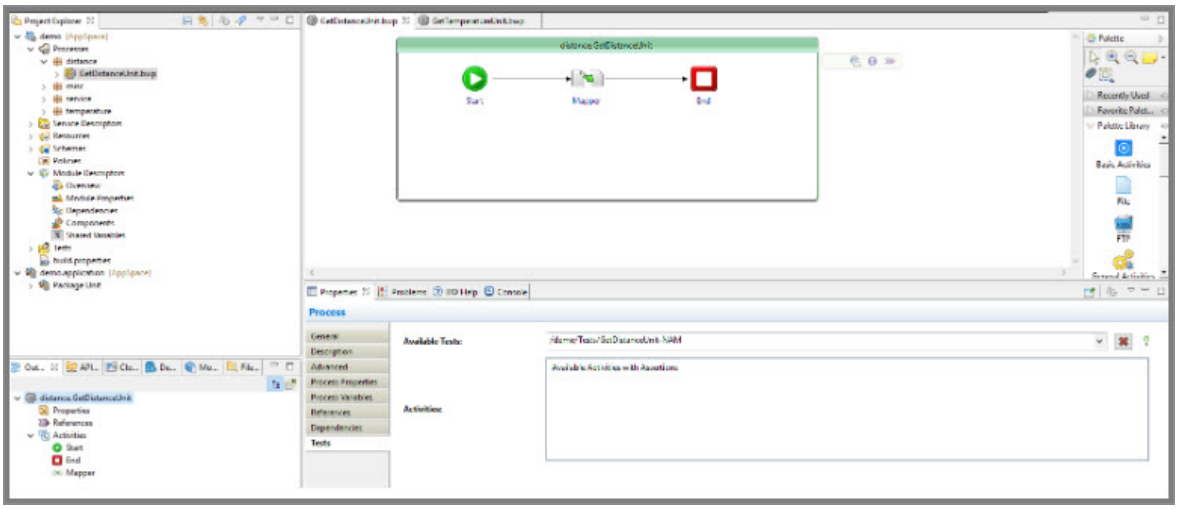
2. In the **New Test File** wizard, change the file name to `GetDistanceUnit-NAM.bwt` and keep the Tests folder as default. Click **Next**.




3. Add the `GetDistanceUnit.bwp` to the process and click **Finish**.



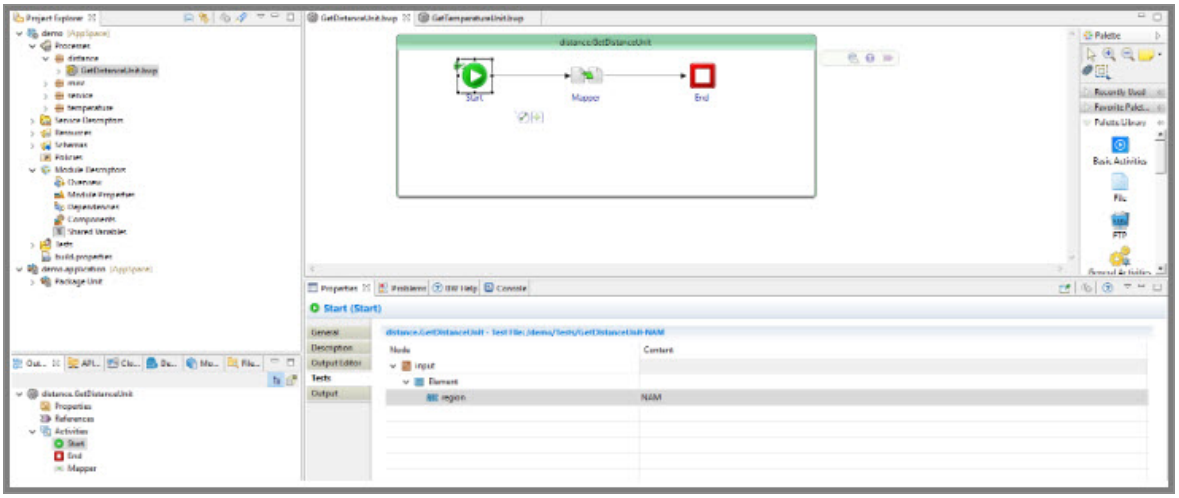

4. In the Project Explorer, open `GetDistanceUnit.bwp` and click the `distance.GetDistanceUnit` process (green box) and select the **Properties** tab. Since this process is added to the Tests file, the **Tests** tab appears on the **Process** panel. Click the Tests tab and the created file is selected in the **Available Tests** dropdown.



**Note:** Clicking the red cross-mark **Delete selected bwt file** deletes the test file permanently.

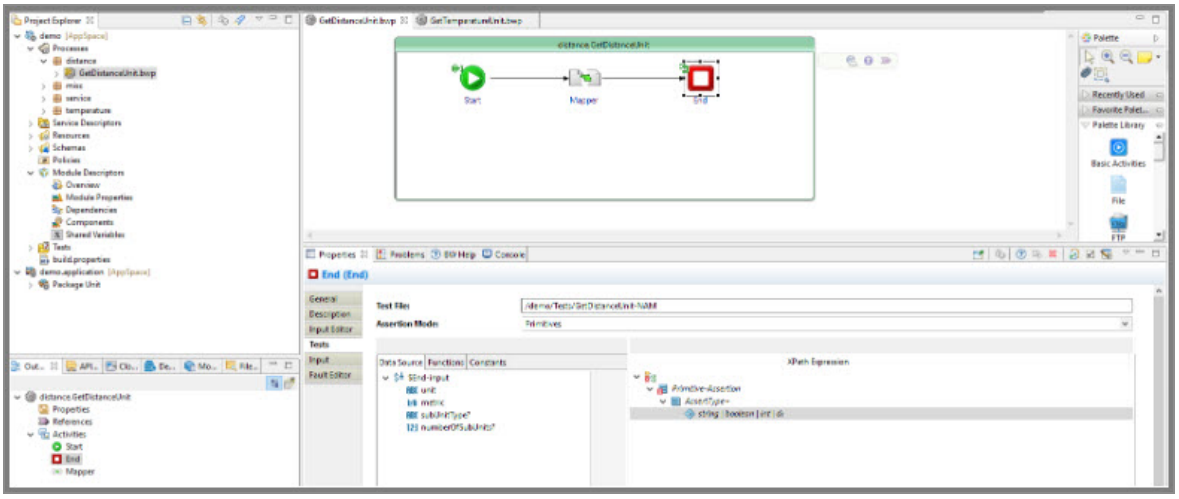
5. Right-click the **Start** activity and select **Add Test > Add Input**. Click the **Tests** tab under **Properties** and add NAM in the **Content** column for the **region** field.

**Note:** NAM should not contain any double quotes ("").

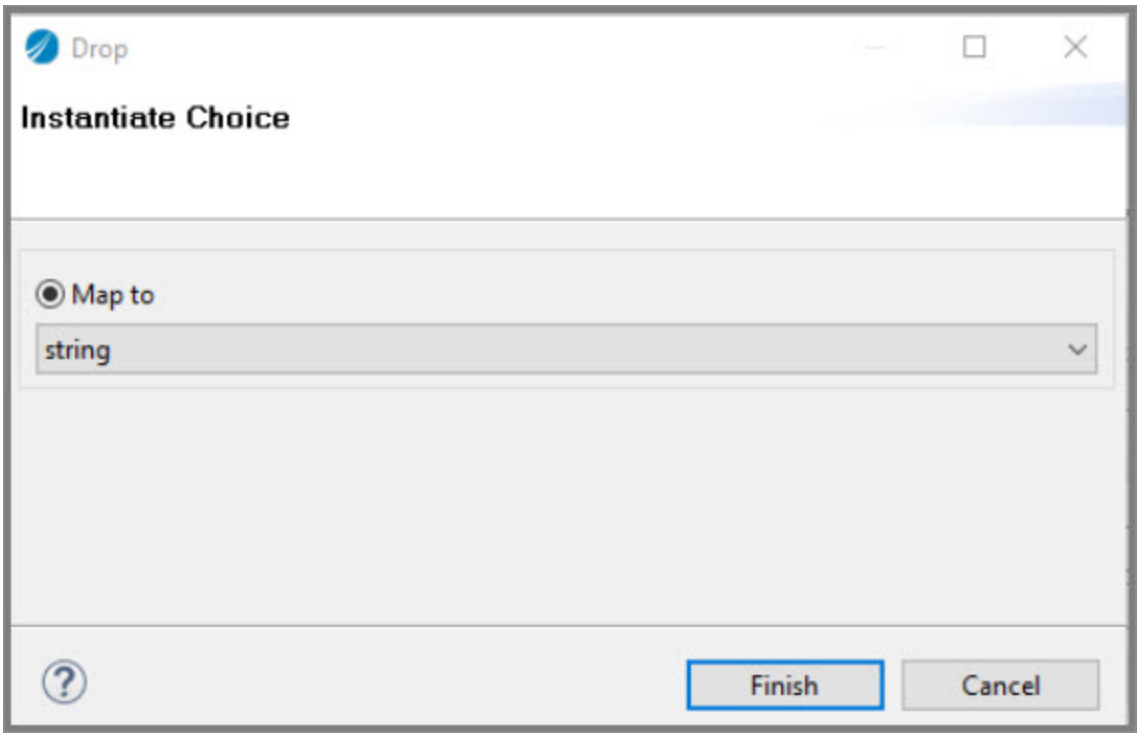


> **Note:** The process does not need to be saved after adding the test inputs and assertions.

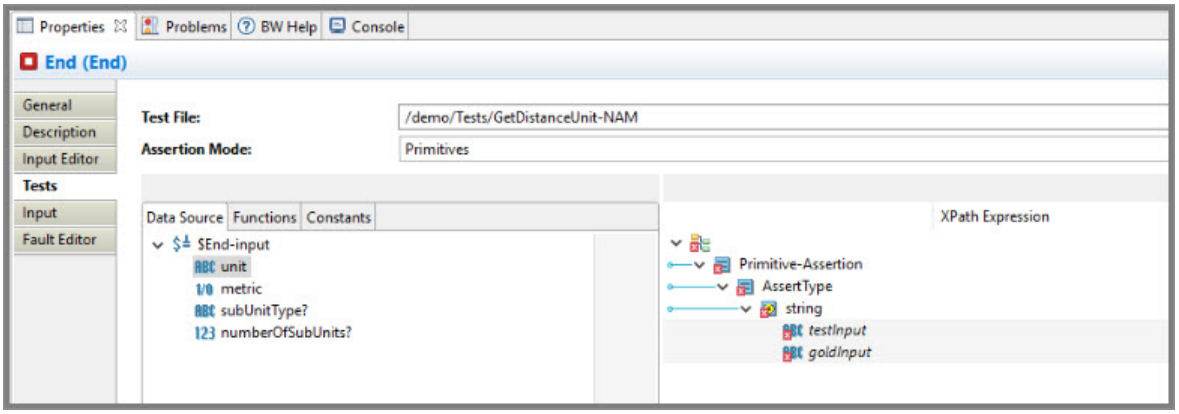
6. Right-click the **End** activity and select **Add Test > Add Input**. Click the **Tests** tab under **Properties** and expand AssertType+ and $End-input, which is both the sides of the mapper.



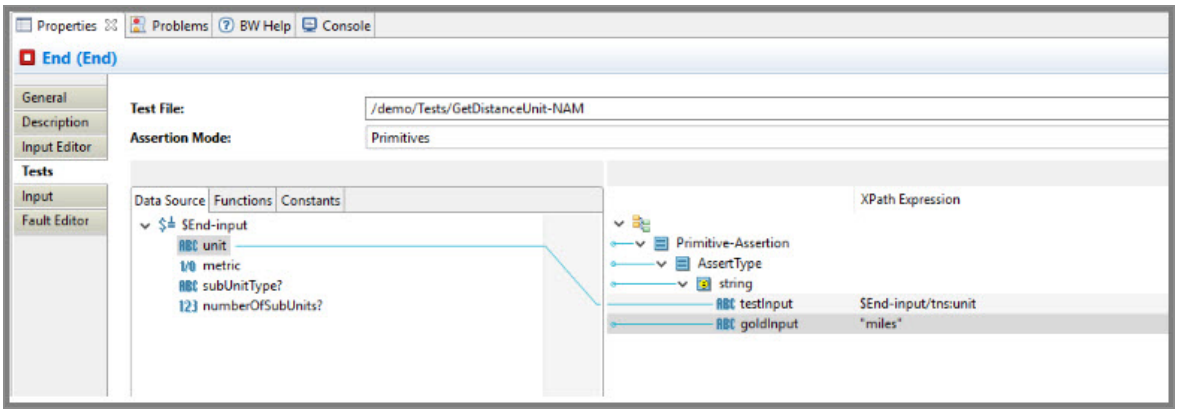
7. Drag the string|boolean|… element from the right-hand side to any element on the left-hand side of the mapper underneath $End-input. The **Drop** wizard opens to select a data type. Select the "String" data type and click **Finish**.



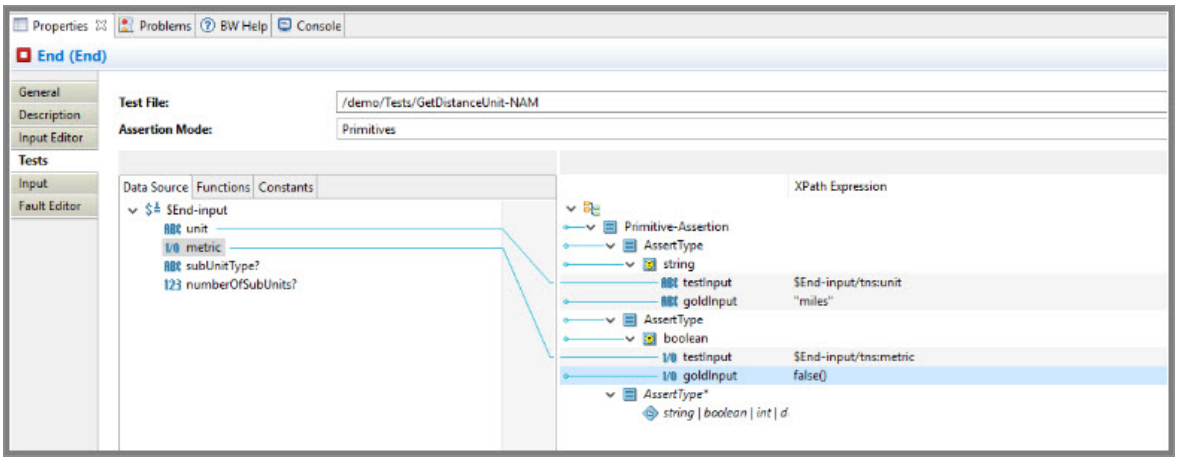

The **testInput** and **goldInput** fields are displayed.



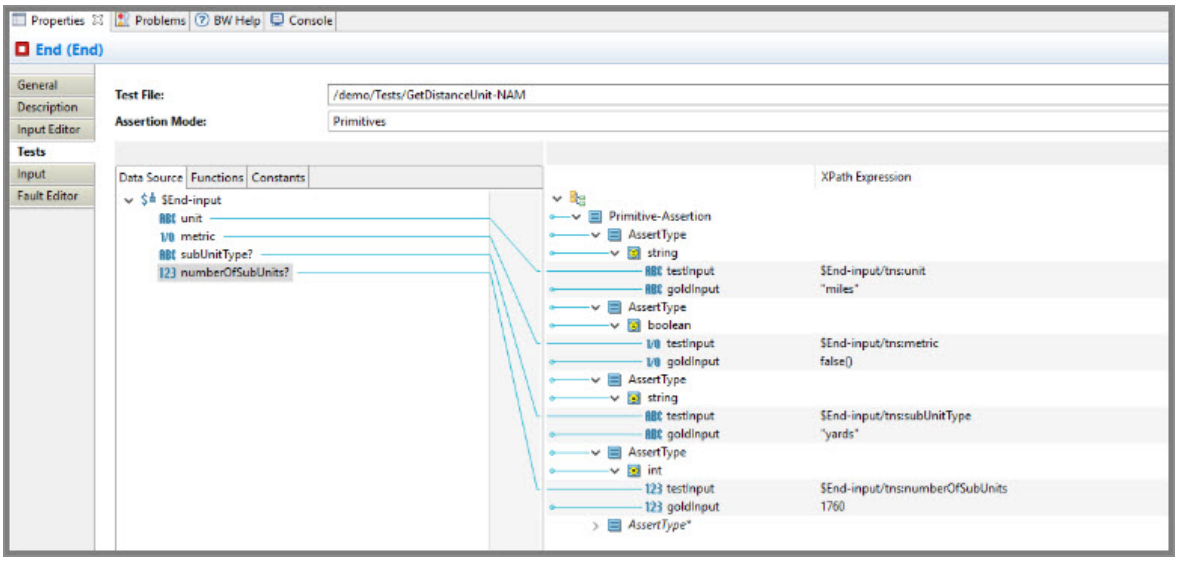

8. In the **Data Source** tab, drag "unit" to the **testInput** field. This is the value that you are evaluating in the assertion. Add miles as an input to the **goldInput** field.



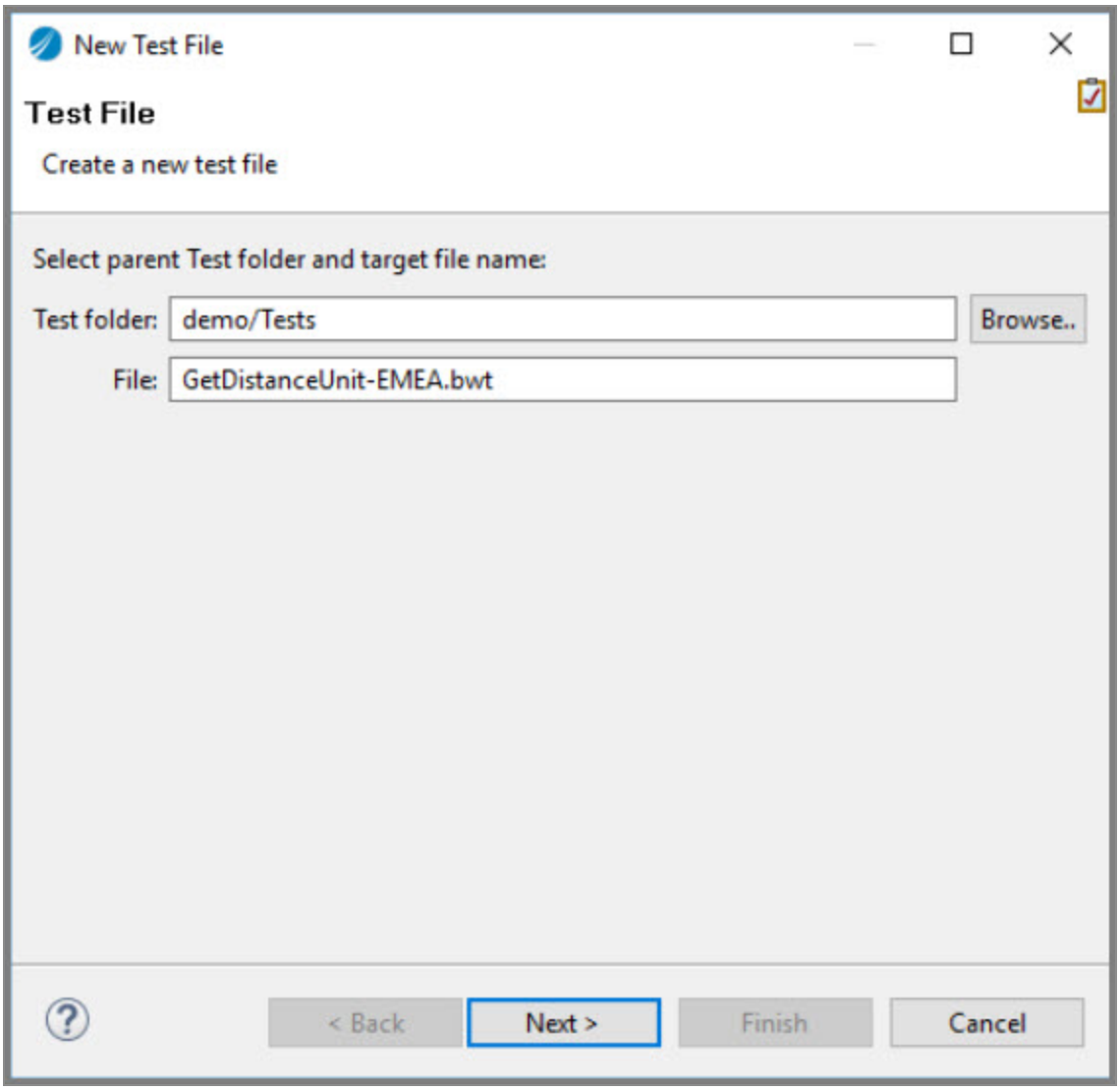

9. Right-click the AssertType and choose Duplicate. Right-click on Primitive-Assertion and choose Expand All. Under the second AssertType element, right-click the AssertType and choose Remove Mapping. Drag the string | boolean… element from the right-hand side to any element under **$End-input** on the left and choose the "boolean" data type. Drag the "metric" element from the left onto the **testInput** field under Boolean and enter false() in the **goldInput** field.



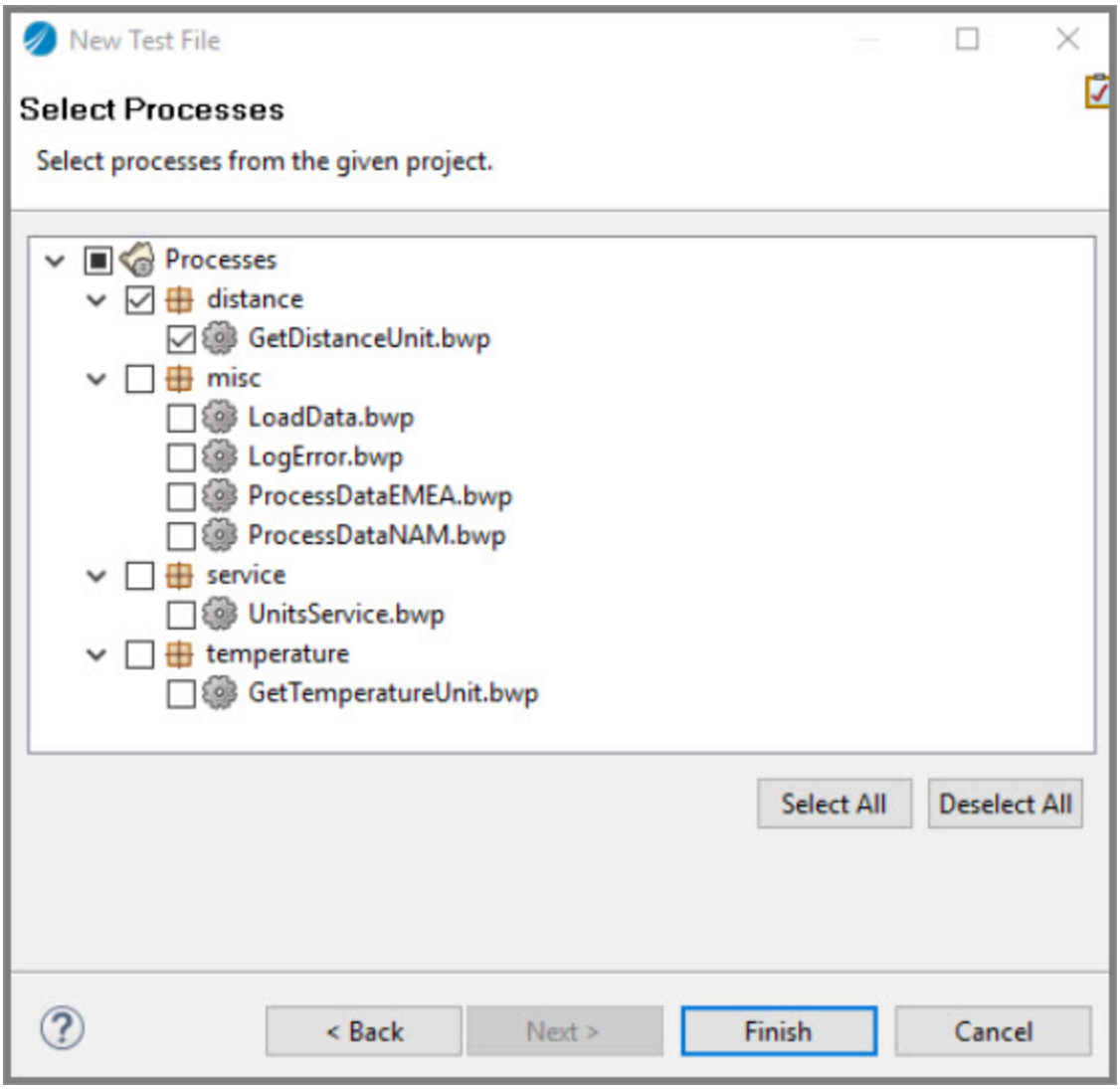

10. In a similar way as above, complete the mappings so that you also assert "subUnitType" and "numberOfSubUnits"



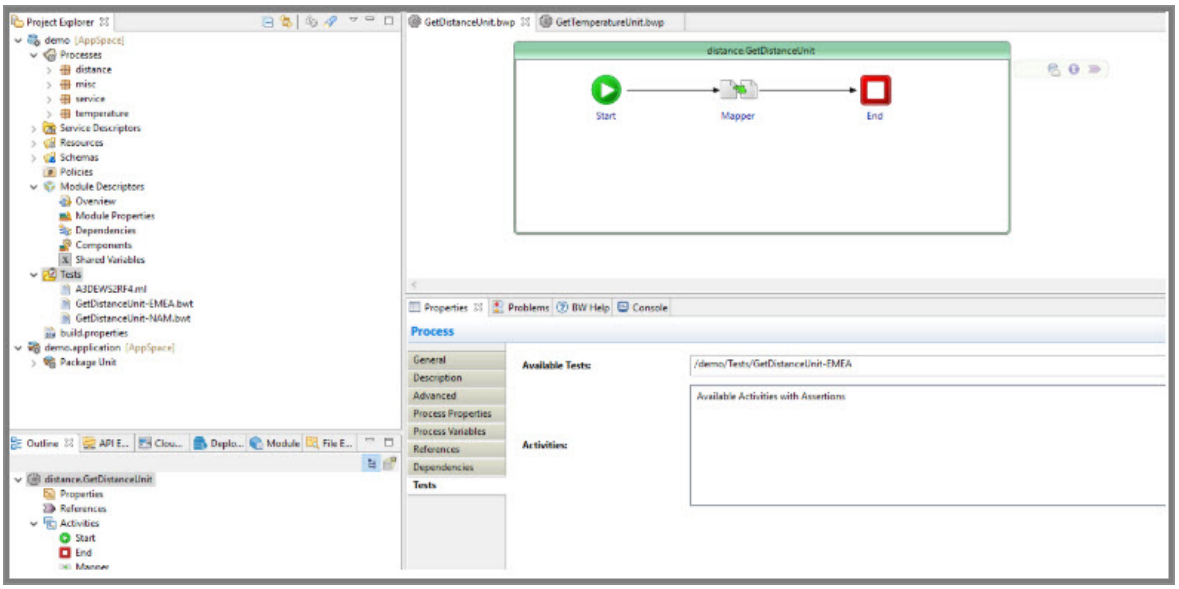

11. To add a new test file, right-click the Tests folder and select **New > Add Test File**. In the **File** field, add the name of the file as `GetDistanceUnit-EMEA.bwt` and click **Next**.



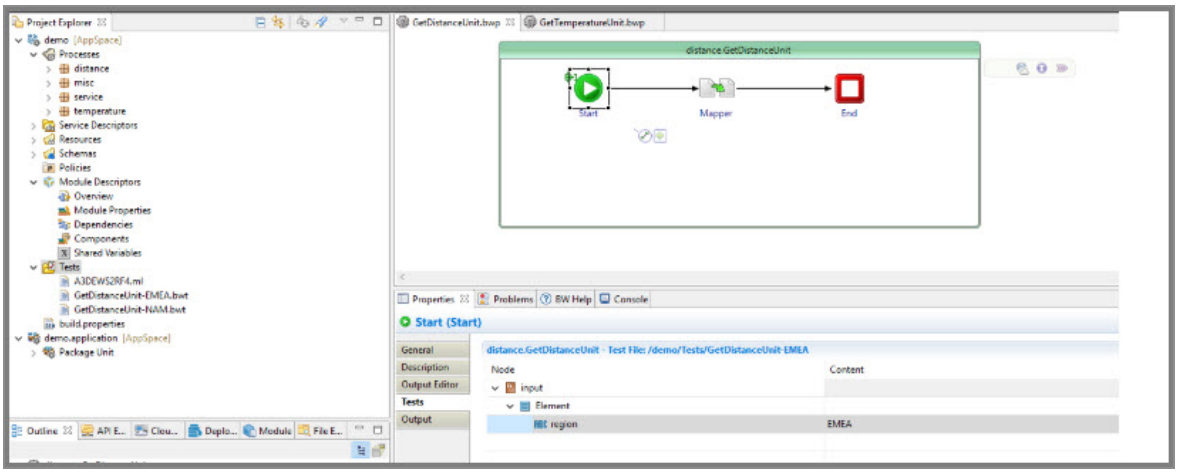

12. Select `GetDistanceUnit.bwp` and click **Finish**.



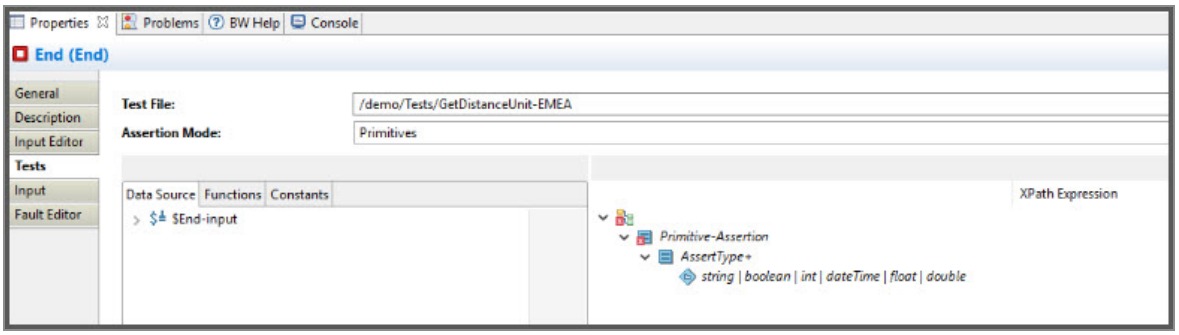

13. In the Project Explorer, open `GetDistanceUnit.bwp` and click the `distance.GetDistanceUnit` process (green box) and select the **Properties** tab. Since this process is added to the Tests file, the **Tests** tab appears on the **Process** panel. Click the Tests tab and the `demo/Tests/GetDistanceUnit-EMEA` test file is selected in the **Available Tests** dropdown. If not, select it manually.




14. Right-click the **Start** activity and select **Add Test > Add Input**. Click the **Tests** tab under **Properties** and add EMEA in the **Content** column for the **region** field.

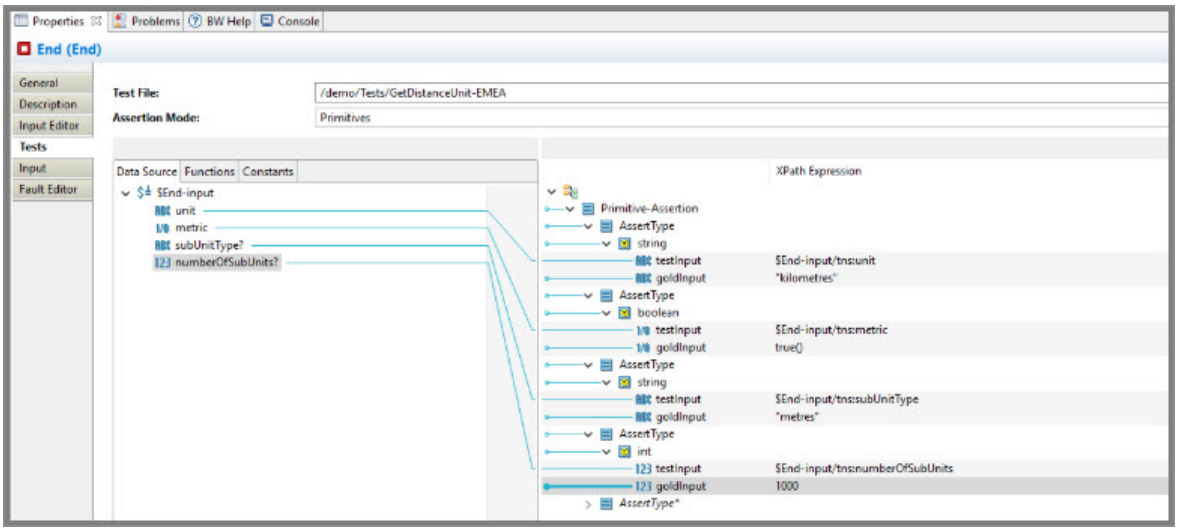



15. Right-click the **End** activity and select **Add Test > Add Input**. Click the **Tests** tab under **Properties** and expand AssertType+ and $End-input, which is both the sides of the mapper.




16. Repeat steps 7, 8, 9, and 10 to set the assertions for GetDistanceUnit-EMEA with "unit", "metric", "subUnitType", and "numberofSubUnits".

    The output looks as follows:




    To run Unit tests in TIBCO Business Studio for BusinessWorks, see Running Unit Tests in Studio.

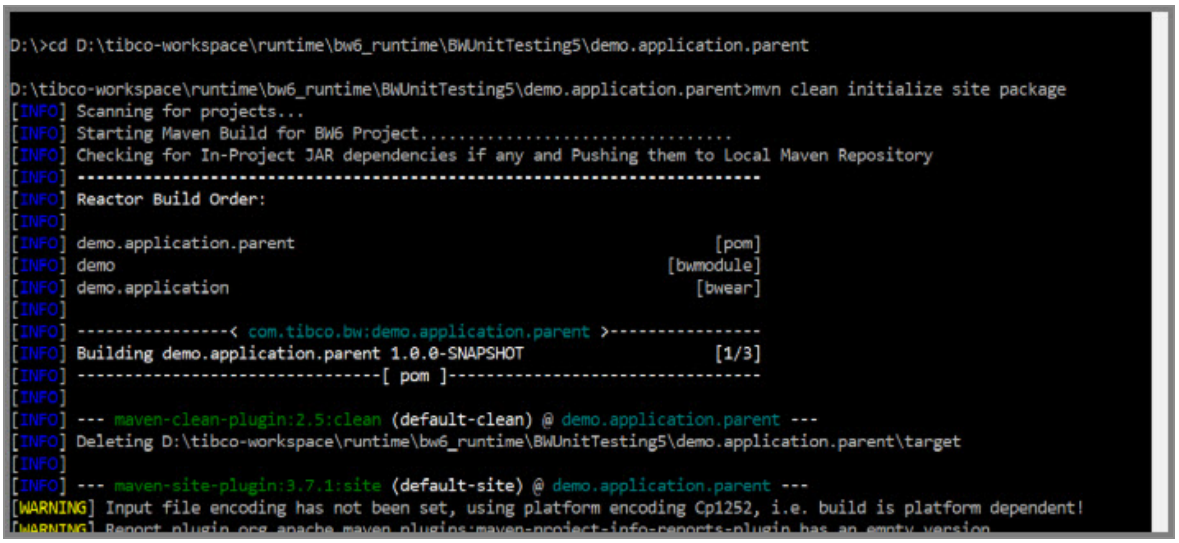
# Running Maven from Command Line

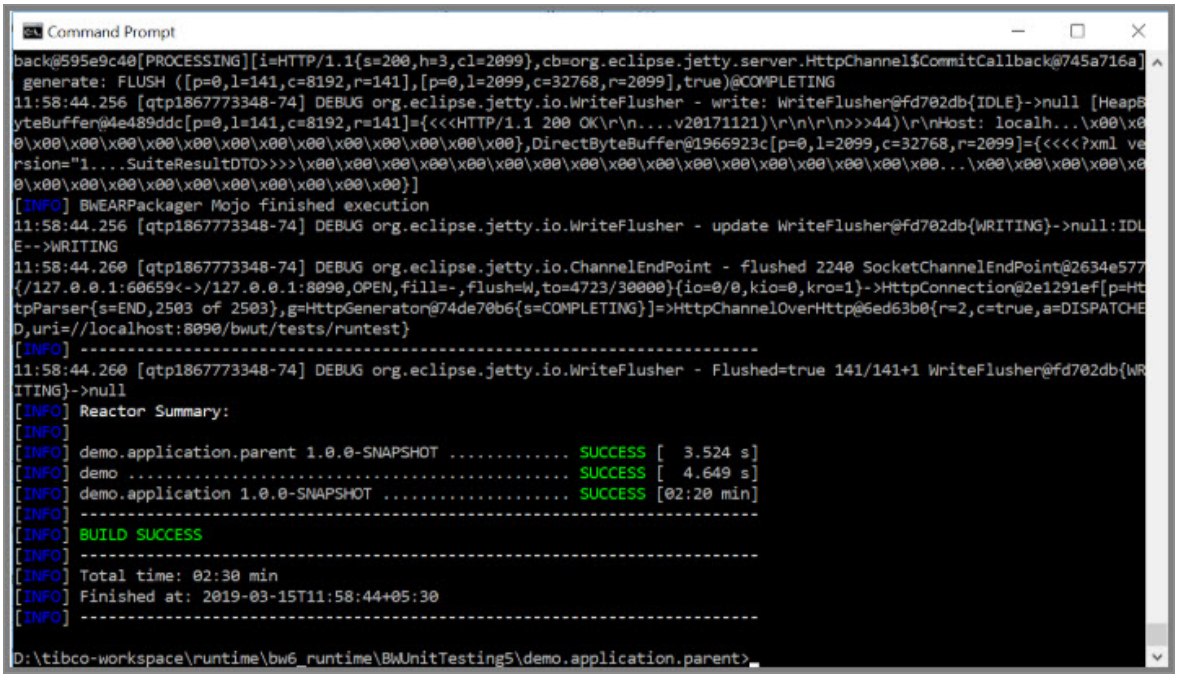
To run Maven plug-in from command line, perform the following steps:

### Procedure

1. Open your command prompt and navigate to the location where your demo project is present.
2. Run the command `clean initialize site package` on your command prompt terminal.

This produces the same result as running Debug within TIBCO Business Studio for BusinessWorks.




# Unit Test Reports and Test Coverage Reports

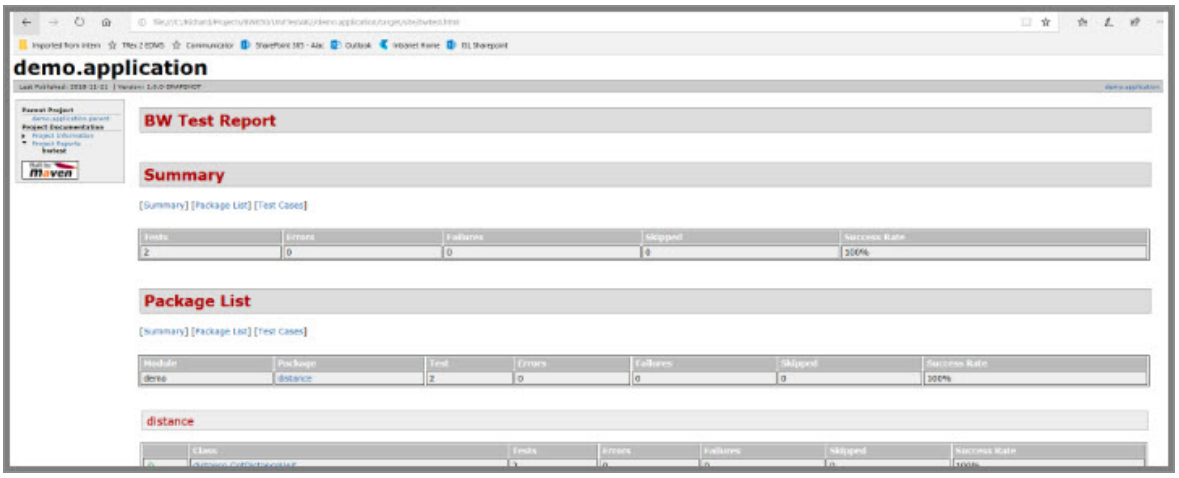
The "site" goal that is included in the Maven debug configuration in TIBCO Business Studio for BusinessWorks and on the command line produces unit test reports and test coverage

reports. These test reports are located at `\demo.application\target\site`.

### Procedure

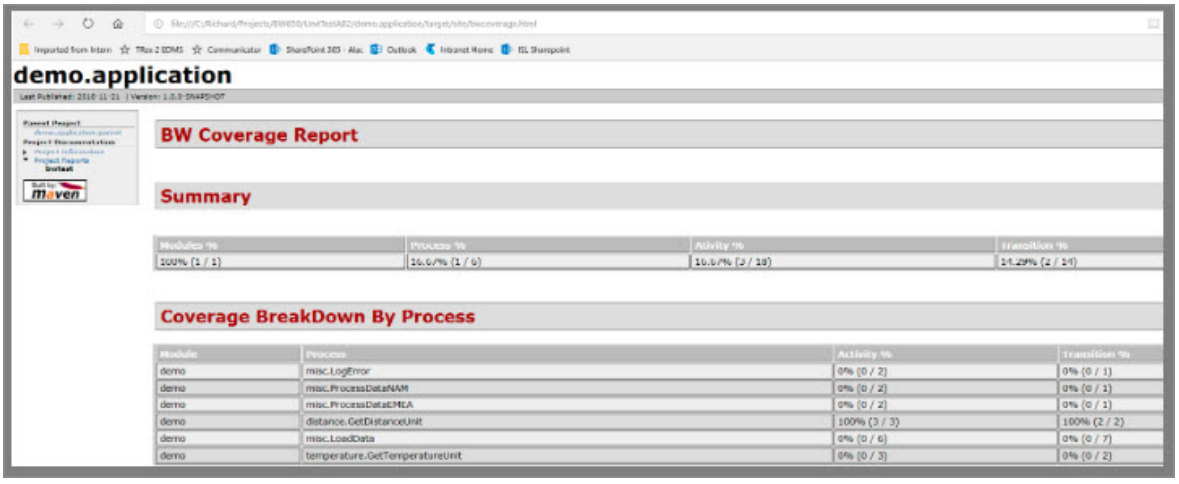
1. Open the `index.html` using the browser.
2. Select **Project Reports > bwtest**.

   This shows a summary of the tests that were run and whether they passed or failed.




3. From the same folder, open `bwcoverage.html`.

   This shows a summary of which processes and activities are covered by unit tests, for the entire project and as a breakdown for each process.



# Limitations for Unit Test Assertions

The following are the limitations for the Unit Test Assertions:

- TIBCO BusinessWorks Container Edition must be installed on the same server where the tests are to run.
- Unit Tests can currently only be invoked with Maven.

# Running Activity Assertions

To run activity assertions in TIBCO BusinessWorks Container Edition, follow these steps:

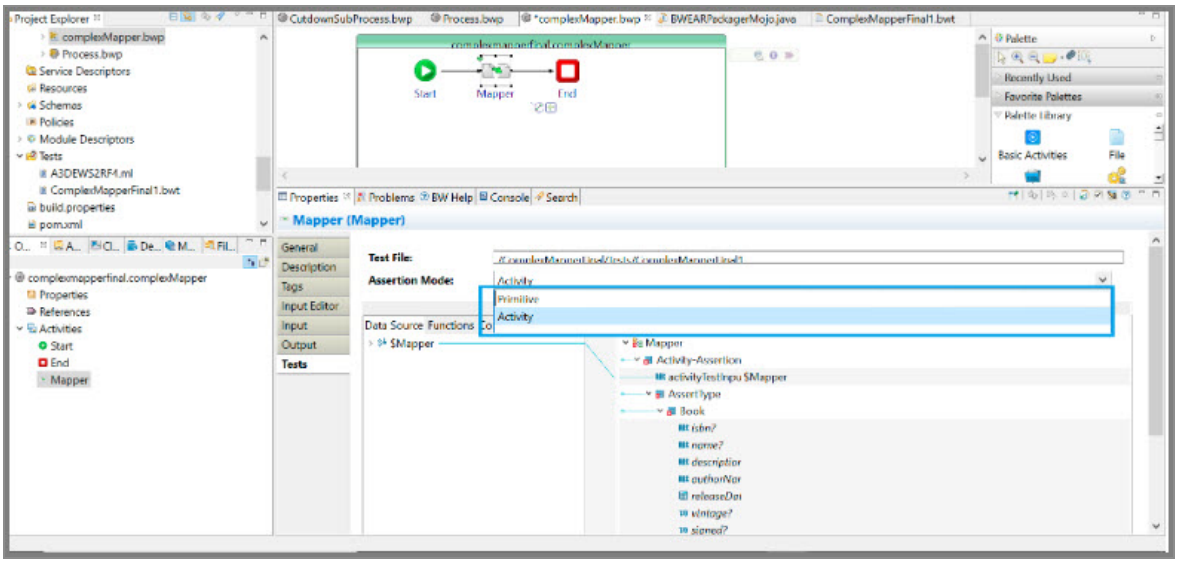
**Procedure**

1. Right-click on the activity from the process or subprocess and select **Add Test > Add Assertion**.

   It adds the **Test** tab to the activity.

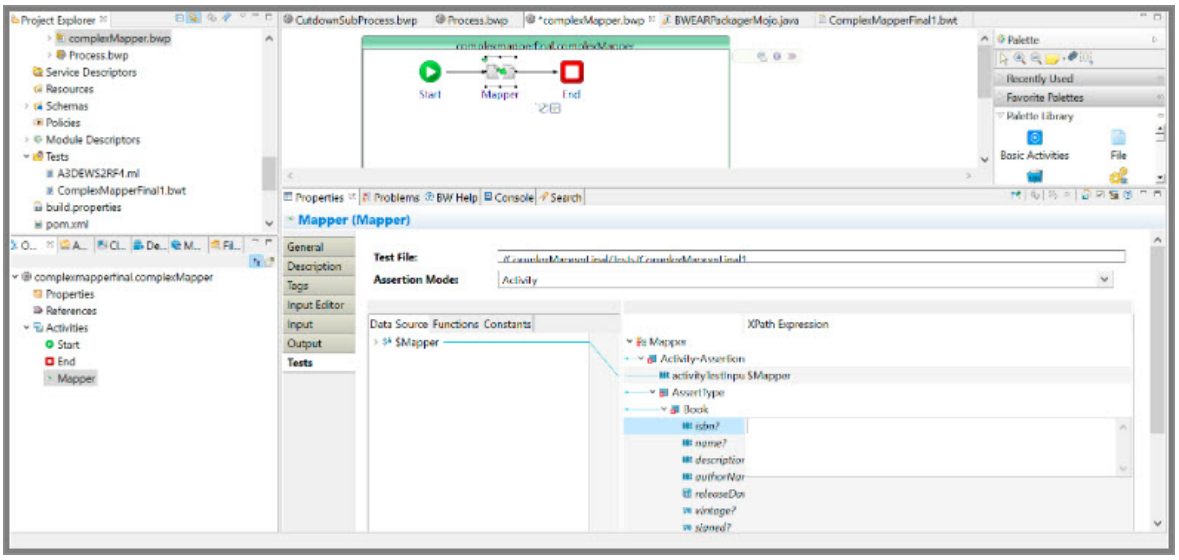
2. On the **Tests** tab, navigate to the **Assertion Mode** dropdown.
   **The Assertion Mode dropdown has two modes:**

   - **Primitive: In this mode, only the primitive types of elements are tested.**
   - **Activity: In this mode, the complete activity outputs are tested.**

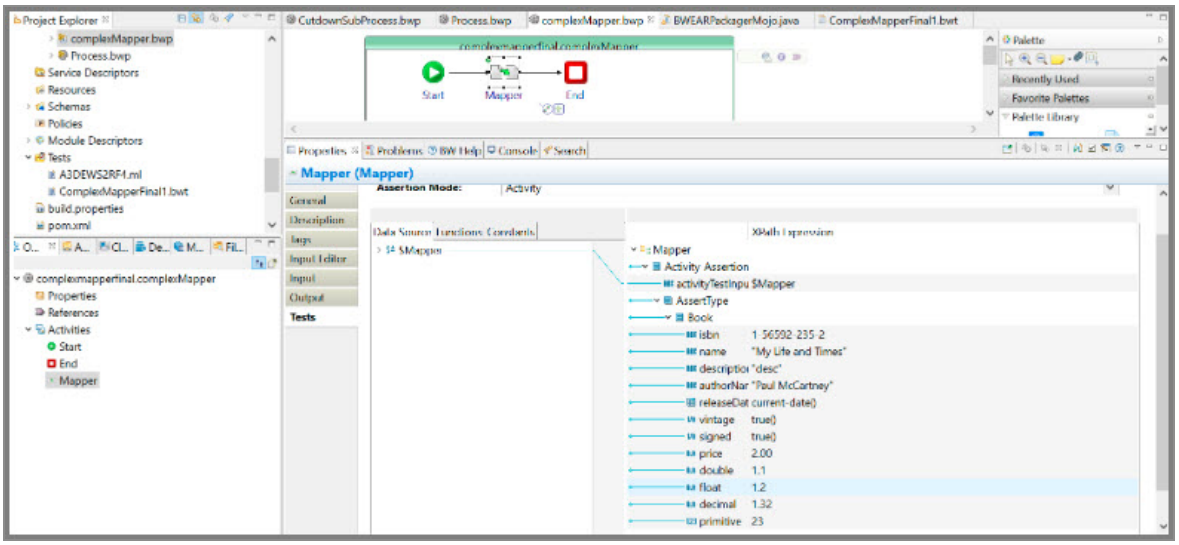



3. Select the **Activity** option from the **Assertion Mode** dropdown. The complete activity

output schema gets loaded with an editable value field under the Assert Type node. Map the activity variable from the datasource section (in Image you can see it is Mapper) to **activityTestInput** field.




4. Provide the gold input to all the elements of an activity schema that is under the assert node.



**Note:** You do not have to save the process after adding test inputs and assertions. Also if the schema having the fields with data type decimal, double, float then add the value in the decimal format, for example, 1.2 or 4.3234.

# Using Gold Input From File

### Procedure

1. Right-click the activity from the process or subprocess and select the **Add Test > Add Assertion** option.

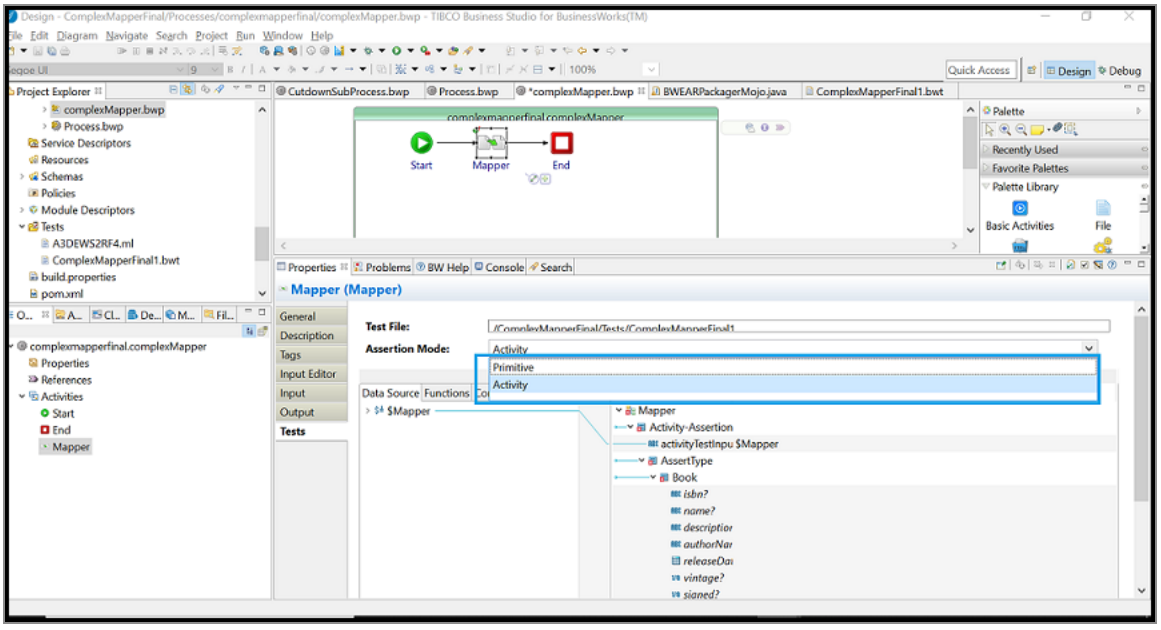
   The **Test** tab is added.

   The **Assertion Mode** dropdown list has two options: **Primitive** and **Activity**.

   Use the **Primitive** option to test only the primitive type elements.

   Use the **Activity** option to test the complete activity output that can contain a complex schema.

2. Select the Activity option.

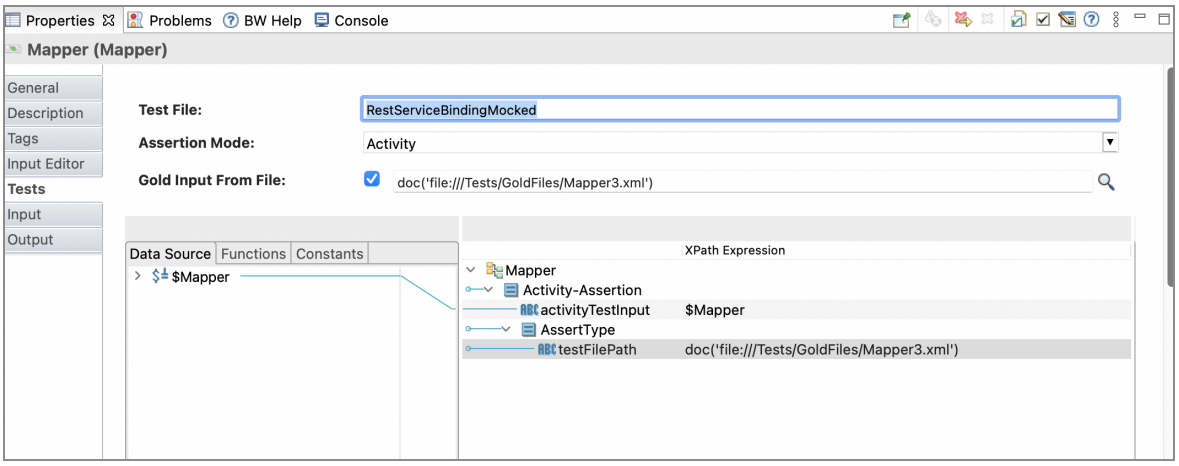



   The **Gold Input From File** checkbox is displayed.

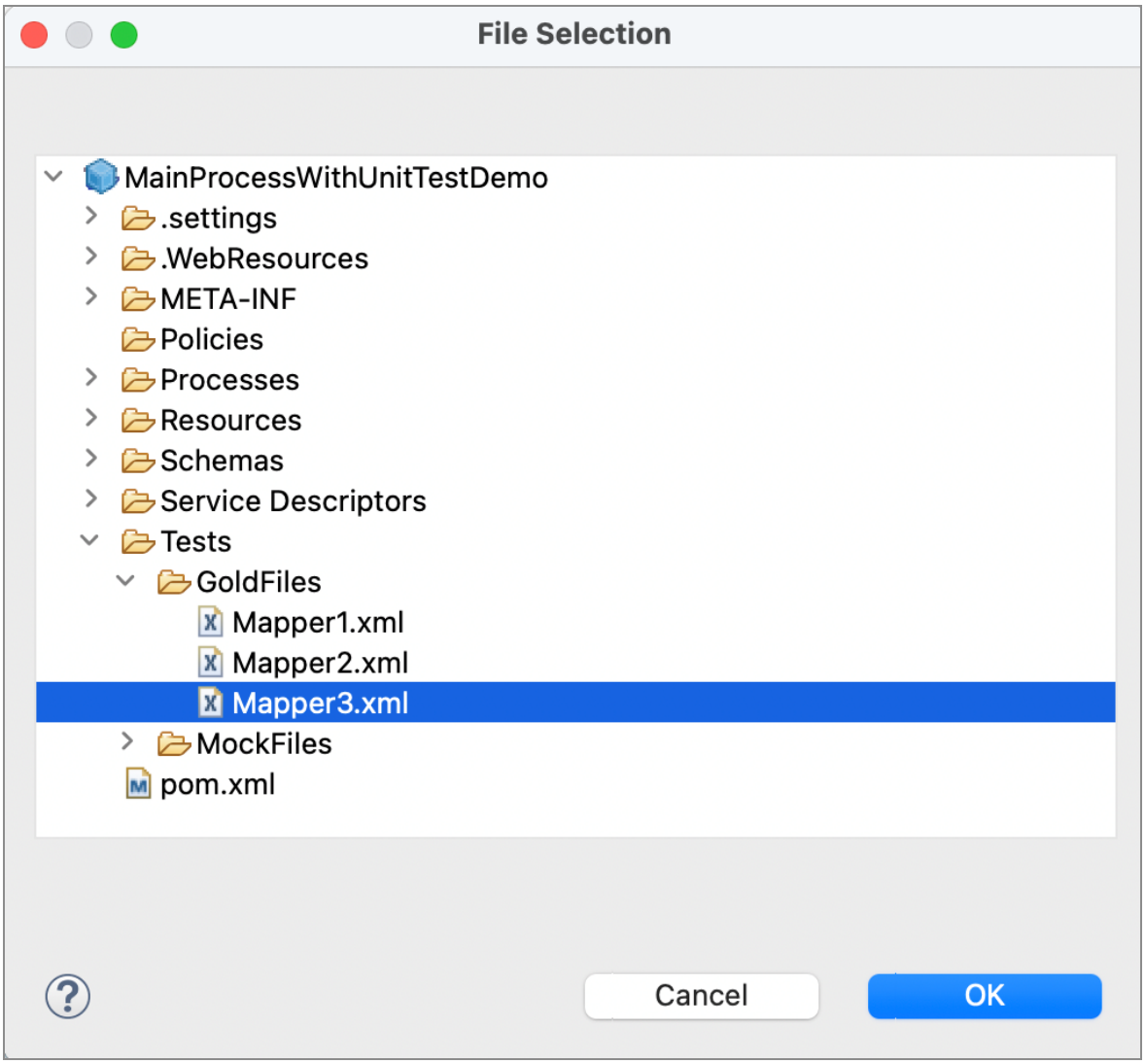
3. To provide the gold input through an XML file, select the **Gold Input From File**

checkbox.

The **AssertType** and **testFilePath** fields are displayed.




4. Map the activity variable from datasource section to **activityTestInput** field.
5. Browse the gold input file from the workspace and select the gold input file. This modifies the testFilePath file in XML.




6. Alternatively, in the **testFilePath** field, use the doc function from the URI function and provide the input file path in the format `file:///inputFilepath`. In the case of Unix systems, please provide the absolute path preceding with an extra forward slash.

   Example: doc(`file:////home/Test/Mock_files/`)

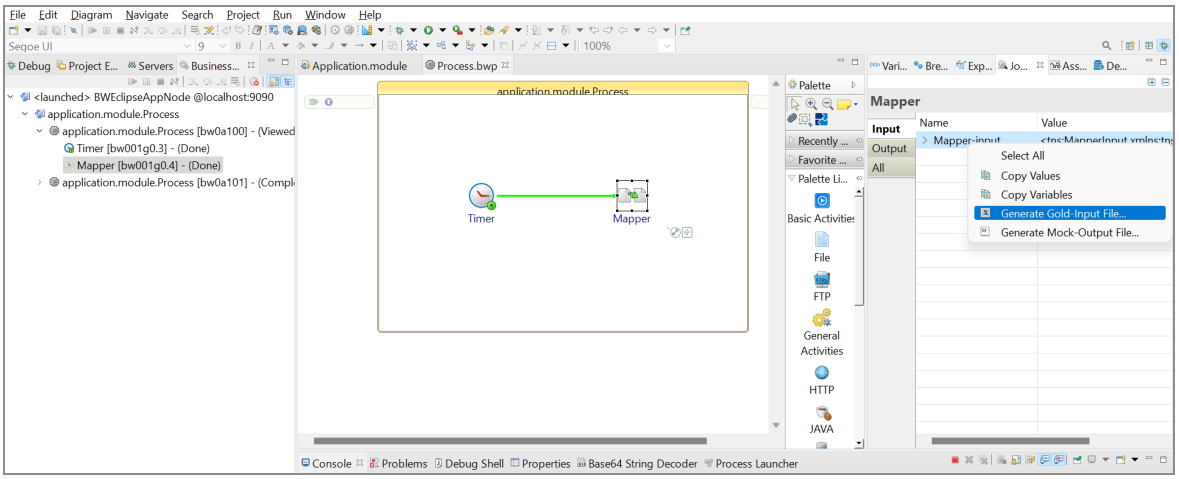
7. Provide the relative gold input file path in the **testFilePath** field.

   You can create a separate folder for gold input files under the "Tests" folder. The relative path has a value like doc `file:///Tests/UnitTestingsComplex.xml`. It is mandatory to provide the `Tests` folder name also in the relative path. In case of Unix,

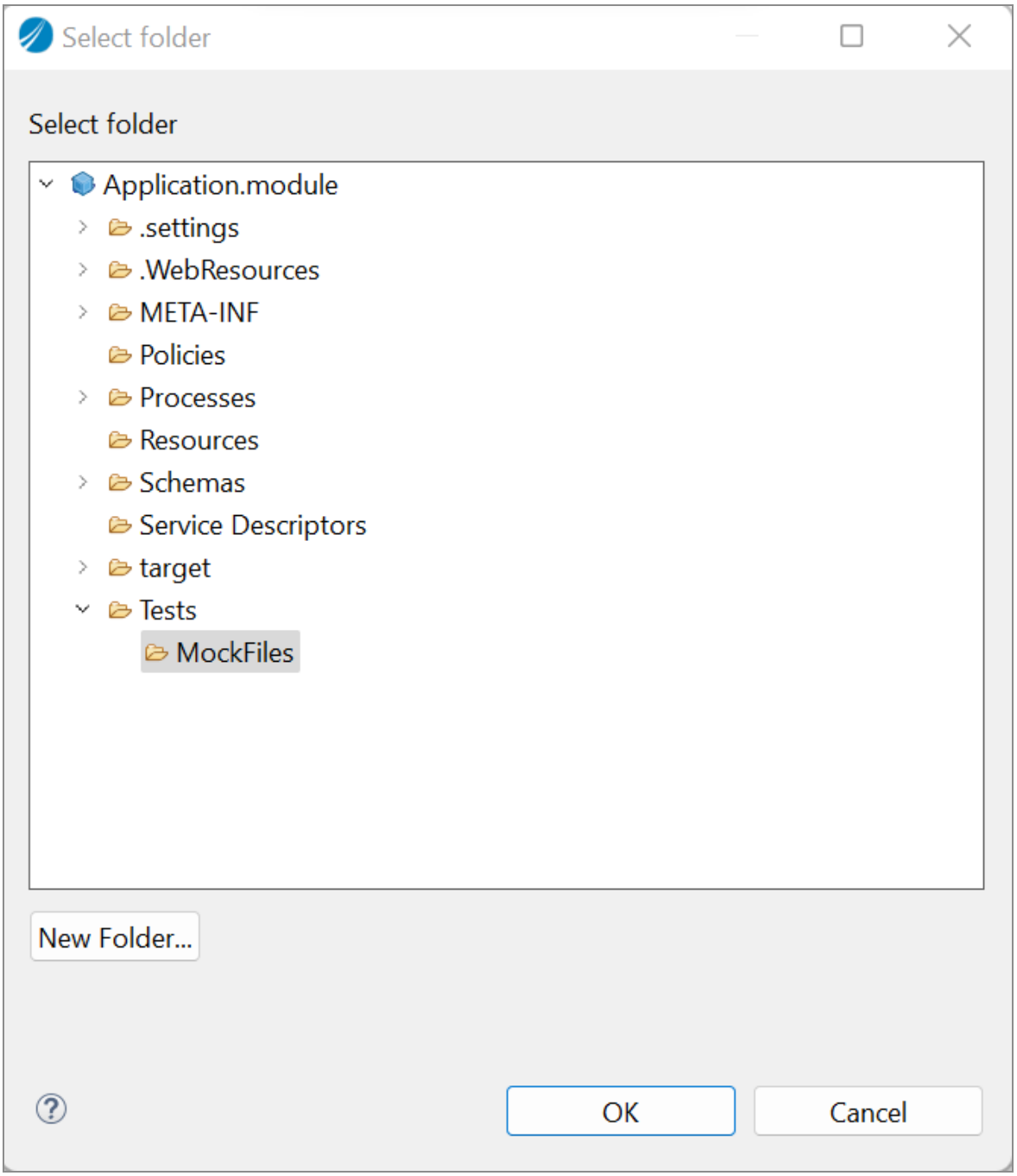
provide the relative path as `file:////Tests/Mock_files/Activity_Assetion_IP_File.xml .`

> **Note:** This feature is available with TIBCO ActiveMatrix BusinessWorks™ Maven Plug-in 2.5.0 and above.

8. To create a gold input file, run the activity for which you want to add the assertions.
9. Observe the **Tests tab > Data Source** section schema. Right-click the activity name on the Debug console and select **Generate Gold Input File** either from the Input job data or Output job data.




This opens a dialog where you can select a folder in which the Gold input file is to be created.




**Note:** Linearize the copied XML data if required.

# Working with a Test Suite

The Test Suite feature provides a functionality to run set of test cases when running the test goal.

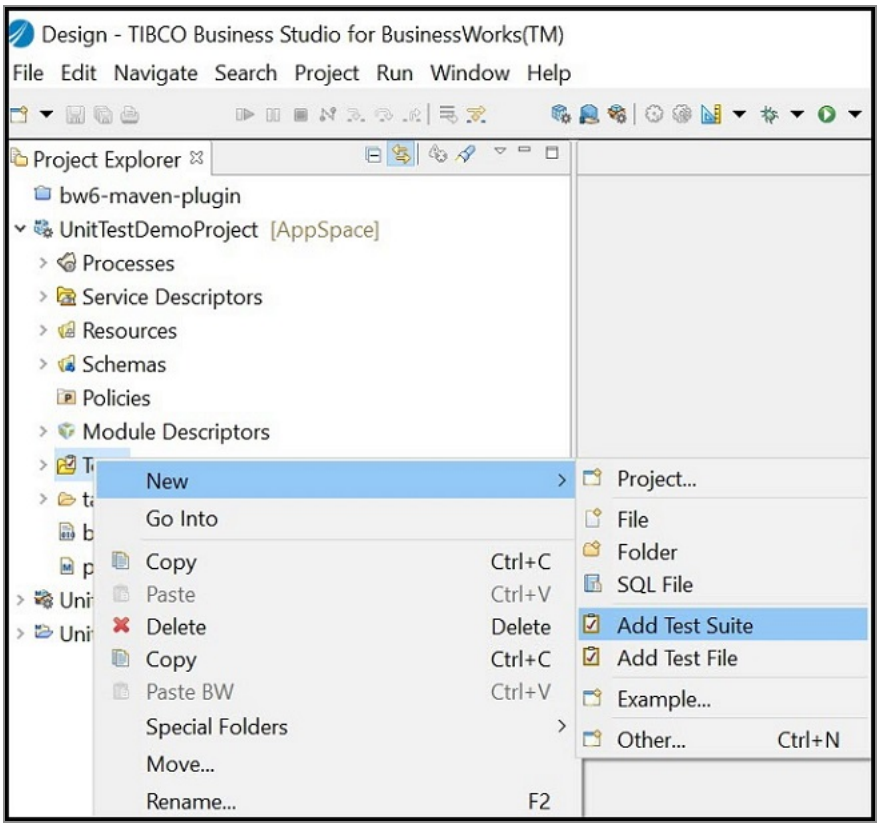
# Adding a Test Suite

### Before you begin

- Ensure you have added Unit Test Assertions. For more information, see Adding Unit Test Assertions.

### Procedure

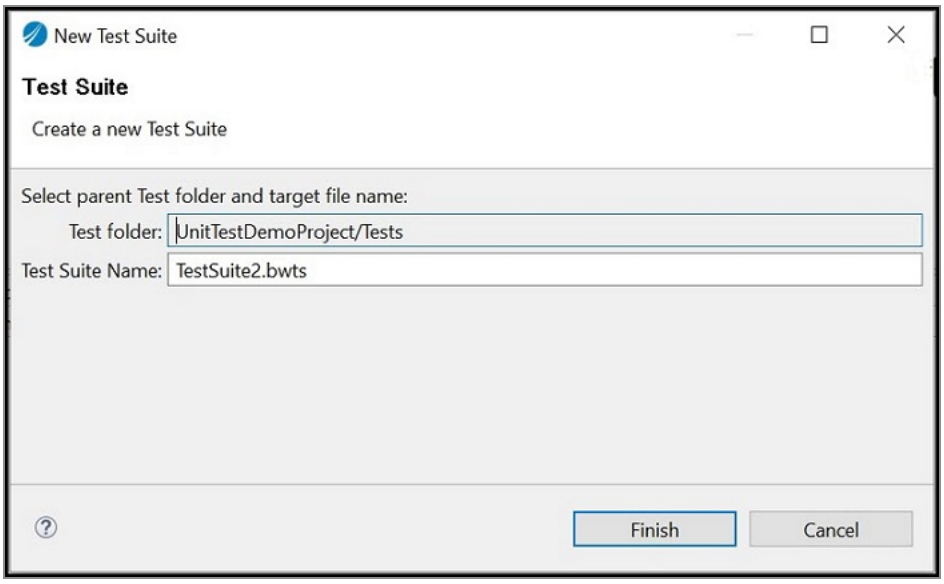
1. Right-click the `Test` folder and select the **New > Add Test Suite** option.




   The Test Suite wizard is displayed.

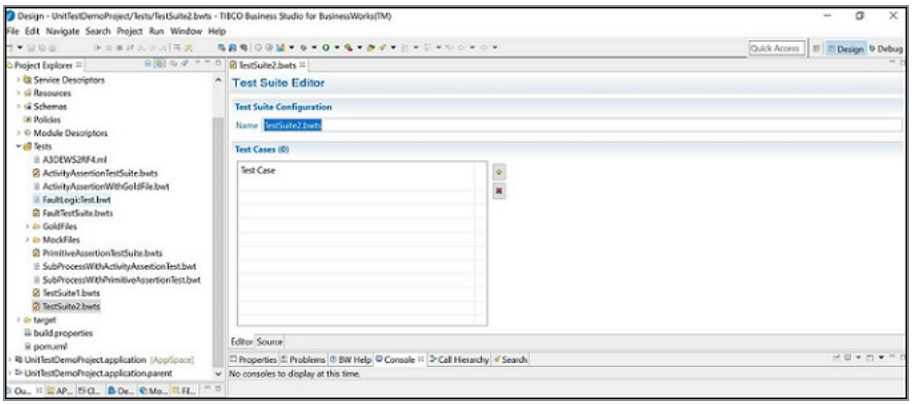
> **Note:** You can also create a Test Suite in the subfolder created under the Tests folder.

2. In the Test Suite wizard. Provide the name in the **Test Suite Name** field. Click **Finish**.



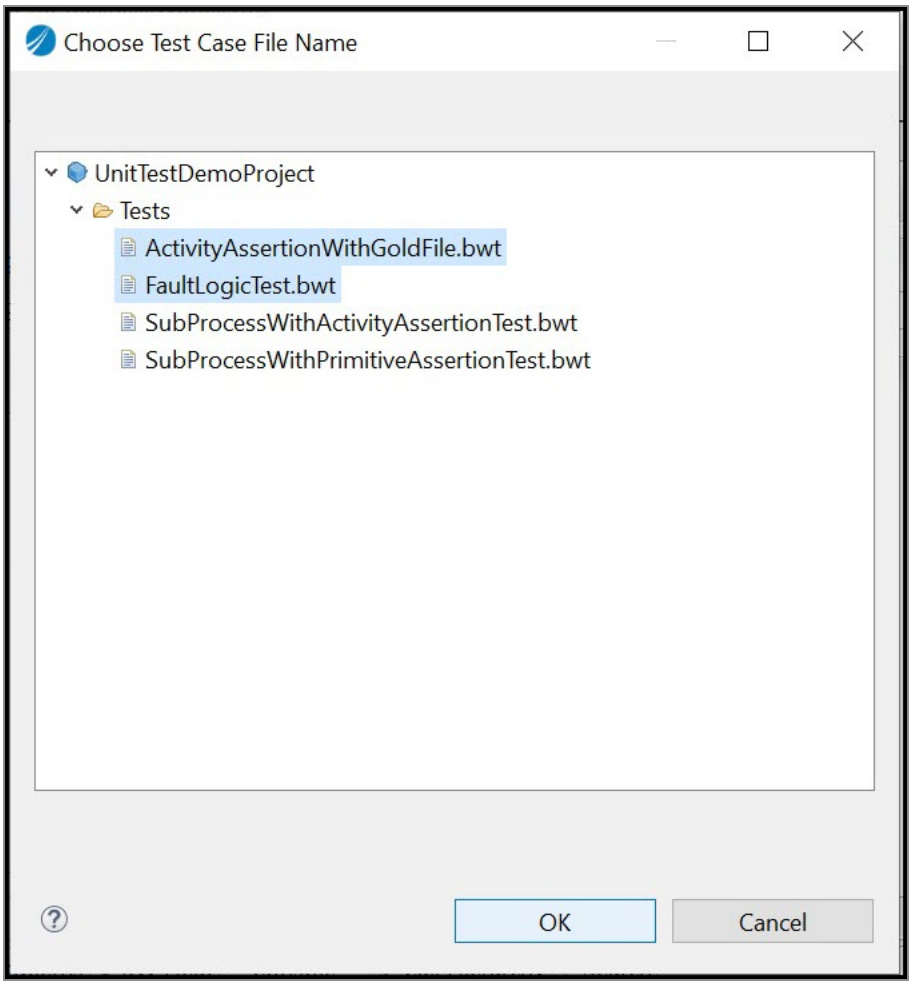

   The test suite is added to the `Test` folder.

3. Open the test suite in the **Test Suite Editor** window.




4. To add test cases in the test suite, click **Add**.
5. Select the test case. Click **Ok**.

   To add multiple test cases, use the Ctrl key and click multiple test cases.




6. To remove a test case from a test suite, select the test case and click **Remove**.

# Running a Test Suite

### Procedure

1. To run a test suite, use the property `testSuiteName` to pass the test suite name while running the test goal.

   For example:

   ```
   test -DtestSuiteName=%Test Suite Name%
   ```

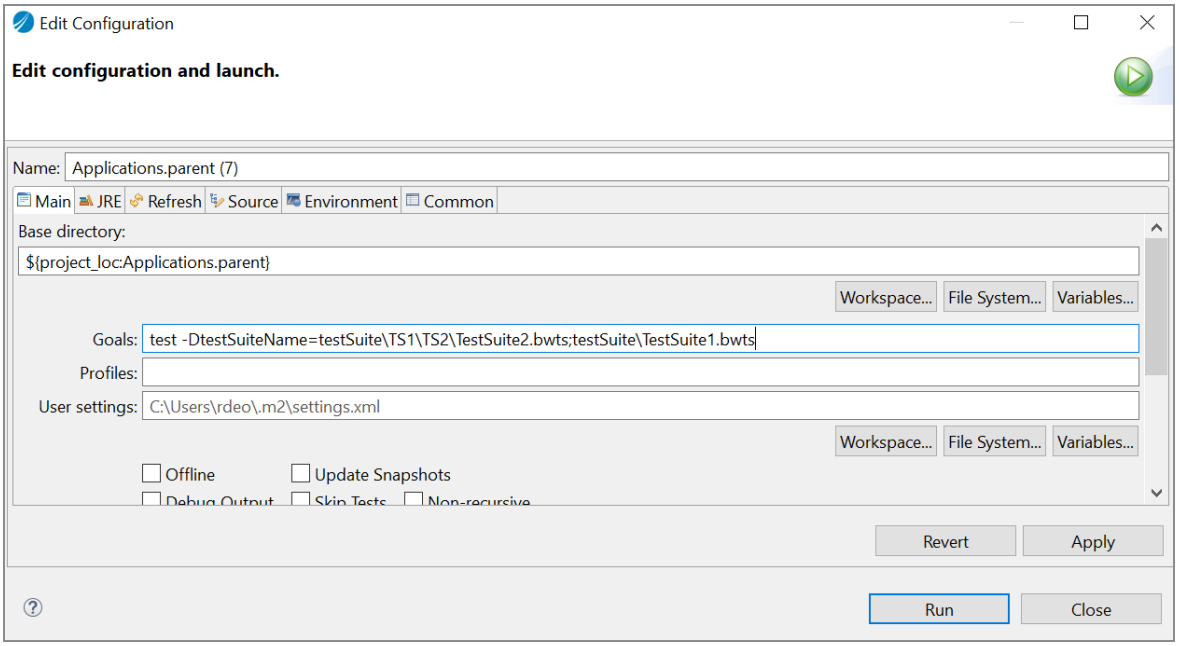
```
test -DtestSuiteName=ActivityAssertionTestSuite.bwts
```

2. To run multiple test suites in a sequence, provide the test suite names separated by ";".

   For example:

```
test -DtestSuiteName=%TestSuiteName1%;%TestSuiteName2%
```

```
test -
DtestSuiteName=ActivityAssertionTestSuite.bwts;FaultTestSuite.bwts
```




> **Note:** If you are running a test-suite present under the sub-folder of the Tests folder, then you need to provide a path of the suite from the sub-folder.

# Adding Mock Support for Activities

This section provides steps for adding mocking support for TIBCO BusinessWorks Container Edition activities with TIBCO ActiveMatrix BusinessWorks™ Plug-in for Maven. You can skip

execution of an activity (usually activities that are based on external service) whose process is under Unit Testing. Mocking support functionality is required mainly for the TIBCO BusinessWorks Container Edition activities that are based or dependent on some external Cloud Service or Database systems, which are eventually under Unit Testing. To run Unit Testing successfully on processes that contain the TIBCO BusinessWorks Container Edition activities, we need to mock the TIBCO BusinessWorks Container Edition activities. Now a dummy output can be added to mock activities that can be used in Unit testing for successful execution. The mocking support can be used to mock the activities from processes or sub-processes.

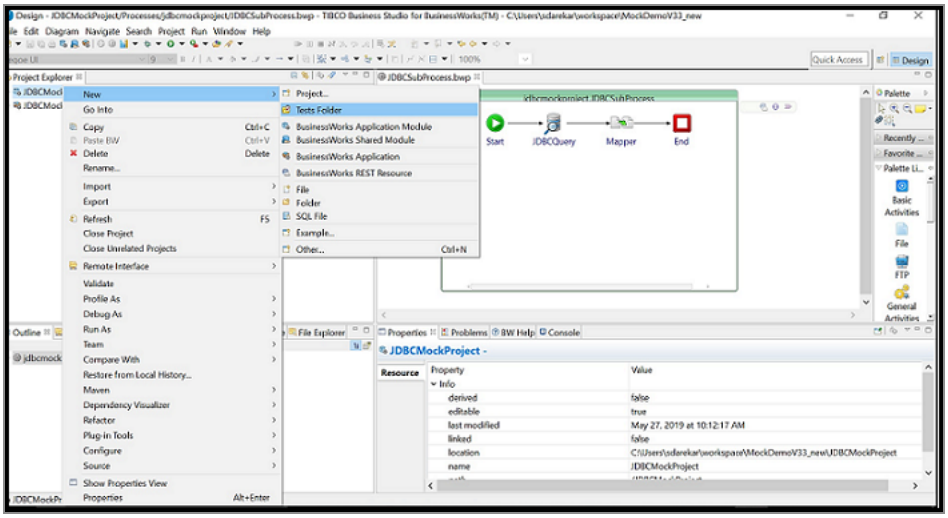
## Adding Mock Output to an Activity

To add mock output to an activity in TIBCO BusinessWorks Container Edition, follow these steps:

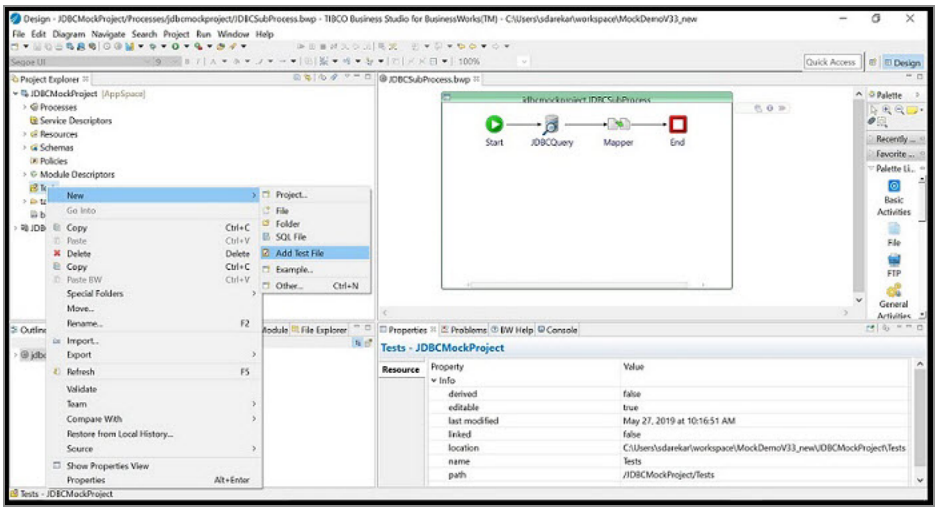
### Procedure

1. Right-click on the module project and select **New > Tests Folder**.

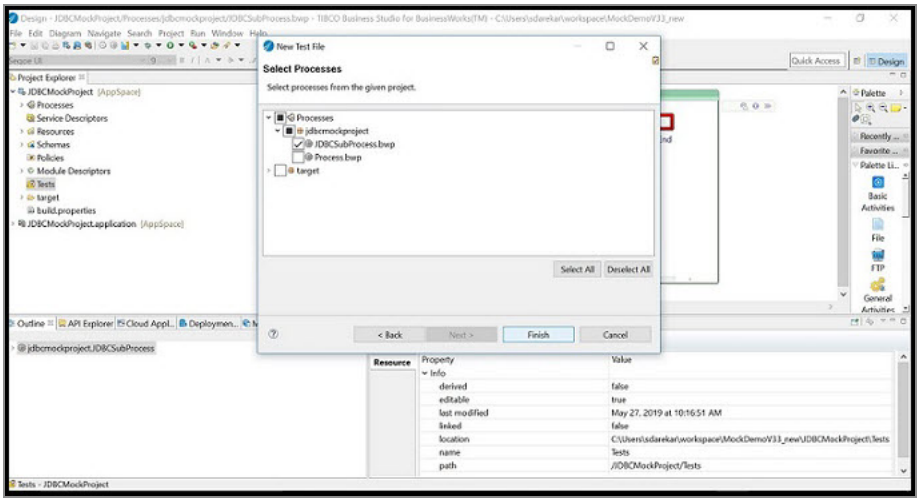
   The `Tests` folder is added in the module project.



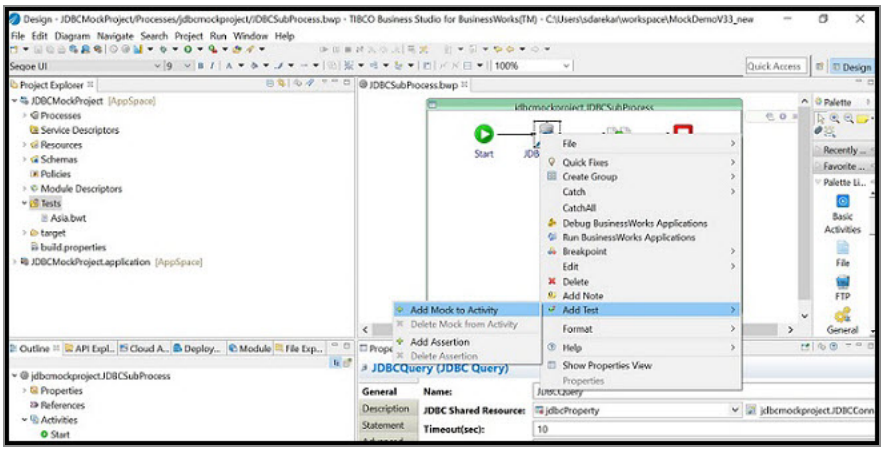

2. In Project Explorer, right-click on the `Tests` folder and choose **New > Add Test File**. If needed, change the name of the `Test` file. Click **Next**.




The New Test File wizard is displayed with a list of processes and subprocesses.

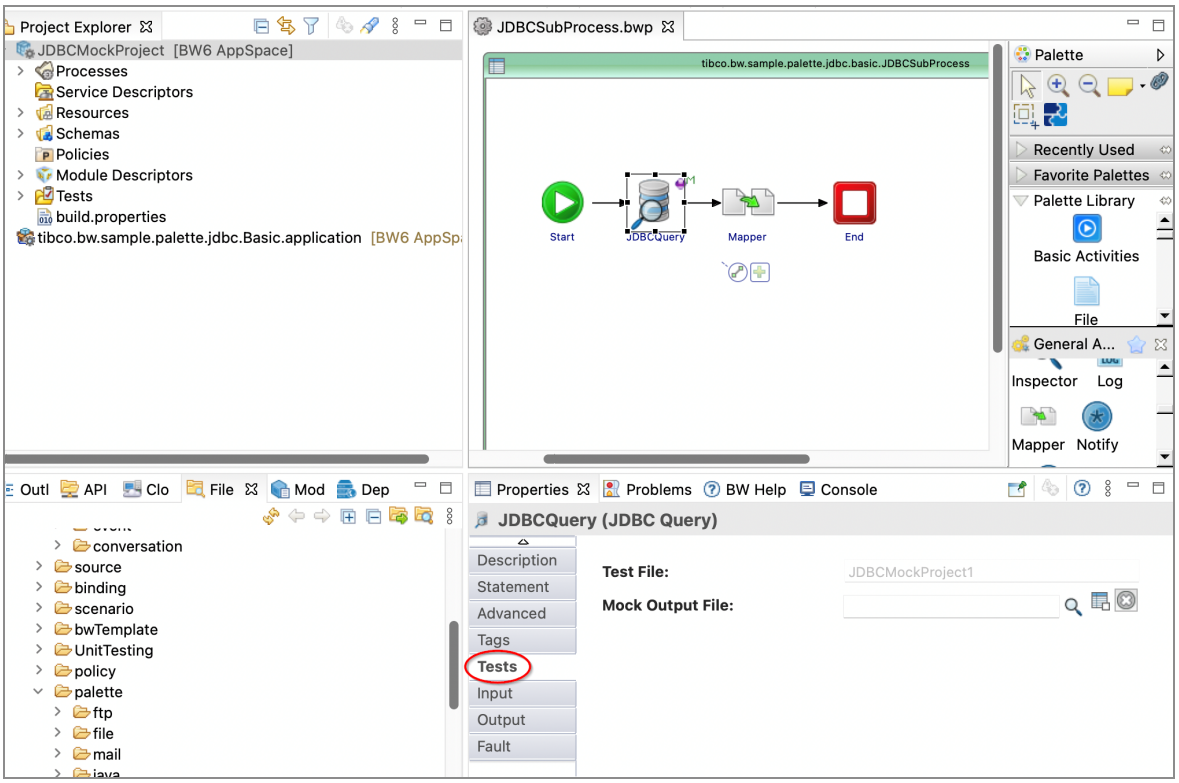



3. Select the process or subprocess having the activities to be mocked.
4. Right-click on the activity to mock and select the **Add Mock To Activity** option.



The new **Tests** tab is added in the property section of the activity.

5. The new **Tests** tab has a file selector to select the output file. Select the output file using File Selector.



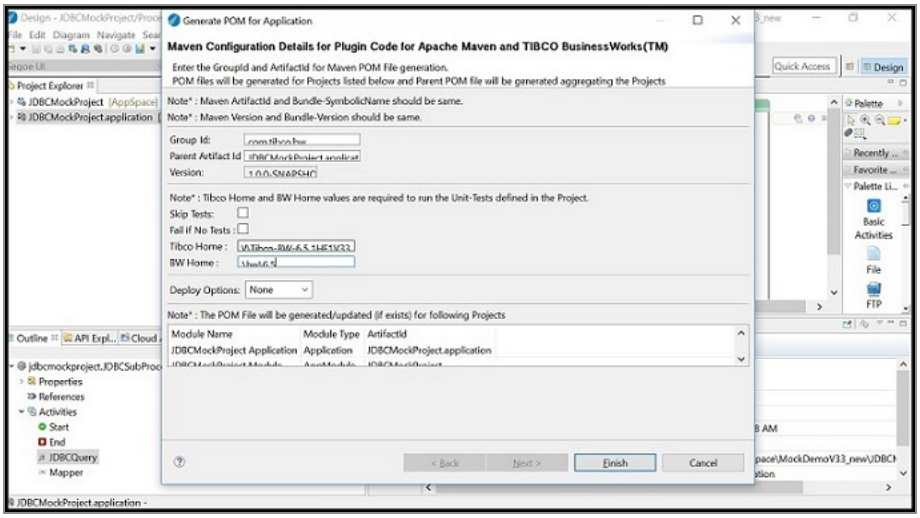

6. In the **Mock Output File** field, browse and select the file from the workspace. The path of the Mock output file can also be set using Module properties.

# Running Unit Tests in TIBCO Business Studio for BusinessWorks

Follow these steps to run unit tests in TIBCO Business Studio for BusinessWorks.

### Procedure

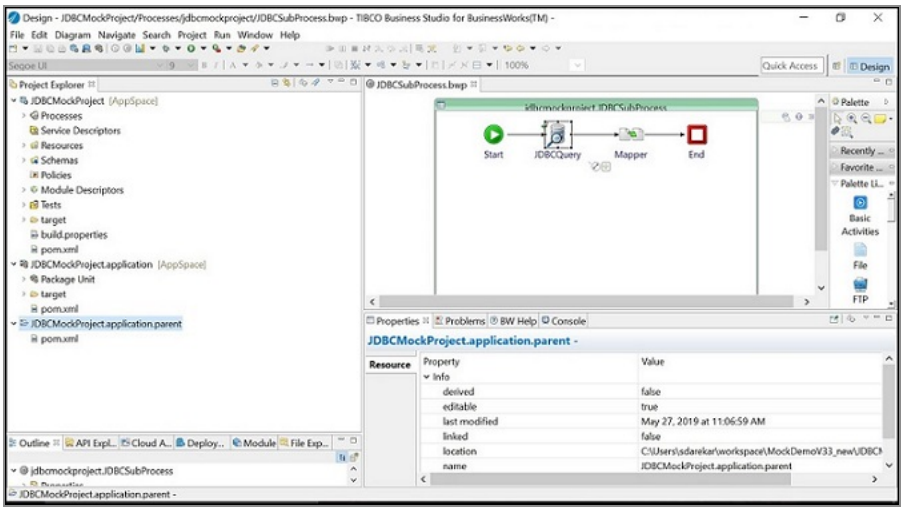
1. In TIBCO Business Studio for BusinessWorks, right-click the.`application` file and select **Generate POM for Application**.



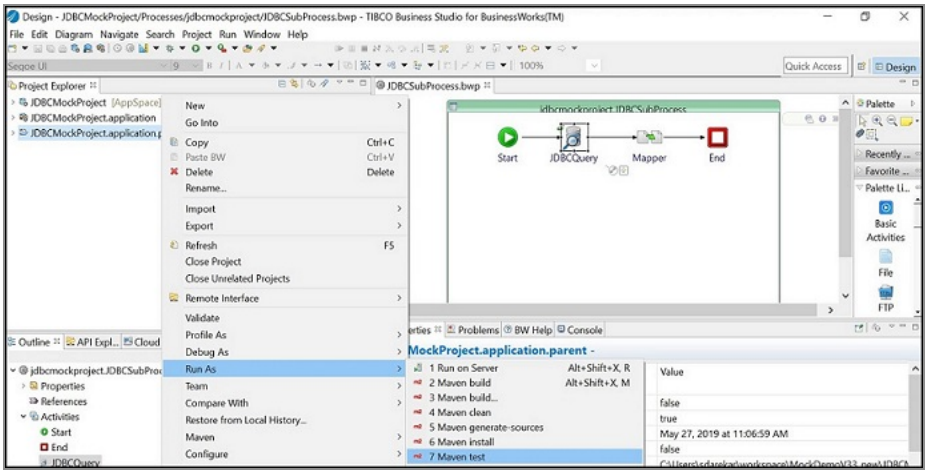

2. Verify the *TIBCO_HOME* value and click **Finish**.

   For example, `C:\tibco\bwce2.x` for Windows. Set *BWCE Home* as the relative path to the version-specific BW folder under *TIBCO_HOME* (with a leading slash and no trailing slash), for example `\bwce\2.x` for Windows.

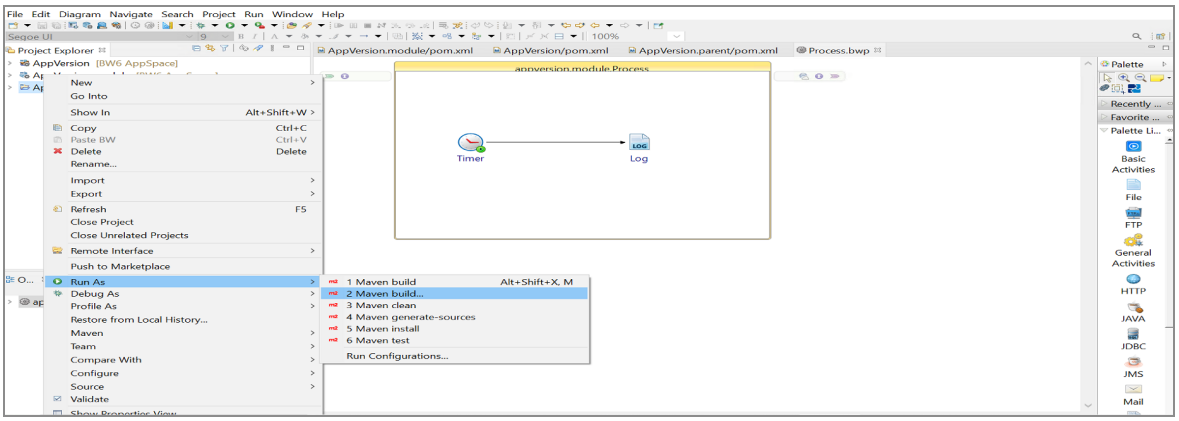
   It converts the existing projects to Maven type and adds a new project called `*.application.parent`, then creates `pom.xml` files in all projects.



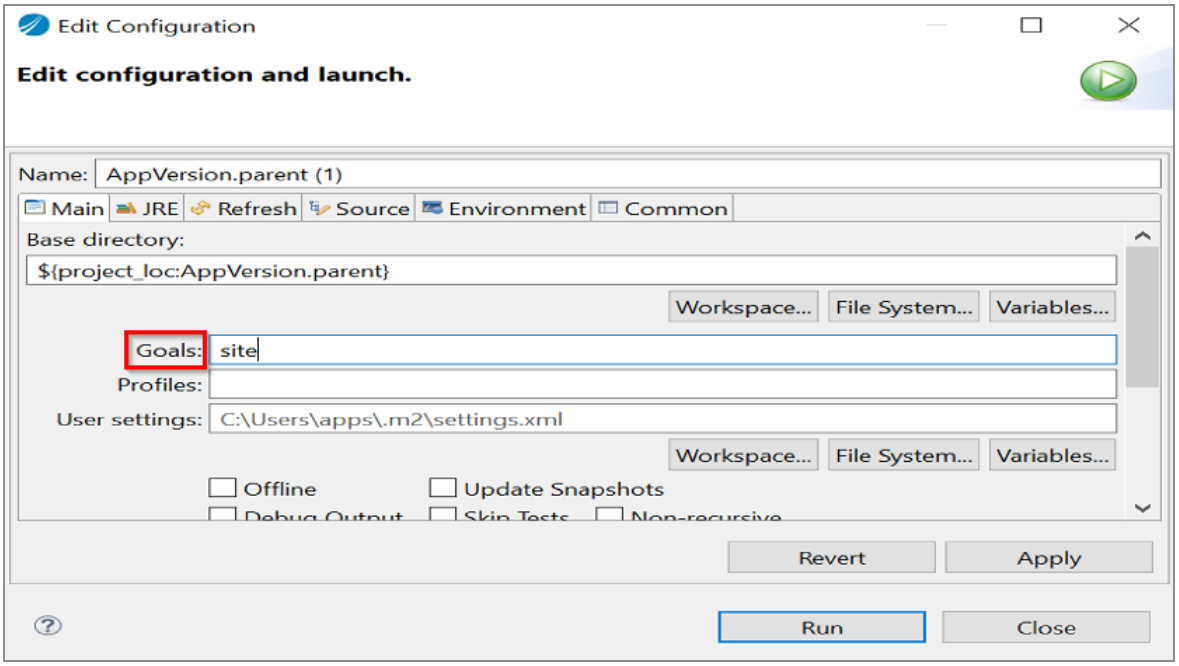
3. Right-click the parent project and run a "test" goal.



4. To run the Maven goals, right-click the `.parent` application, and select **Run As > Maven build**.




5. Provide the Maven goal in the **Goals** field that is to be executed, then click **Run**.




# Generating Mock Output File

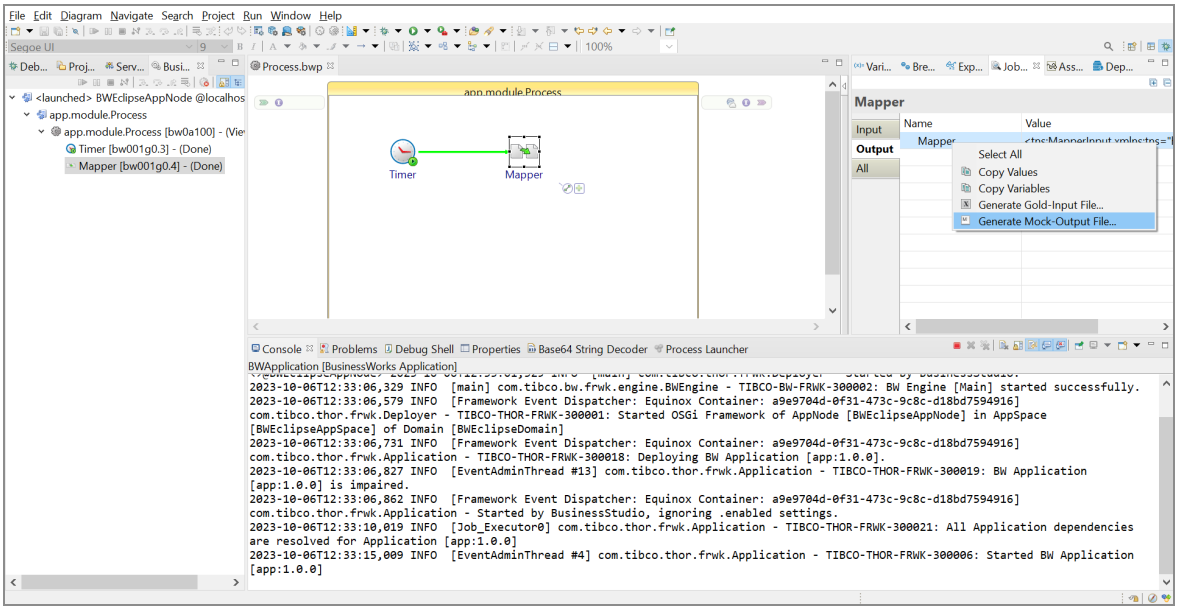
To generate the mock output files in TIBCO Business Studio for BusinessWorks, follow these steps:

### Procedure

1. Run the application in Debug mode from TIBCO Business Studio for BusinessWorks.

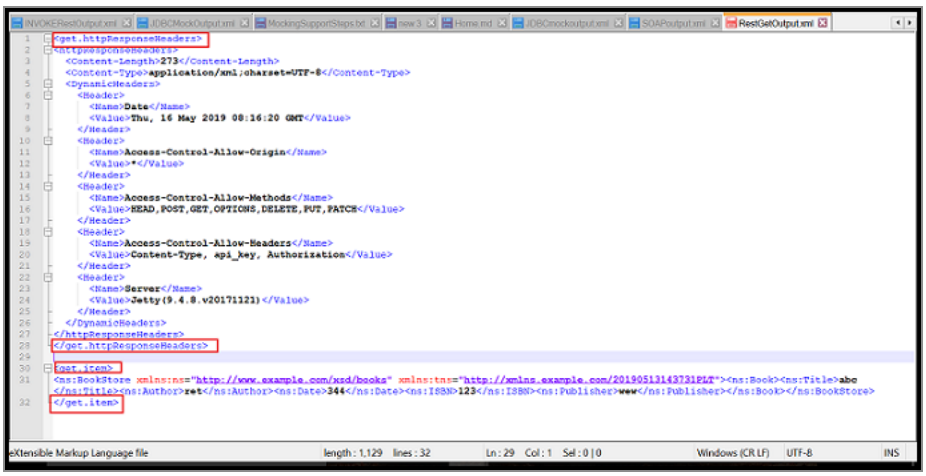
   The Debug perspective is displayed.

2. In the Debug perspective, select the **Output** tab from **Job Data** view for an activity whose mock output file is to be generated.
3. Right-click the activity name on the **Output** tab and select **Generate Mock Output File**.




This opens a dialog where you can select a folder in which the Mock output file is to be created.

4. Services like REST and SOAP can have multiple variables. So in the job data, the output is shown for multiple variables. In this case, append the file for each variable data.



# Limitations for Mock Support

The following are the limitations for Mock Support in TIBCO Business Studio for BusinessWorks.

- TIBCO BusinessWorks Container Edition needs to be installed on the same server where the tests are to be run.
- Unit Tests can currently only be invoked with Maven.

# Adding Mock Fault to an Activity

This document provides steps to add Mock Fault for activities in TIBCO BusinessWorks Container Edition with the Maven Plug-in. You can also mock faults generated by activities and test the exception handling logic, and test all the transitions.

### Before you begin

- Activities to be mocked must be present in a process or subprocess included under Unit testing.
- Generate a valid Mock Fault file. For more information on generating the mock fault file, see Generating Mock Fault File.

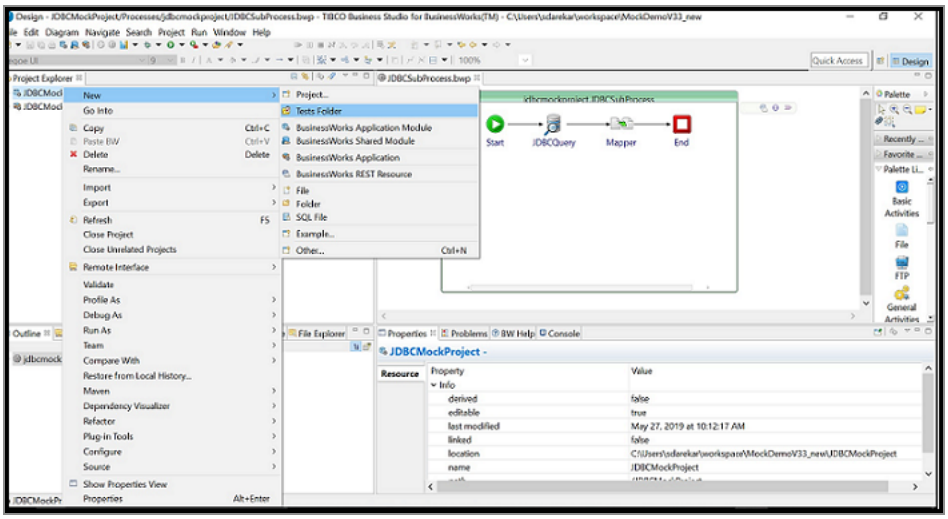
Make sure the demo project with the process or subprocess that has faults to be mocked, is created.

To add a mock fault to an activity in TIBCO BusinessWorks Container Edition, follow these steps:

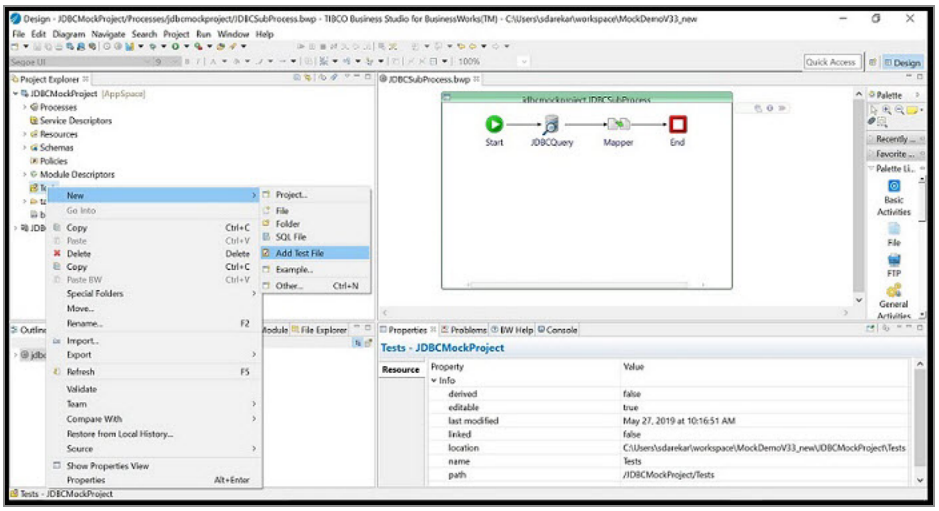
### Procedure

1. Right-click on a module project and select **New > Tests Folder**.

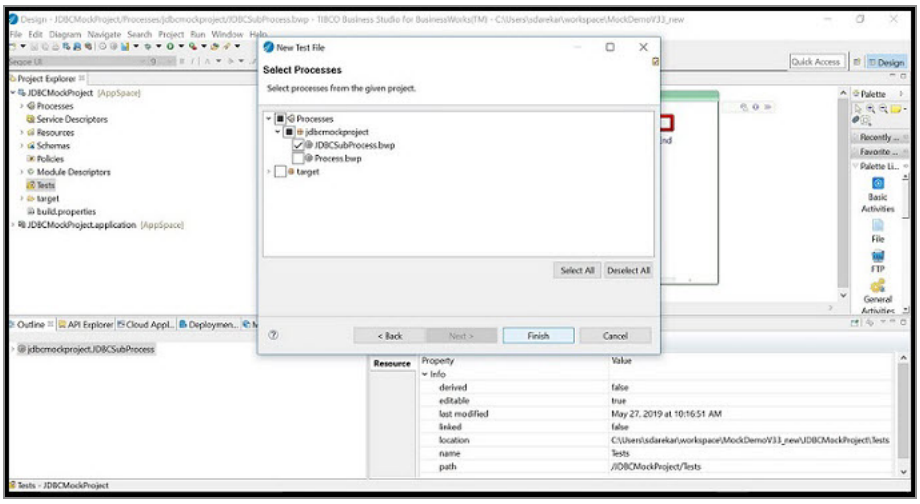
   The `Tests` folder is added in the module project.



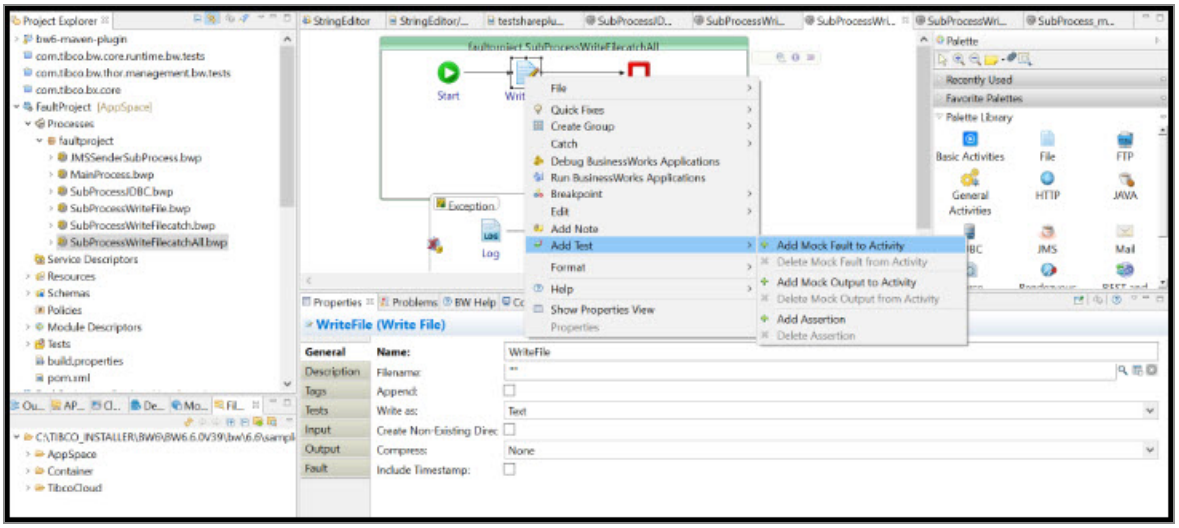

2. In Project Explorer, right-click on the `Tests` folder and choose **New > Add Test File**. If needed, change the name of the `Test` file and click **Next**.




The New Test File wizard with a list of processes or subprocesses.



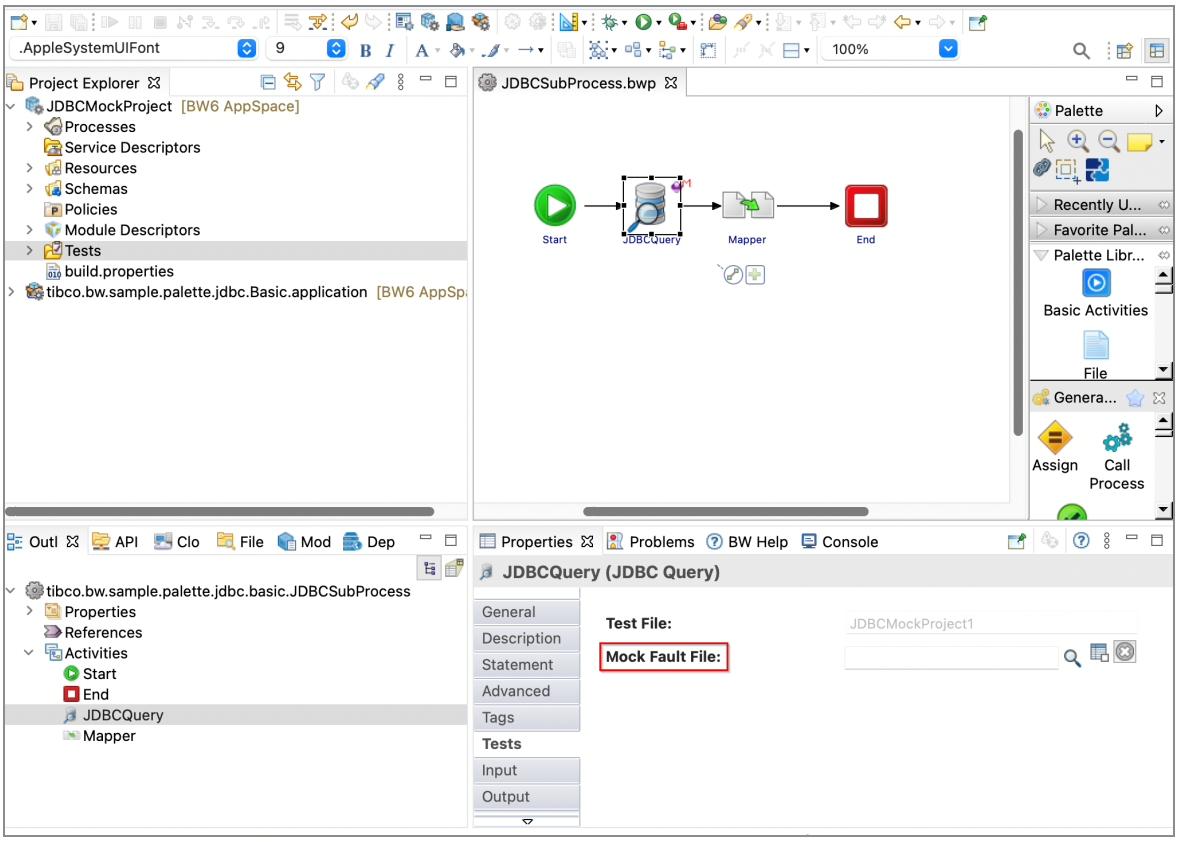

3. Select the process or subprocess which having the activities fault to be mocked.
4. Right-click on the activity to mock fault and select the **Add Mock Fault To Activity** option.




The new **Tests** tab is added in the **Properties** section of the activity.

5. The new **Tests** tab has a file selector to select the mock fault file. Select the mock fault data file using the file selector.

   You can provide the relative mock fault file in the **Mock Fault File** field.




To run Unit Tests in TIBCO Business Studio for BusinessWorks, see Running Unit Tests in Studio.

# Generating Mock Fault File

To generate the mock fault files in TIBCO Business Studio for BusinessWorks, perform the following steps:

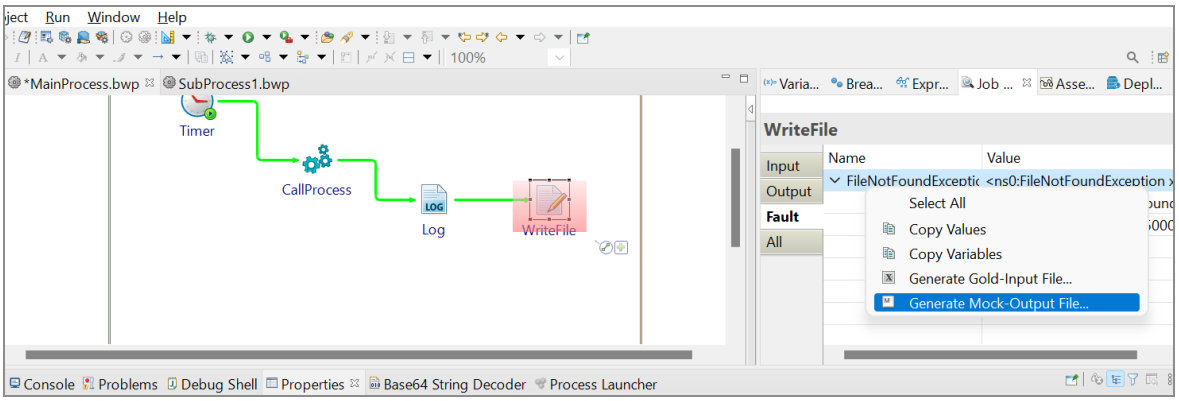
### Procedure

1. Run the application in debug mode from TIBCO Business Studio for BusinessWorks.

   The **Debug** perspective is displayed.

2. In the Debug perspective, select the **Fault** tab in the **Job Data** view for a faulted activity for which the mock fault file is to be generated.

3. Right-click on the activity name on the **Fault** tab and select **Generate Mock Output File**.




This opens a dialog where you can select a folder in which the Mock fault file is to be created.

# Limitations for the Mock Fault Support

The Mock input feature has the following limitations in TIBCO Business Studio for BusinessWorks.

- TIBCO BusinessWorks Container Edition must be installed on the same server where the tests are to be run.
- Unit Tests can currently only be invoked with Maven.

# Adding Mock Support to SOAP and REST Service Binding

This document provides steps to add Mock Input to SOAP and REST Service Binding in TIBCO BusinessWorks Container Edition with the Maven Plug-in. To mock a Service Binding, mock the respective operation and then the corresponding job flow gets run with mock input while running the test case. If a Service has multiple operations, a test file must be created for each operation and then add the mock input accordingly to test each flow associated with the operation.

### Before you begin

- Service Binding to be mocked must be present in a process under Unit testing.

- Generate a valid Mock Input file. For more information on generating the mock input file, see Generating Mock Input File.

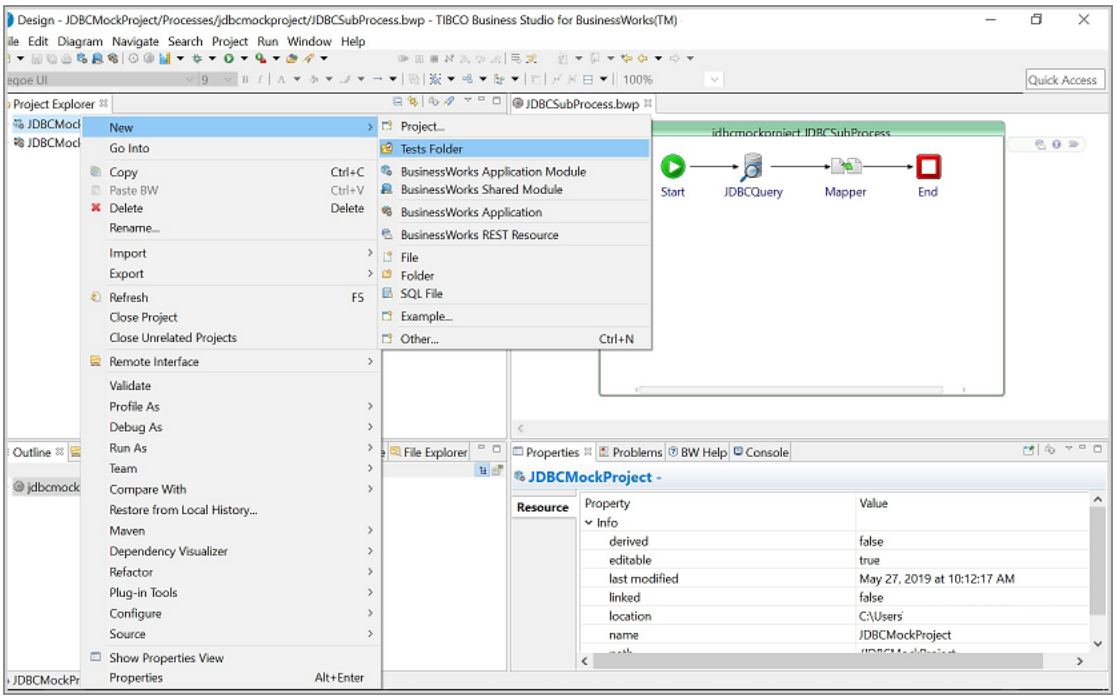
Ensure the demo project that has the service to be mocked, is created.

To add mock input for service binding in TIBCO BusinessWorks Container Edition, follow these steps:

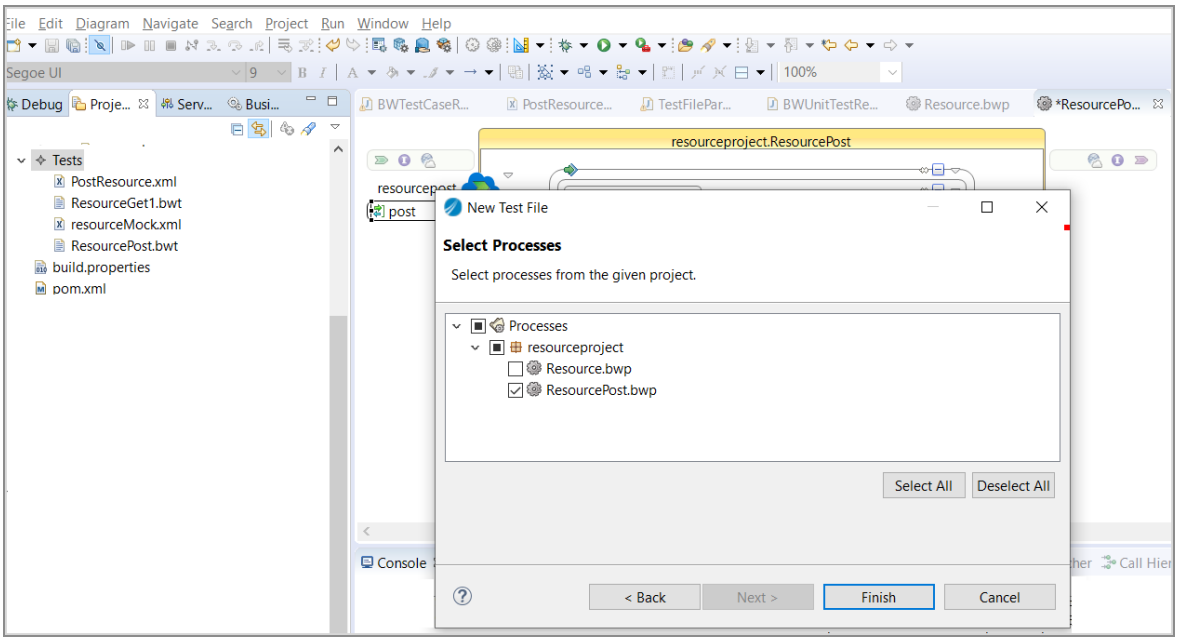
## Procedure

1. Right-click a module project and select **New > Tests Folder**.

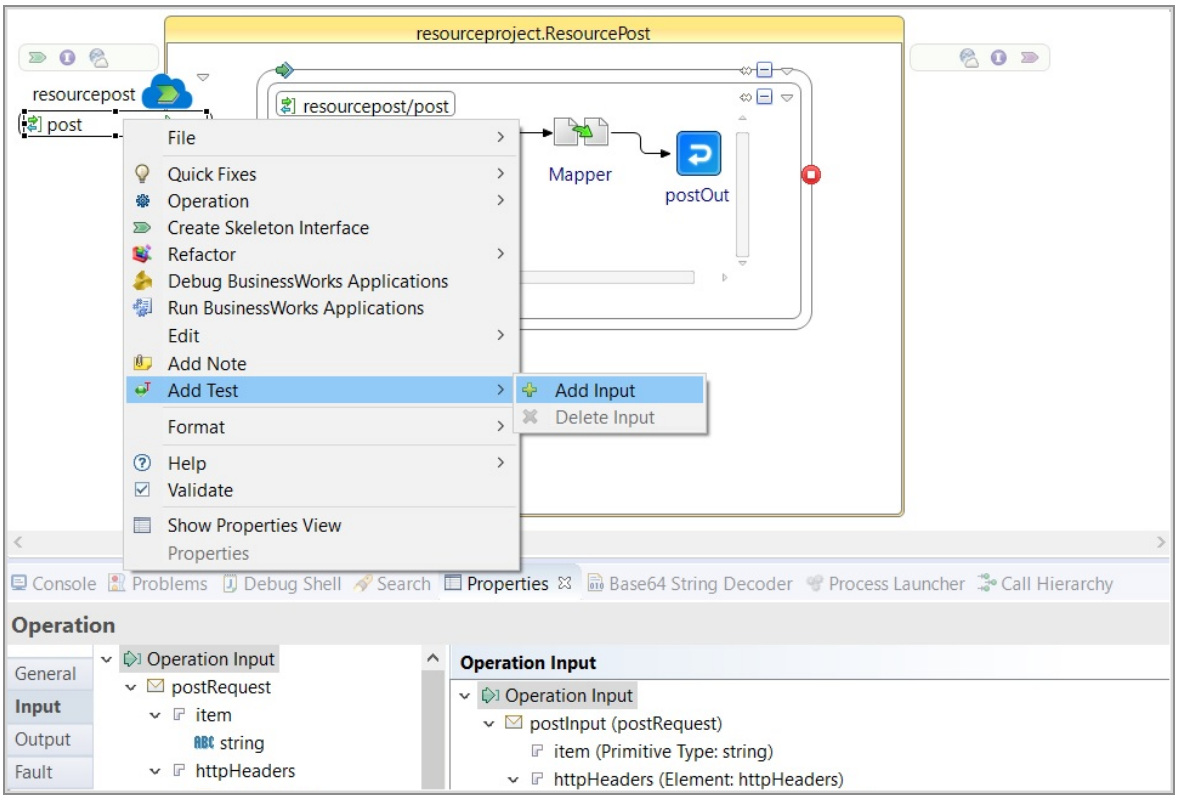
   The `Tests` folder is added in the module project.




2. In Project Explorer, right-click on the `Tests` folder and choose **New > Add Test File**. If needed, change the name of the `Test` file and click **Next**.

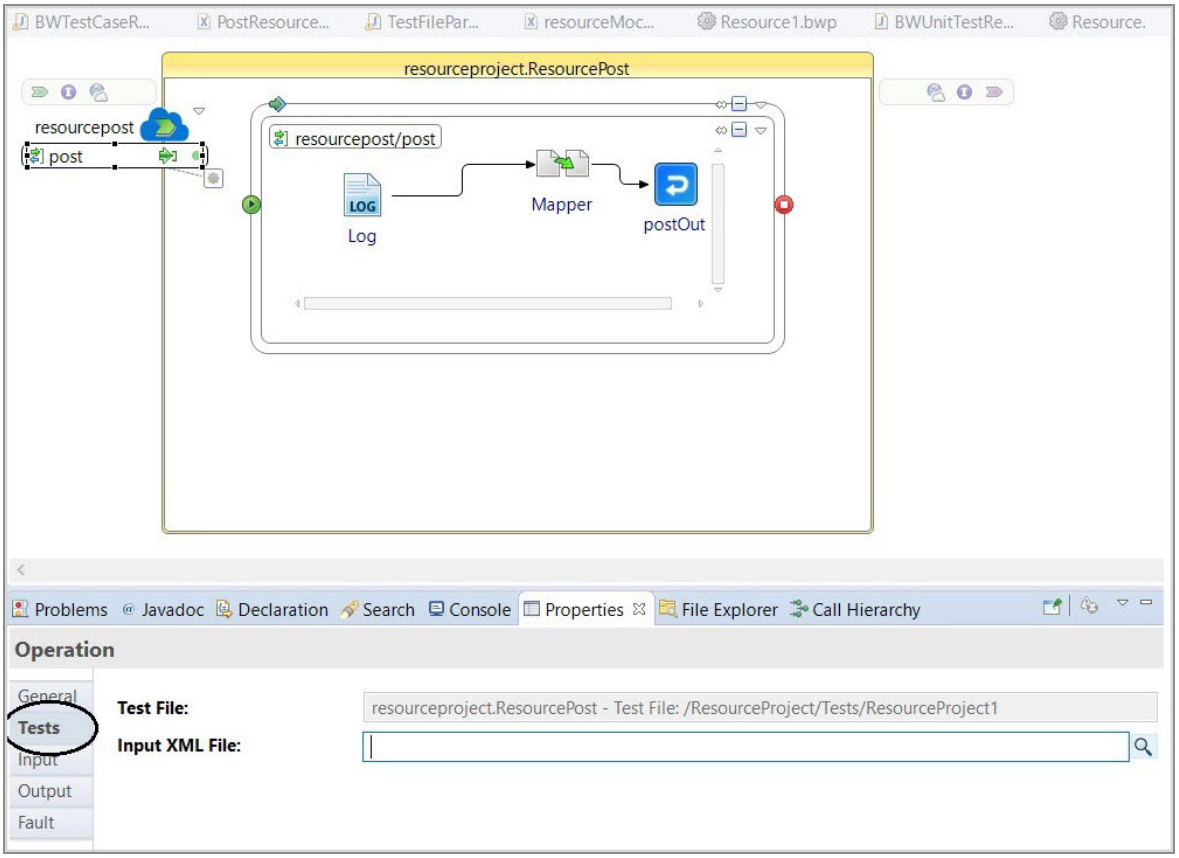



3. Select the process or subprocess having the service to be mocked.
4. Right-click on the operation to mock and select the **Add Input** option.



The new **Tests** tab is added in the **Properties** section of the activity.

5. The new **Tests** tab has a file selector to select the mock input file. Use the **Input XML File** field to select the mock input data file using the file selector.




To run Unit Tests in TIBCO Business Studio for BusinessWorks, see Running Unit Tests in Studio.

# Generating Mock Input File

To generate the mock input files in TIBCO Business Studio for BusinessWorks, perform the following steps:

### Procedure

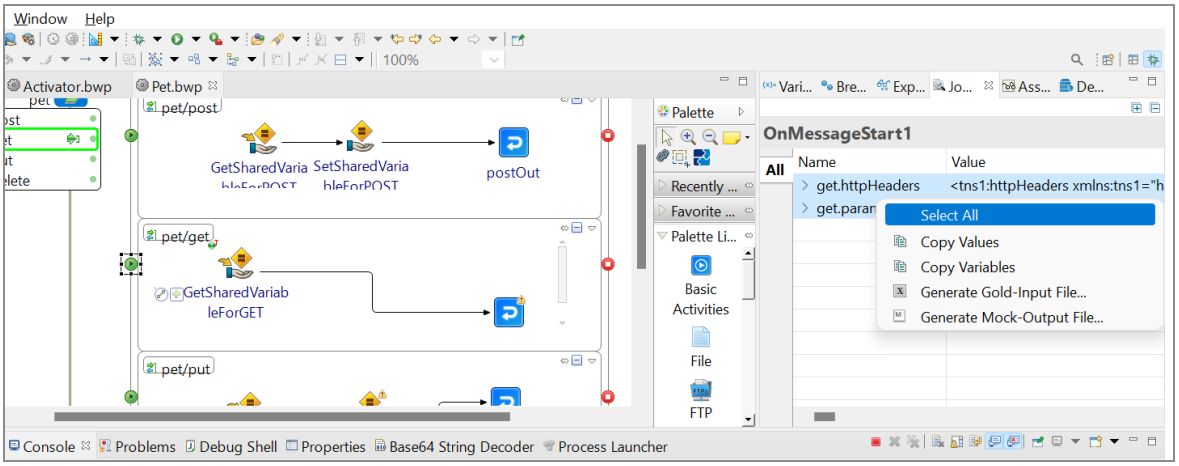
1. Run the application in debug mode from TIBCO Business Studio for BusinessWorks.

   The **Debug** perspective is displayed.

2. In the Debug perspective, select the **All** tab in the **Job Data** view for the operation for

which the mock input file is to be generated.

3. Right-click on the operation name on the **All** tab and select the **Select All** option. Then select **Generate Mock Output File**.




This opens a dialog where you can select a folder in which the Mock input file is to be created.

# Limitations for the Mock Support to SOAP and REST Service Binding

The Mock input feature has the following limitations in TIBCO Business Studio for BusinessWorks.

- TIBCO BusinessWorks Container Edition must be installed on the same server where the tests are to be run.
- Unit Tests can currently only be invoked with Maven.

**Note:** For mocking SOAP Service Binding with HTTP transport, the HTTP connection must be pointed to an unoccupied port or localhost.

# Adding Mock Support for Process Starter

Now you can add the Mock Input to the Process Starter in TIBCO BusinessWorks Container Edition 2.7.0 with Maven Plug-in 2.9.0. You can skip the execution of a Process Starter by adding the Mock Input to the Process Starter, whose process is under Unit Testing. The Assertion support is not provided to the Process Starter, because Process Starter creates TIBCO BusinessWorks Container Edition jobs continuously and they are dependent on the third party. The Unit Testing is specific with a single job only, so there is no need to add the assertion to the Process Starter. Hence, only the Mock Input support to the Process Starter is provided.

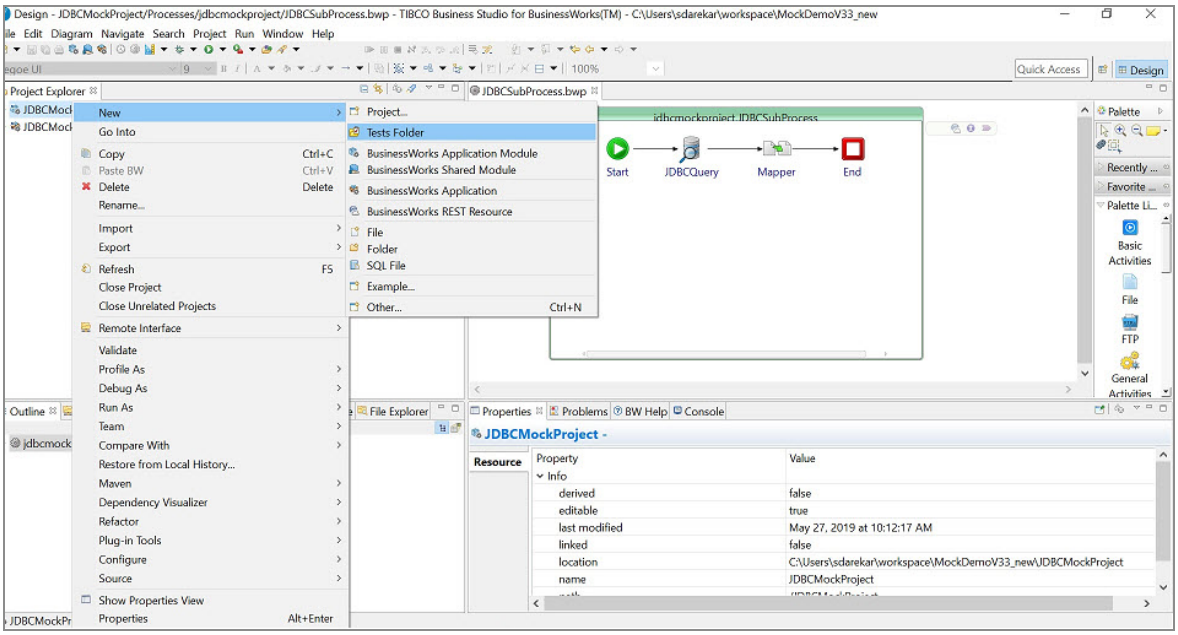
**Note:** When you want to test the activities from the Main Process, it is recommended to mock the Input of Process Starter.

### Before you begin

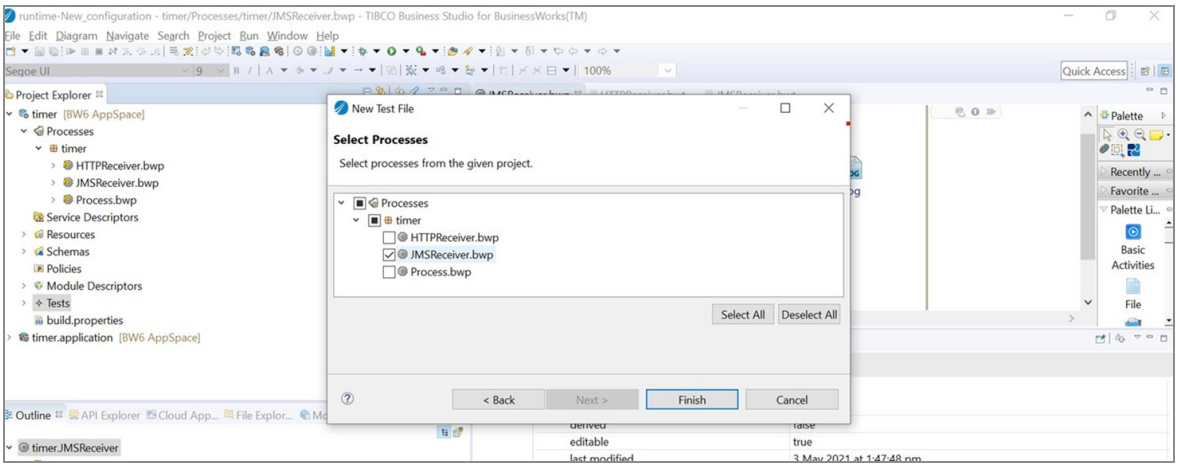
- Process Starter to be mocked should be present in the process, which is under Unit Testing.
- Generate a valid Mock Input XML file. For more information, see Generating the Mock Input File.

### Procedure

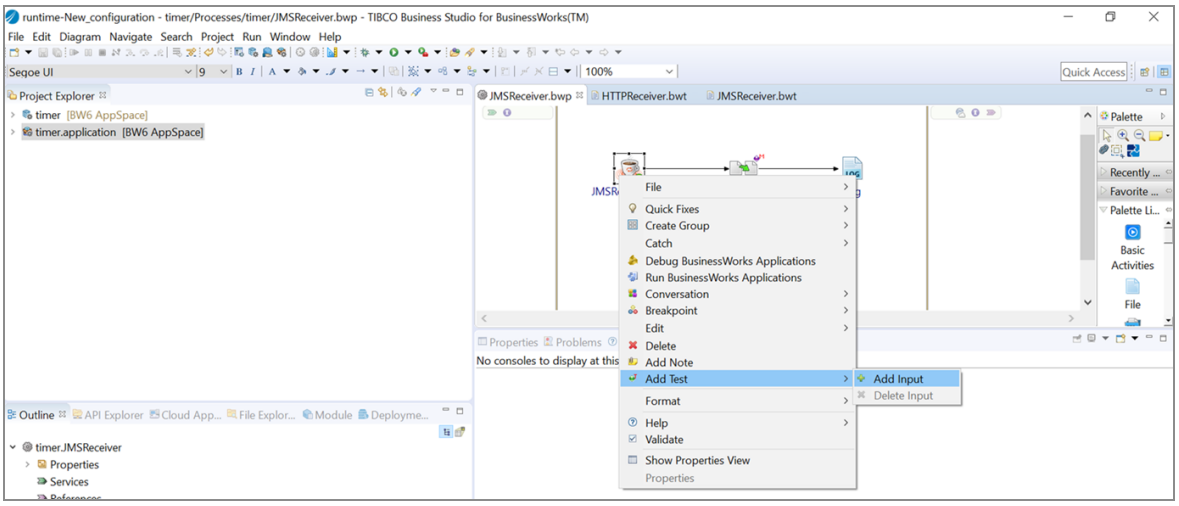
1. Right-click the module project and select **New > Tests Folder**. This adds the **Tests** folder in the module project.



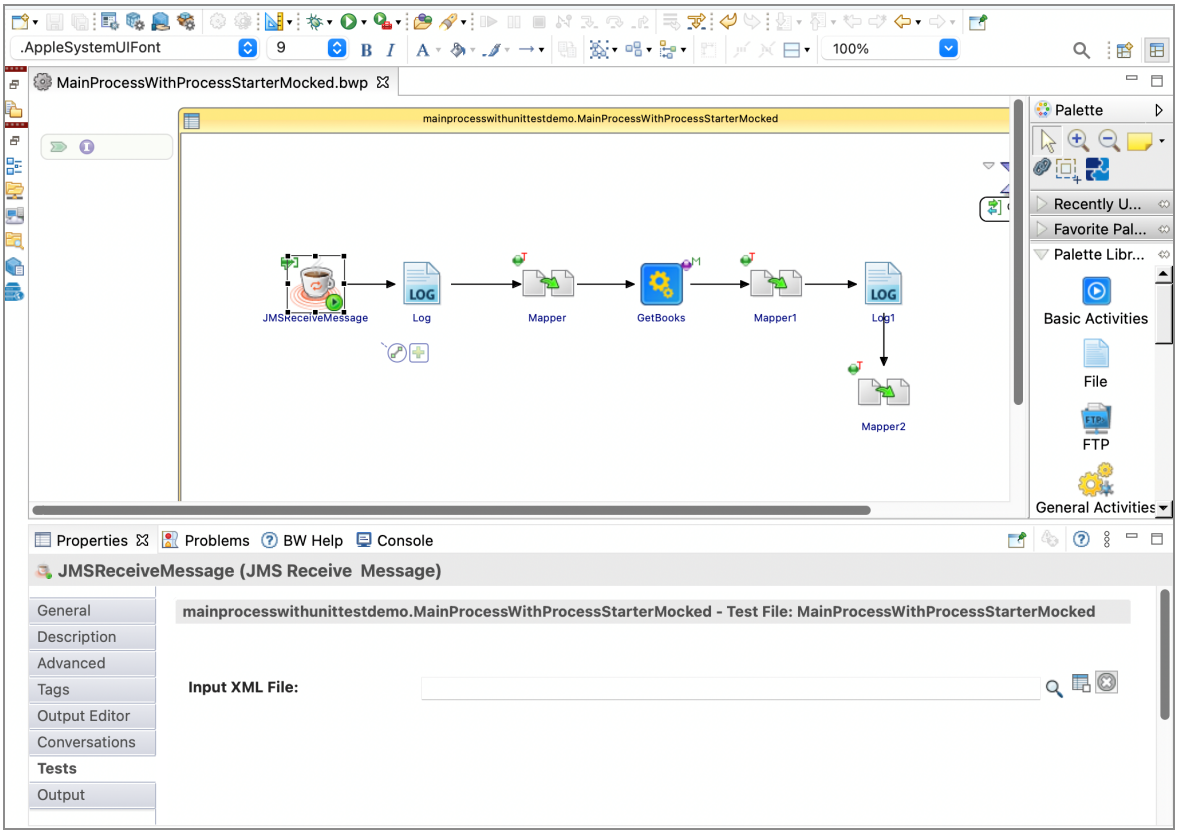

2. Right-click the **Tests** folder in the Project Explorer pane and select **New > Add Test File**. Change the test file name if required and click **Next**. This shows the **New Test File** wizard with a list of available processes.
3. Select the process having the Process Starter to be mocked.




4. Right-click the Process Starter to mock, click **Add Test > Add Input**.




The **Tests** tab is added in the Properties section of the activity. The **Tests** tab contains the **Input XML File** option to select the path of the Mock Input XML file. The path of the Mock input file can also be set using Module properties.

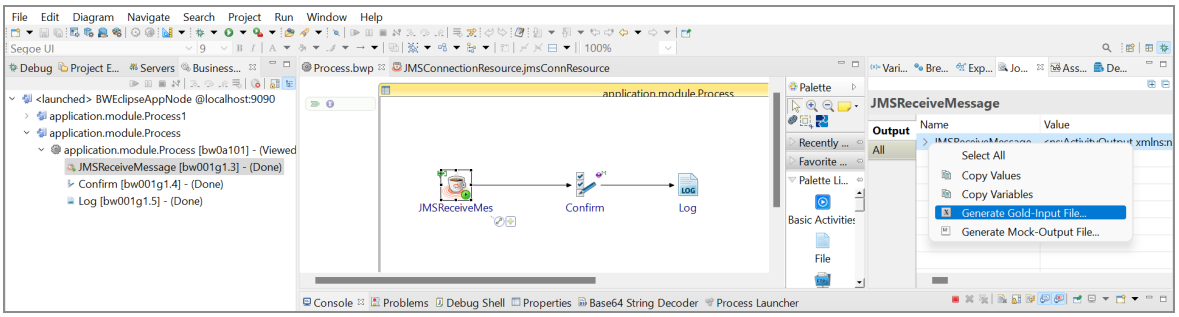


# Generating the Mock Input File

To generate the Mock input file, perform the following steps:

### Procedure

1. Run the application in the debug mode from TIBCO Business Studio for BusinessWorks.
2. Select the **Output** tab from the Job Data for Process Starter for which the Mock input file is to be generated.
3. Right-click the activity name in the Output tab and select the **Select All** option to select all the data. Then select **Generate Gold Input File**.




   This opens a dialog where you can select a folder in which the Mock input file is to be created.

# Limitations for Mock Process Starter

The Mock Support for Process Starter feature has the following limitations in TIBCO Business Studio for BusinessWorks:

- TIBCO BusinessWorks Container Edition must be installed on the same server where the tests are to be run.
- Unit Tests can currently only be invoked with Maven.

# Running ActiveMatrix BusinessWorks Design Utility Goal

The `bwdesignUtility` Maven goal provides a command-line interface to validate the ActiveMatrix BusinessWorks project and generate the process diagram.

Assuming that the user already has the ActiveMatrix BusinessWorks Unit Test Project with POM generated and includes a valid `TIBCO_Home` and `BW_Home`, follow these steps to run the `bwdesignUtility` goal:

### Procedure

1. Navigate to the ActiveMatrix BusinessWorks unit test `parent` project workspace and open the command prompt or Git Bash.
2. To validate the ActiveMatrix BusinessWorks project, run the `bwdesignUtility` goal by passing the `commandName` argument with value `validate` in the following way:

   ```
   mvn com.tibco.plugins:bw6-maven-plugin:bwdesignUtility -
   DcommandName=validate
   ```

   This validates the project.
3. To generate the process diagram for the project, enter the `bwdesignUtility` goal by passing the `commandName` argument with value `gen_diagrams` in the following way:

   ```
   mvn com.tibco.plugins:bw6-maven-plugin:bwdesignUtility -
   DcommandName=gen_diagrams
   ```

   This generates a process diagram in the Resources folder of the ActiveMatrix BusinessWorks application project.
4. To generate the Manifest JSON file from the project whose deployment target is `TibcoCloud`, enter the `bwdesignUtility` goal by passing the `commandName` argument with a value `generate_manifest_json` in the following way:

   ```
   mvn com.tibco.plugins:bw6-maven-plugin:bwdesignUtility -
   DcommandName=generate_manifest_json
   ```

   This generates the manifest JSON file in the ActiveMatrix BusinessWorks application project.

> **Note:** This goal is available from Maven Plug-in version 2.8.1 onwards.

5. To validate the ActiveMatrix BusinessWorks project, generate the process diagram and the manifest JSON file for the project sequentially, run the `bwdesignUtility` command without passing an argument `commandName`.

```
mvn com.tibco.plugins:bw6-maven-plugin:bwdesignUtility
```

# Using Custom Xpath Functions with TIBCO ActiveMatrix BusinessWorks Plug-in for Maven

To use custom XPath function with TIBCO ActiveMatrix BusinessWorks Plug-in for Maven, perform the following steps:

### Procedure

1. Create a custom XPath function project with TIBCO BusinessWorks Container Edition. For more information, see "Creating Custom XPath Functions" in *TIBCO BusinessWorks Container Edition Bindings and Palette Reference*.
2. Create a sample BW application using the custom Xpath function created in Step 1.
3. In the Project Explorer, ensure that the custom xpath function project is added in the **Includes** application.
4. To generate the POM files, right-click the project and select **Generate POM for application**. The parent `pom.xml` project must list down all the modules as below:

```
<modules>
       <module>../CXFDemo</module>
       <module>../CXFTest.module</module>
       <module>../CXFTest</module>
</modules>
```

5. Add the `cxf` common extension dependency in the custom XPath function `pom.xml` project.

```
<dependencies>
	<dependency>
		<groupId>com.tibco.plugins</groupId>
		<artifactId>com.tibco.xml.cxf.common</artifactId>
		<version>${cxf.common.version}</version>
		<scope>provided</scope>
	</dependency>
<dependencies>
```

**Note:** Replace the `${cxf.common.version}` with version available in the BW_HOME. For example, 1.3.400.

6. Create a maven run configuration. Select the BW application parent project as the base directory.
7. Provide the maven goal `clean Test`.
8. To generate the EAR, provide the Maven goal `clean package`.

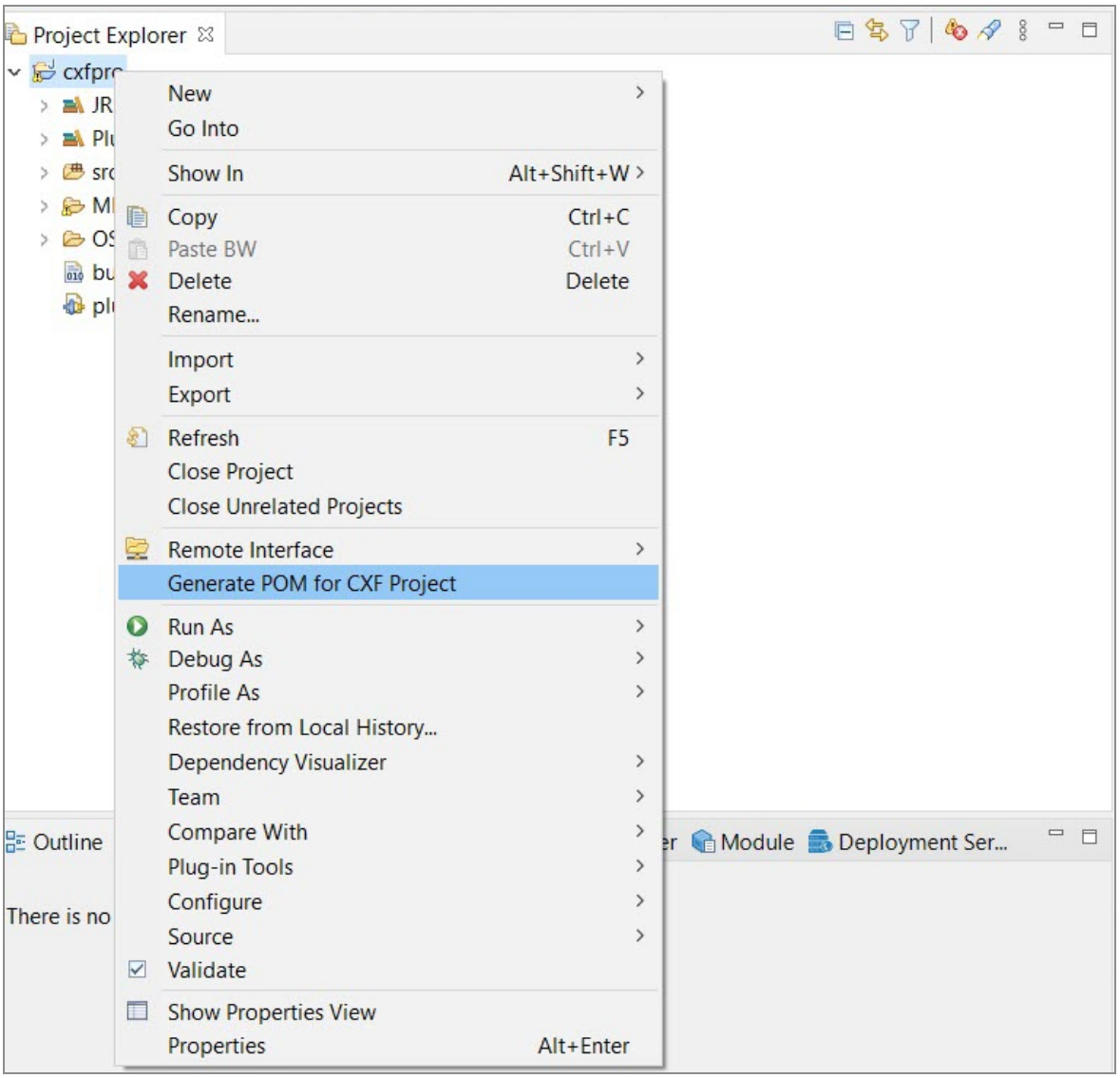
The BW application must have unit tests defined. For more information, see Unit Testing.

# Using External Custom XPath Function with TIBCO ActiveMatrix BusinessWorks Plug-in for Maven

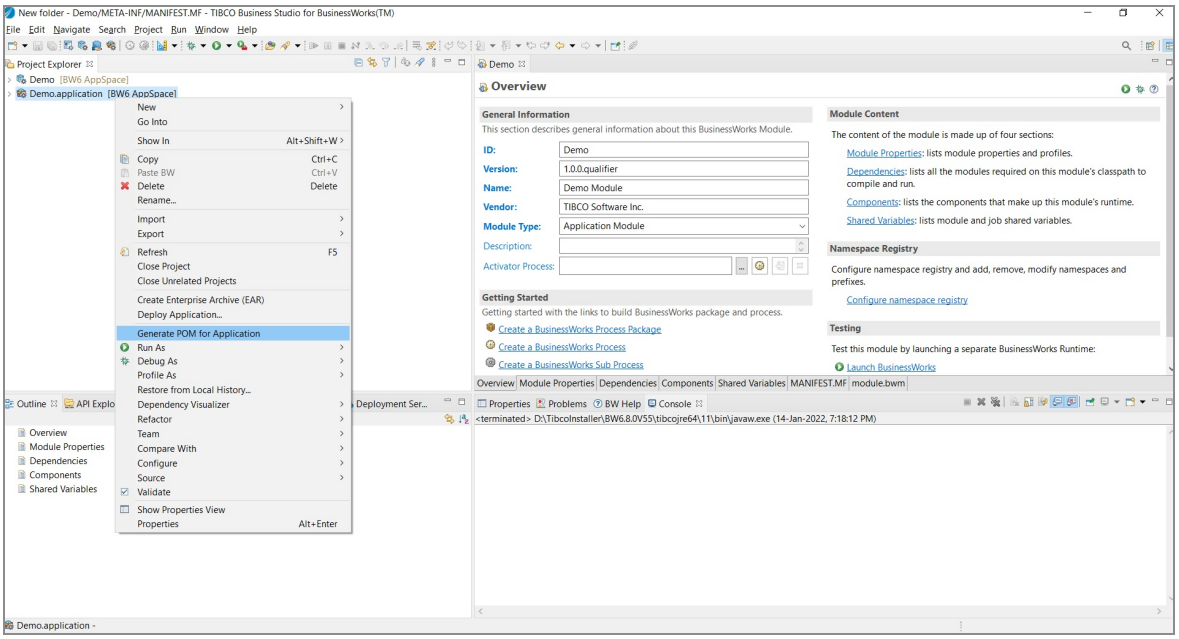
To use an external custom XPath function with TIBCO BusinessWorks Container Edition, perform the following steps:

### Procedure

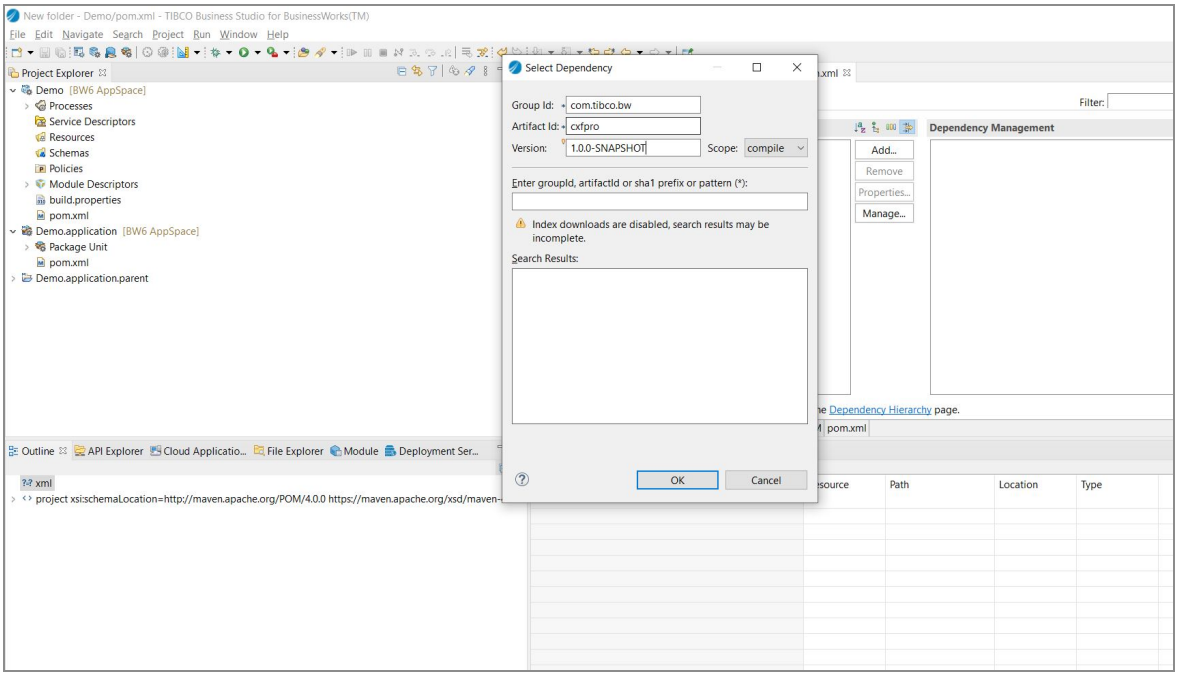
1. Create a custom XPath function project with TIBCO BusinessWorks Container Edition. For more information, see "Creating Custom XPath Functions" in *TIBCO BusinessWorks Container Edition Bindings and Palette Reference.*
2. Right-click on the created project and select **Generate POM for CXF Project**. This mavenizes the project and generates a `pom.xml` for the custom XPath function project.



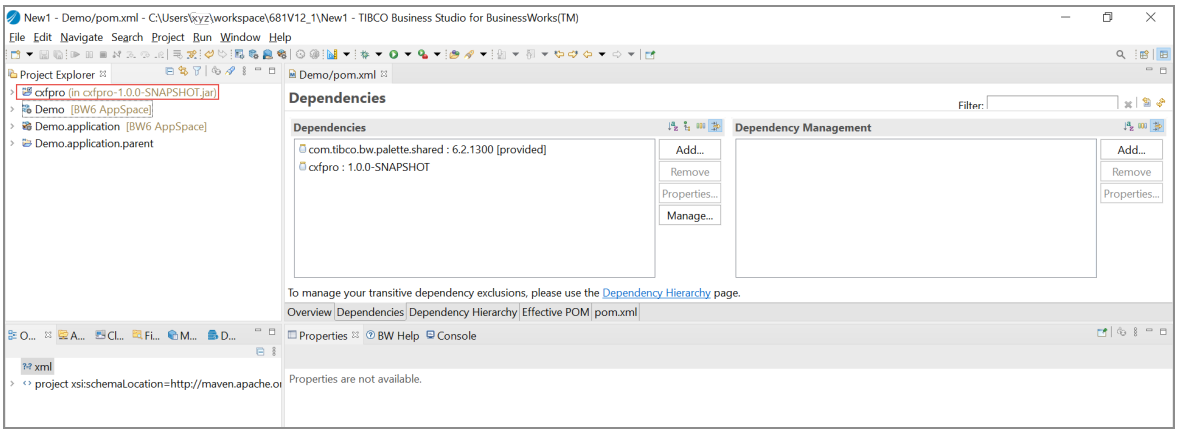

3. Create a Run or Debug Configuration for Maven. Select the custom Xpath function project and run the `clean install` goal. This installs the custom XPath function project in the local `.m2` repository.
4. Open a new eclipse workspace. Create an application project and to generate the POM for the application, right-click the application and select **Generate POM for Application**.




5. Open the Application Module `pom.xml` and add the custom XPath function project dependency, which is present in the local Maven repository, and save the `pom.xml`.




The CXF project is displayed in the Project Explorer.




> **Note:** The icon changes for the custom XPath function project indicating the project is referenced and is not in the workspace.

6. To start using the custom functions in the BW project, right-click the CXF project and select the **Install CXF Project** option.
7. The BW application must have unit tests defined. For more information, see Unit Testing.
8. Create a maven run configuration. Select the BW application parent project as the base directory.
9. Provide the maven goal `clean Test`.
10. To generate the EAR, provide the Maven goal `clean package`.

# Using Shared Modules with TIBCO ActiveMatrix BusinessWorks Plug-in for Maven

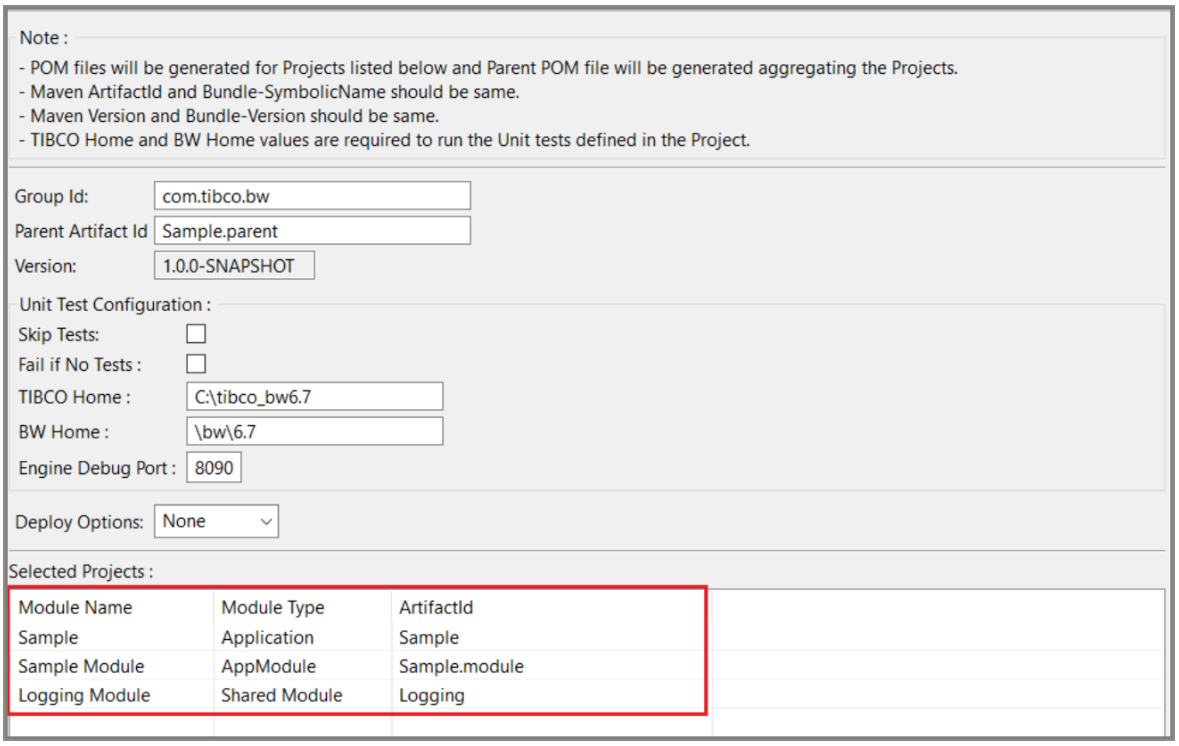
To use shared modules with the TIBCO ActiveMatrix BusinessWorks Plug-in for Maven, perform the following steps:

### Before you begin

Ensure that a shared module project is created in the workspace.

### Procedure

1. Create an application project **Sample** and a shared module **Logging** and refer the subprocess from the shared module in the application module.
2. To generate the POM files, right-click on the project and select **Generate POM for application**. The wizard lists down all the shared modules referenced by the application.




The parent `pom.xml` project lists down all the below modules:

```
<modules>
	<module>../Logging</module>
	<module>../Sample.module</module>
	<module>../Sample</module>
</modules>
```

If a new shared module is added after mavenizing the project, open the **Generate POM for application** wizard again to regenerate the `pom.xml` files.

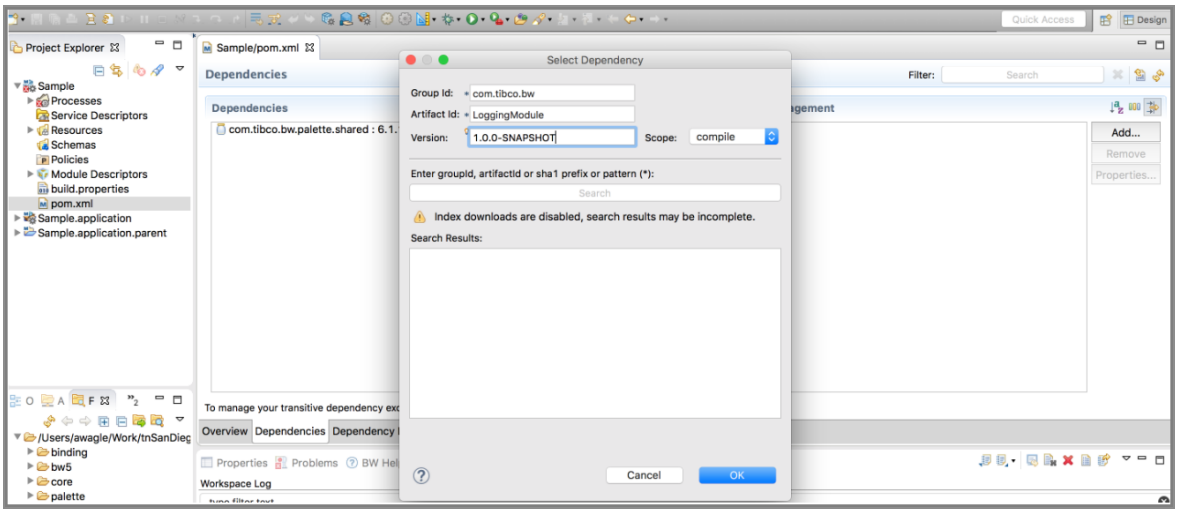
# Using External Shared Modules with TIBCO ActiveMatrix BusinessWorks Plug-in for Maven

**Note:** The following steps are also applicable for using Binary Shared Modules with ActiveMatrix BusinessWorks Plug-in for Maven.

To use external shared modules with ActiveMatrix BusinessWorks, perform the following steps:

### Procedure

1. Create a shared module project with ActiveMatrix BusinessWorks.
2. Right-click the created project and select **Generate POM for Shared Module**. This mavenizes the project and generates a `pom.xml` for the shared module.
3. Create a Run or Debug Configuration for Maven. Select the shared module project and run the `clean install` goal. This installs the shared module project in the local `.m2` repository.
4. Open a new eclipse workspace. Create an application project and to generate the POM for the application, right-click the application and select **Generate POM for Application**.
5. Open the Application Module `pom.xml` and add the shared module dependency, which is present in the local Maven repository, and save the `pom.xml`.




The shared module project is displayed in the Project Explorer.

> **Note:** The icon changes for the shared module project indicating the project is referenced and is not in the workspace.

# Running Test Cases from an External Shared Module

To run test cases from an external shared module, perform the following steps:

1. Right-click the parent project in which the module is added as a POM dependency and select **Run As > Maven Build**.
2. In the Edit Configuration wizard, configure the goals to achieve the following scenarios:

| Scenarios | Using Studio | Using Command Line |
|---|---|---|
| Run Test Cases from the External Shared Module that are added as | test -DrunESMTest=true | mvn test -DrunESMTest=true |

| Scenarios | Using Studio | Using Command Line |
|---|---|---|
| POM dependencies | | |
| Run Test Suites from the External Shared Module | test -DrunESMTest=true -DESMtestSuiteName=”testSuite1.bwts” | mvn test -DrunESMTest=true-DESMtestSuiteName="testSuite1.bwts" |
| Run Multiple Test Suites from the External Shared Module | test -DrunESMTest=true -DESMtestSuiteName=”testSuite1.bwts;testSuite2.bwts” | mvn test -DrunESMTest=true-DESMtestSuiteName="testSuite1.bwts;testSuite2.bwts" |
| Generate BusinessWorks Coverage report | site -DrunESMTest= true | mvn site -DrunESMTest=true |

**Limitation**

If the Test Case from the External Shared Module fails while using the above mentioned method, you should import the Shared Module Project into the Workspace. Then change the fault data and publish it in the `.m2` repository and run the test case again.
It is expected that you should test the External Shared Module while it is being developed, by creating a dummy application and adding the shared module dependency in it. Run the test goal on the parent project that can run the test cases from the Shared Module as well.

# Dependency Exclusions

To exclude dependencies that are not used in the project, Maven supports dependency exclusions. You can set exclusions on a specific dependency in your POM file. For example,

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>com.tibco.bw</groupId>
    <artifactId>MainApplication.parent</artifactId>
    <version>1.0.0-SNAPSHOT</version>
    <relativePath>../MainApplication.parent</relativePath>
  </parent>
  <artifactId>MainApplication.module</artifactId>
  <packaging>bwmodule</packaging>
  <dependencies>
    <dependency>
      <groupId>com.tibco.plugins</groupId>
      <artifactId>com.tibco.bw.palette.shared</artifactId>
      <version>6.2.1300</version>
      <scope>provided</scope>
    </dependency>
    <dependency>
      <groupId>com.tibco.bw</groupId>
      <artifactId>ProjectB</artifactId>
      <version>1.0.0</version>
      <exclusions>
        <exclusion>
          <groupId>com.tibco.bw</groupId>
          <!--  Exclude Project-D from Project-B  -->
          <artifactId>ProjectD</artifactId>
        </exclusion>
      </exclusions>
    </dependency>
    <dependency>
      <groupId>com.tibco.bw</groupId>
      <artifactId>ProjectC</artifactId>
      <version>1.0.0</version>
    </dependency>
  </dependencies>
  <build>
    <sourceDirectory>src</sourceDirectory>
    <outputDirectory>target/classes</outputDirectory>
    <plugins>
      <plugin>
        <groupId>com.tibco.plugins</groupId>
        <artifactId>bw6-maven-plugin</artifactId>
        <version>2.9.4</version>
        <extensions>true</extensions>
      </plugin>
    </plugins>
  </build>
</project>
```

**Note:**

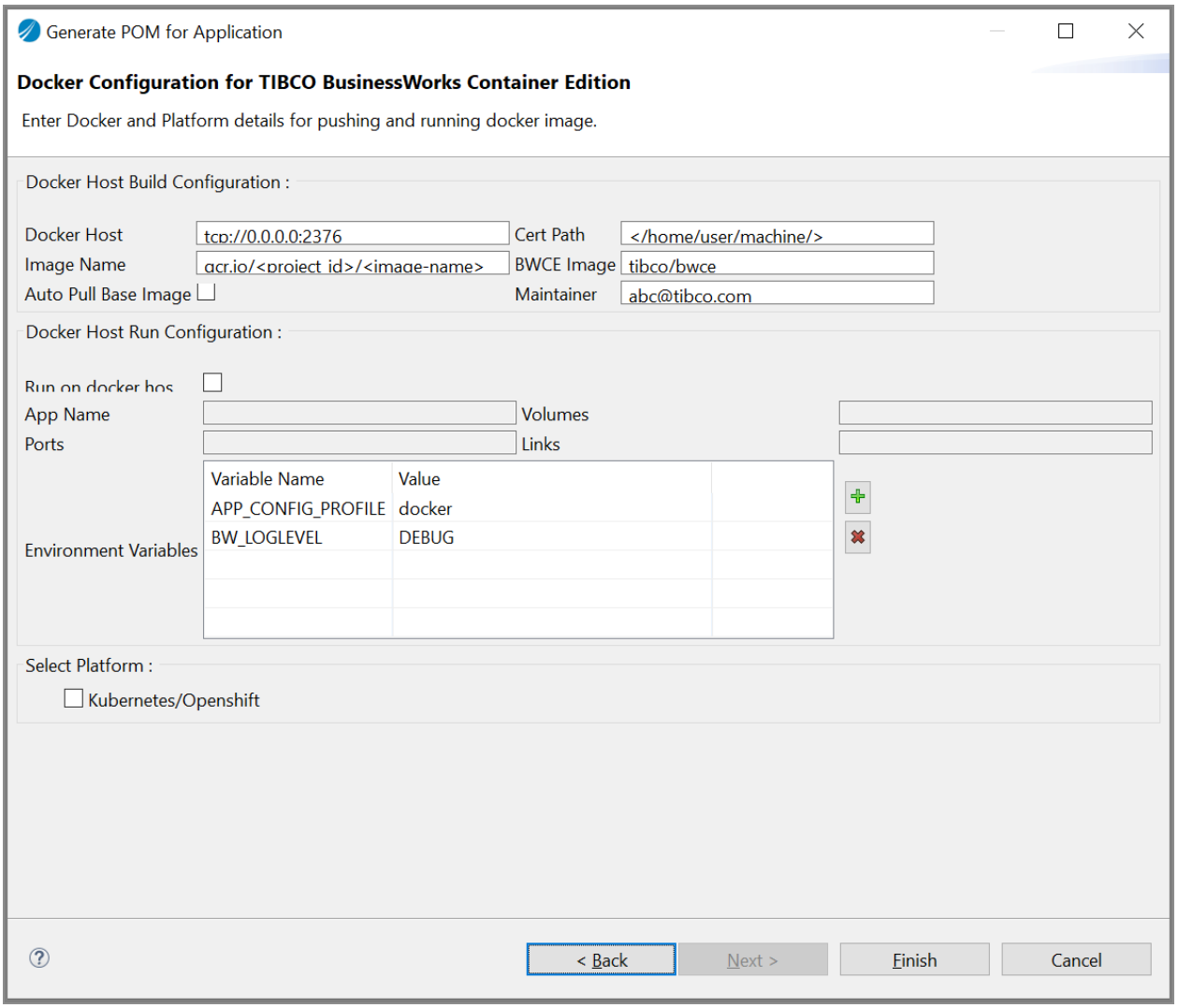
- Dependency exclusions are to be used when you generate an EAR using the Maven install or package goals.
- Support for dependency exclusions is provided on the top-level POM.

# Building Applications for TIBCO BusinessWorks Container Edition in Docker and Kubernetes or Openshift

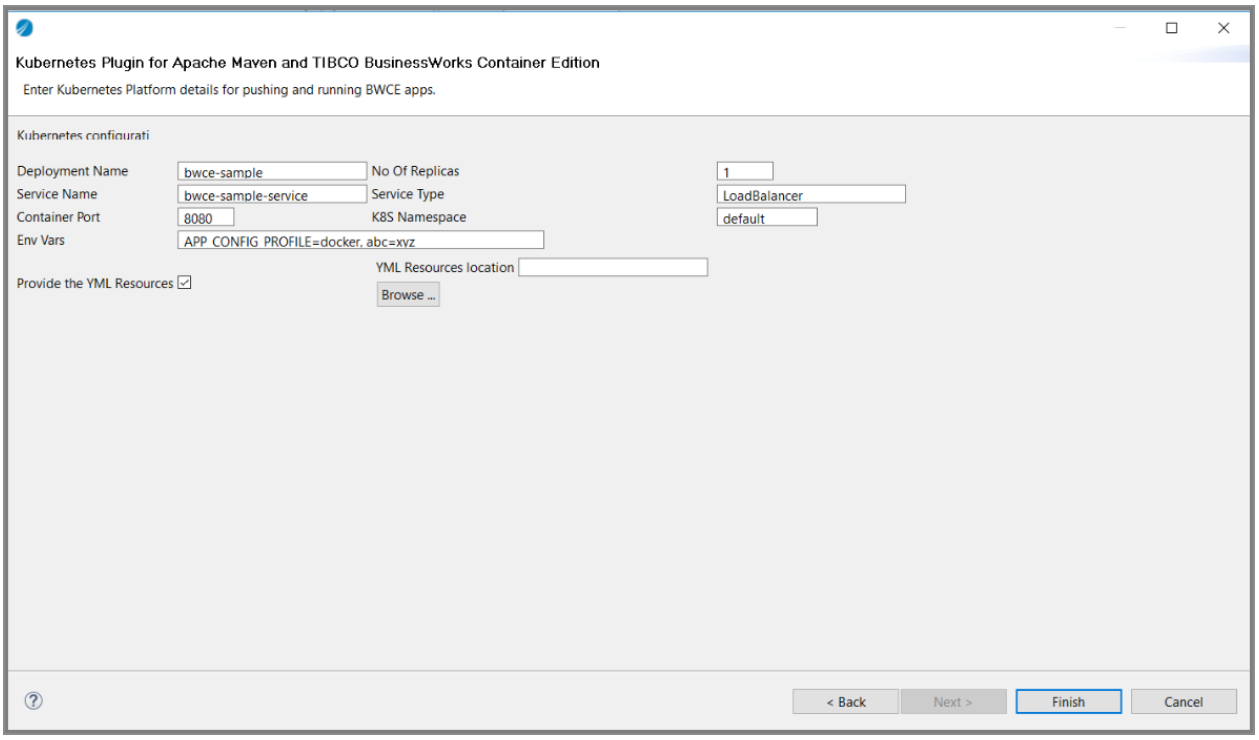
Starting from version 2.0.0 of the TIBCO ActiveMatrix BusinessWorks Plug-in for Maven, the Fabric8 Maven Plug-in used for the maven build process for Docker and Kubernetes or Openhsift is upgraded from version 2.2.102 to version 3.5.41. The Docker Maven Plug-in used with Fabric8 is upgraded from version 0.14.2 to version 0.26.1. The upgraded versions of these plug-ins allow you to write partial Kubernetes or Openshift yaml files so deployments or services or HPA are predefined.

### Before you begin

If the Docker platform is selected as deployment option while generating the POM file, the following window is displayed.




If the Kubernetes platform is selected as deployment option while generating the POM file, the following window is displayed:




Configure the options as required for your project.

- For Docker: The `docker-dev.properties` and `docker-prod.properties` file and the variable name as `docker.property.file`
- For Kubernetes: The `k8s-dev.properties` and `k8s-prod.properties` file and the variable name as `k8s.property.file`
- For more environments, you can manually create copies of one of these properties file and rename it to your environment. For example, docker-qa.properties, k8s-qa.properties, and so on. By default, both dev and prod properties are same, and contains values which are specified from TIBCO Business Studio for BusinessWorks pop-up, so you can manually edit these values in properties file for your environment (prod or dev).

To mavenize the build process for Docker and Kubernetes, perform the following steps:

### Procedure

1. When generating the POM file, provide the deployment or service or HPA YAML files,

select the **Provide the YML Resources** checkbox and provide the location of the stored files of your local machine.

2. Provide the maven goal `clean package initialize docker:build`.
   During docker build, tag the images for kubernetes as `gcr.io/project-name/image-name` and `REGISTRY-SERVICE-IP:5000/project-name/image-name` for Openshift. Also, for Openshift, remove the backward slashes in the `bwdocker.host` property in docker properties file, for the IP address to be correctly parsed.
3. Provide Maven goal `initialize docker:start`.
4. Provide Maven goal `initialize docker:push`.

For Openshift, execute the `docker push REGISTRY-SERVICE-IP:5000/project-name/image-name` command from command line at the location of the maven application project.

Before push, ensure you generate token and authorize your docker host for GCP repo for Kubernetes or a corresponding repository for your Openshift environment.

Follow the steps described below for **DOCKER AUTHORIZATION BEFORE TRYING DOCKER:PUSH** for authorization and docker login.

- `com.tibco.plugins:bw6-maven-plugin:bwfabric8json`
- `initialize fabric8:resource`
- `initialize fabric8:apply`

The commands above can be chained together to execute all the goals in a single step, in the following manner :

```
clean package initialize docker:build docker:push com.tibco.plugins:bw6-maven-plugin:bwfabric8json fabric8:resource fabric8:apply
```

### DOCKER AUTHORIZATION BEFORE TRYING DOCKER:PUSH

**For Kubernetes**

--- Windows --- gcloud auth print-access-token docker login -u _token -p "your token" https://gcr.io

--- Linux/OSX ---- docker login -u _token -p "$(gcloud auth print-access-token)" https://gcr.io

**For Openshift**

--- Windows --- docker login -u (oc whoami -t) REGISTRY-SERVICE-IP:5000

--- Linux/OSX ---- docker login -u oc whoami -p oc whoami -t REGISTRY-SERVICE-IP:5000

## Using OpenJDK for deploying applications using Maven in a Docker container

### Procedure

1. In the user home directory, create the `customJavaPath.properties` file and give the OpenJDK path as a value for the `customJAVAPath` property. For example, `customJavaPath=/usr/lib/jvm/java-11-openjdk-amd64/bin/java`.
2. Create the bwce base docker image using the docker file:

```
FROM eclipse-temurin:11-alpine
LABEL maintainer="TIBCO Software Inc."
ADD . /
RUN chmod 755 /scripts/*.sh && apk update && apk add unzip openssh
net-tools && apk add --no-cache bash
ENTRYPOINT ["/scripts/start.sh"]
```

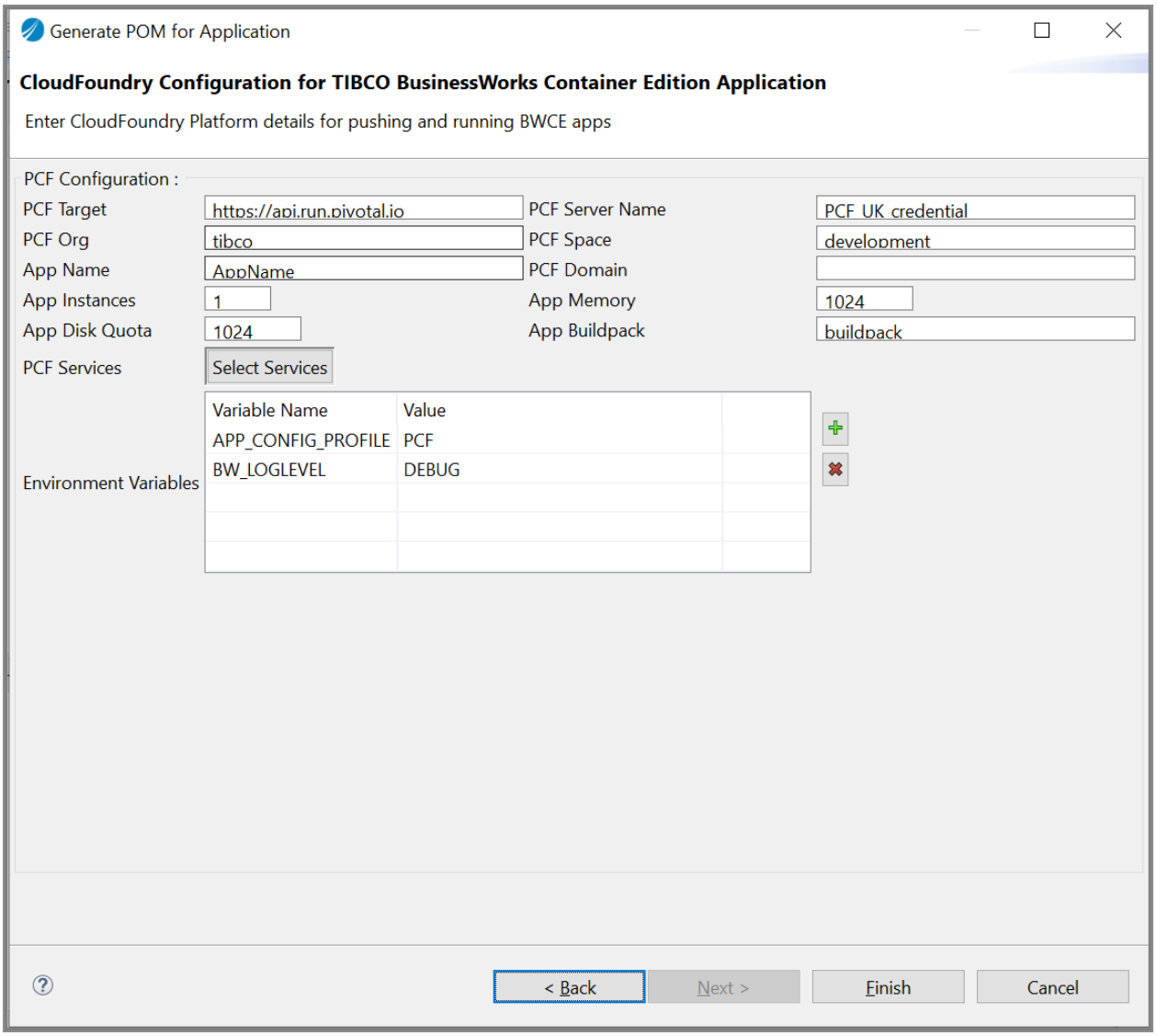
> **Note:** To use OpenJDK11 and remove TIBCO JRE from eclipse-temurin:11-jre-alpine based container image, navigate to https://github.com/TIBCOSoftware/bwce-docker/tree/openjdk-alpine and clone the 'eclipse-alpine' repository to your local machine. For more information to create the base Docker image for Linux container, see "Creating the TIBCO BusinessWorks Container Edition Base Docker Image for Linux Containers" in the *TIBCO BusinessWorks Container Edition Application Developement*.

3. Create an application image using the above bwce base docker image.
4. Deploy the application on Docker.

# Building Applications for TIBCO BusinessWorks Container Edition in Cloud Foundry

To build applications with the TIBCO BusinessWorks Container Edition in Cloud Foundry, perform the following steps:

If the Cloud Foundry platform is selected as a deployment option during POM generation, the following window is displayed.




Specify the following guidelines for some of the Cloud Foundry specific fields:

1. **PCF Server Name**: The PCF_UK_credential (Define this as `<\server>` inside your `config/settings.xml` of Apache Maven Home)
   PCF_UK_credential admin xxxxxxxxxxx.
2. **App Memory**: Minimum should be 1024 MB.
3. **App Buildpack**: BWCE build pack, which the developer has pushed to Cloud Foundry instance.
4. **PCF Services**: Click this button to sign in to Cloud Foundry and select the required services you want to bind to your application.

### PROPERTIES FILES

- By default, the **pcfdev.properties** and **pcfprod.properties** files are generated with a variable name as `pcf.property.file`.
- For more environments, one of these properties file can be copied and renamed to the other environment. For example, `pcfqa.properties`. Its values can be customized for the required environment.

### GOALS FOR CLOUD FOUNDRY

- Run the maven build goal `cf:push`, `cf:scale` and so on.
- While running the goals, provide the credentials using system arguments, `-Dcf.username` and `-Dcf.password`.
- You can try other goals from TIBCO Business Studio for BusinessWorks, by creating new maven run configurations for different goals, or from the terminal pointing to the workspace using the `mvn initialize cf:command -Dpcf.property.file=pcfdev.properties` command.

**Note:** For all non-web application, if you are using Cloud Foundry Elastic Runtime 1.6 or above then, a health-check error occurs while `cf:push`, so you have to use Cloud Foundry CLI (6.13 or above) to set the health-check as none. After pushing the application, repush the application after setting the health-check as none. You can use `- cf set-health-check App_Name none` command on the command line.

# Maven Goals

The Maven plug-in in TIBCO BusinessWorks Container Edition simplifies the build process and enhances project management by providing a structured approach. It helps streamline the process of building, managing, and deploying software projects by automating tasks such as compiling source code, managing project dependencies, and creating distributable artifacts.

*Maven goals*

| Lifecycle Phases | Description |
|---|---|
| `clean` | Removes all files generated by the previous build ex-target folder. |
| `generate-sources` | Generates any source code for inclusion in compilation. |
| `install` | Installs the package into the local repository, for using as a dependency in other projects locally.<br><br>**Note:** When you configure the POM file to be deployed on Docker, the package is deployed on the respective container. |
| `test` | Tests the compiled source code using a suitable unit testing framework. These tests do not require the code to be packaged or deployed. |
| `site` | To generate a report (`target > site > bwcoverage.html/bwtest.html`). |
| `validate` | To validate whether a project is correct, and all the necessary information is available. |
| `package` | Takes the compiled code and packages it in its distributable format, such as a JAR. The EAR is generated in the same workspace. |
| `compile` | Compiles the source code of the project. |

| Lifecycle Phases | Description |
|---|---|
| `verify` | Runs the checks if any on the results of integration tests to ensure that quality criteria are met. |
| `deploy` | Run in the build environment. It copies the final package to the remote repository for sharing with other developers and projects. |

The default Maven lifecycle consists of multiple phases. Some of them are mentioned in the above table that runs in a sequential order to complete the project build process.

Considering the lifecycle phases above, the Maven plug-in performs the following steps when a default lifecycle is used:

1. Maven validates the project first.
2. Tries to compile the sources.
3. Runs those against the tests.
4. Packages the binaries (for example, `jar/ear`).
5. Runs integration tests against that package.
6. Verifies the integration tests.
7. Installs the verified package to the local repository.
8. Deploys the installed package to a remote repository.

Here, the `Maven > install` command follows the default lifecycle.

## Maven Plug-in Properties

| Property | Description | Values |
|---|---|---|
| `disableMocking` | To disable mocking for all mock | • `true`: Disables mocking for all mocked activities of the BusinessWorks application.<br>• `false`: Enables mocking for all mocked activities of the BusinessWorks application. |

| Property | Description | Values |
|---|---|---|
| | activities of the BusinessWorks application. | This property can be used along with "test" and "site" goals.<br>Example:<br>`mvn test -DdisableMocking=true` |
| `disableAssertions` | To disable assertions added for all activities of the BusinessWorks application. | • `true`: Assertions cannot run for all activities that are under unit testing in the BusinessWorks application.<br>• `false`: Assertions are run for all activities that are under unit testing in the BusinessWorks application.<br>This property can be used along with "test" and "site" goals.<br>Example:<br>`mvn test -DdisableAssertions=true` |
| `showFailureDetails` | To show provided input and Gold input in case of test failure. | • `true`: It shows the failure details in the console logs and TIBCO Business Studio for BusinessWorks execution reports for the activities that are under unit testing in the TIBCO Business Studio for BusinessWorks application.<br>• `false`: It does not show the failure details in the console logs and TIBCO Business Studio for BusinessWorks execution report for the activities that are under unit testing in the TIBCO Business Studio for BusinessWorks application.<br>This property can be used along with `test` and `site` |

| Property | Description | Values |
|---|---|---|
| | | goals. It is set to `true` by default from TIBCO BusinessWorks Maven Plug-in 2.7.1 and above.<br>Examples:<br>`mvn test -DshowFailureDetails=true`<br>`mvn site -DshowFailureDetails=true` |
| `testSuiteName` | To run the Test suite. Provide the test suite name as a value to the property while running the "test" goal. | This property can be used along with `test` and `site` goals.<br>Example:<br>`mvn test -DtestSuiteName=ActivityAssertionTestSuite.bwts`<br>You can also run multiple test suites in sequence by providing the test suite names separated by a semicolon ";".<br>`mvn test -DtestSuiteName=ActivityAssertionTestSuite.bwts;FaultTestSuite.bwts` |
| `customArgEngine` | To pass the custom argument for the property file when starting | This property supports Absolute path, Relative path, and URL-based file path.<br>**Note:** In case of relative path, you must keep the "properties" file in the Application Project.<br>Example: |

| Property | Description | Values |
| --- | --- | --- |
| | the BWEngine, create a `.properties` file that has the list of custom arguments in the form of `-Dkey=value`. The path of the same `.properties` file must be passed to the `customArgEngine` property. | `mvn test -DcustomArgEngine="D:\Issues\customArgEngine\sample.properties"`<br>Where, the `sample.properties` file has the list of custom arguments |
| `skipInitMainProcessActivities` | To skip `init` for all main process activities. | • `true`: It skips `init` for all main process activities.<br>• `false`: It initiates the main process activities.<br>Examples:<br>`mvn test -DskipInitMainProcessActivities=true`<br>`mvn site -DskipInitMainProcessActivities=false` |

| Property | Description | Values |
|---|---|---|
| `skipInitAllNonTestProcessActivities` | To skip `init` for all non-unit test process activities. | • `true`: It skips `init` for all non-unit test process activities.<br>• `false`: It initiates all non-unit test process activities.<br>Examples:<br>`mvn test -DskipInitAllNonTestProcessActivities=true`<br>`mvn site -DskipInitAllNonTestProcessActivities=false` |
| `startOnDeploy` | To restrict an application to auto-start after deployment. By default the value of `startOnDeploy` is "true". | • `true`: It auto-starts the application after deployment.<br>• `false`: It does not auto-start the application after deployment.<br>Example:<br>`mvn install -DstartOnDeploy=false` |
| `independentComponentStartup` | This property runs the BWEngine independently even if there are | • `true`: The BWEngine starts independently.<br>• `false`: The BWEngine does not start independently.<br>Example:<br>`mvn test -` |

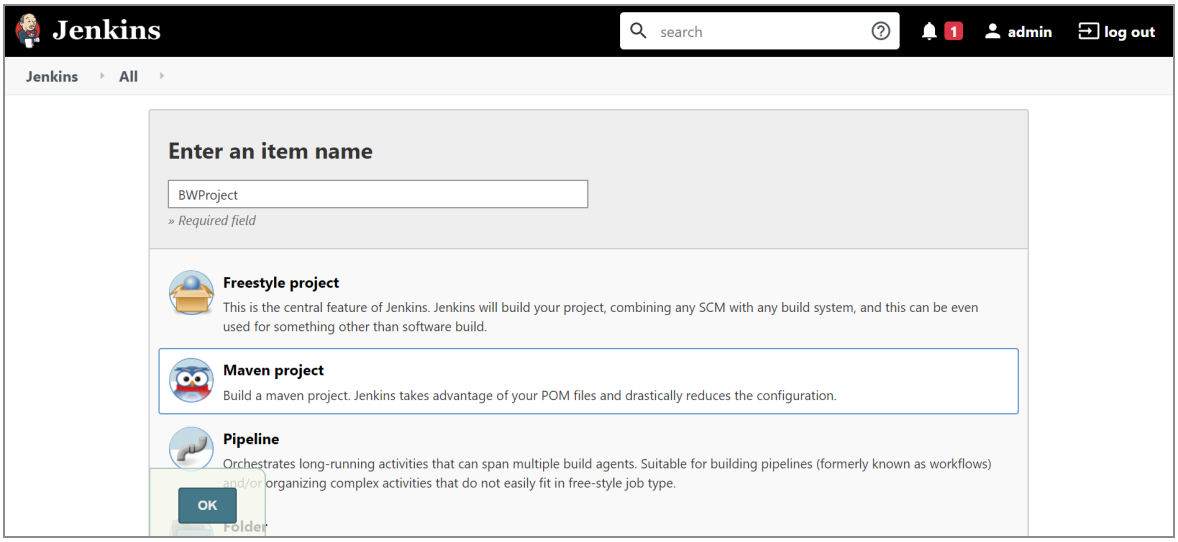
| Property | Description | Values |
| --- | --- | --- |
| | errors in an unused shared resource.<br><br>**Note:** This property gives errors for the reconfigured shared resources and runs the maven goal successfully. | `DindependentComponentStartup=true`<br><br>`mvn test -DindependentComponentStartup=false`<br><br>`mvn test -DindependentComponentStartup=true -DskipInitAllNonTestProcessActivities=true -DskipInitMainProcessActivities=true` |

# Running Continuous Integration/Continuous Deployment (CI/CD) using Jenkins

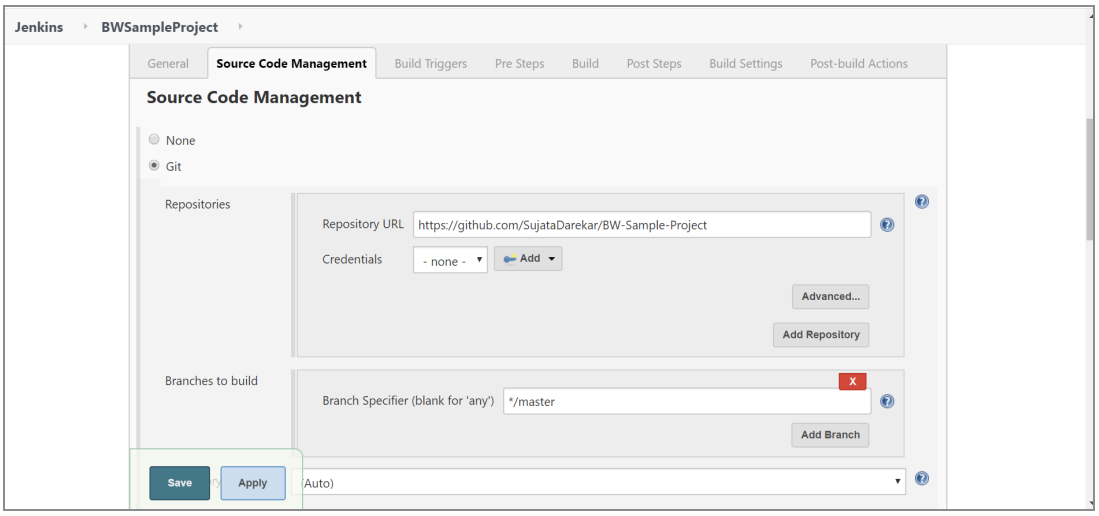
To run CI/CD using Jenkins, perform the following steps:

### Procedure

1. On the Jenkins Dashboard, go to **Manage Jenkins > Manage plugins > Available** and download the following plug-ins:
    - Maven Integration
    - Git Plug-in
2. On the Jenkins Dashboard, click **New Item** and add a name for the Maven Project, for example `BWSampleProject`. Then click **OK**.

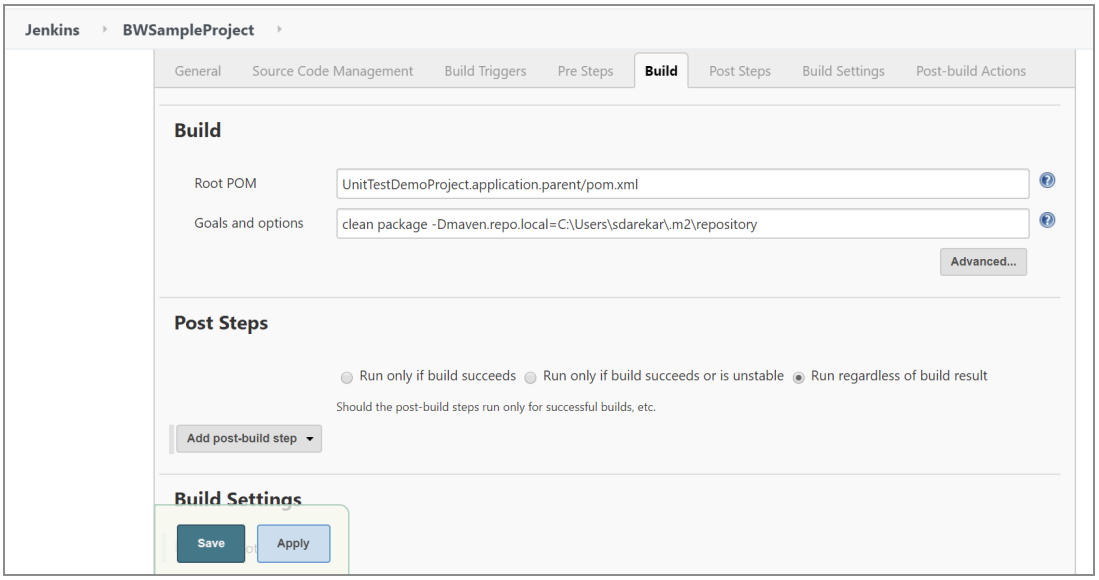



3. On the Configure page, set the following attributes:
    a. On the **General** tab, add a description for the project if needed.
    b. On the **Source Code Management** tab, select **Git** and add the GitHub repository URL where the project is present.




c. On the **Build** tab, provide the value in the **Root POM** field, then provide the Maven goal in the **Goals and options** field to execute.

   You can pass the environment variables, such as, `–Dmaven.repo.local=C:\Users$username.m2\repository` so that Jenkins can refer to a `user local .m2` repository where all the dependencies are present.




4. Click **Apply** and **Save**.
5. Go to Project Window and click **Build Now**.

# Troubleshooting

This section provides information on how to solve some commonly observed issues.

| Issue Description | Cause and Resolution |
| --- | --- |
| If a `jar/artifact/plugin` is missing in the **Problems** tab after importing a project. For example, the `"com.tibco.plette.shared.jar"` jar is missing whose dependency is present in autogenerated `pom.xml` file. | **Cause:** The palette shared jar is located under `<TIBCO-HOME>\bwce\2.x\system\shared\`. Before BWCE 2.7.0 it was packaged with the Maven plug-in installer. After BWCE 2.7.0, the Maven plug-in is provided out-of-box with BW, the palette shared jar gets installed in the local `.m2` repo during product installation. You can install it again by running the script at `<TIBCO-HOME>\bwce\2.x\maven\install.bat`. Ensure the `mvn --version` command works on your machine before you run `install.bat` command.<br><br>**Resolution:** Update the project. **Right click ->Maven->Update Project** |
| When the maven project with the assertion is executed with Maven Goals, the following error occurs `Failed to execute goal com.tibco.plugins:bw6-maven-plugin:2.9.1:bwtest (default-bwtest) on project ERROR [qtp811813182-94] com.tibco.bw.thor.management.bw.tests.rest.BWUnitTestResource - null` | **Cause:** It is caused due to the missing latest maven plug-in jar at `.m2` repository.<br><br>**Workaround:**<br><br>1. Place the updated jar at `TIBCO-HOME\bwce\2.x\maven\b` |

| Issue Description | Cause and Resolution |
| --- | --- |
| | `w6-maven-plugin`location.<br>2. At the same location, update the `Install.bat` and POM file with the correct jar version (change 2.9.1 to 2.9.2 in the script) which is supposed to be used.<br>3. Run `install.bat` which updates and places the latest maven plug-in jar at `.m2` repository. |
| Intermittently assertion or mocking icons are missing on activities. | **Resoution:** Select the **Tests** tab again to see the icons. |

# 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 BusinessWorks™ Container Edition page:

- *TIBCO BusinessWorks™ Container Edition Release Notes*
- *TIBCO BusinessWorks™ Container Edition Installation*
- *TIBCO BusinessWorks™ Container Edition Application Development*
- *TIBCO BusinessWorks™ Container Edition Application Monitoring and Troubleshooting*
- *TIBCO BusinessWorks™ Container Edition Bindings and Palettes Reference*
- *TIBCO BusinessWorks™ Container Edition Concepts*
- *TIBCO BusinessWorks™ Container Edition Error Codes*
- *TIBCO BusinessWorks™ Container Edition Getting Started*
- *TIBCO BusinessWorks™ Container Edition Migration*
- *TIBCO BusinessWorks™ Container Edition Performance Benchmarking and Tuning*
- *TIBCO BusinessWorks™ Container Edition REST Implementation*
- *TIBCO BusinessWorks™ Container Edition Refactoring Best Practices*
- *TIBCO BusinessWorks™ Container Edition 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

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.cloud.com/legal.