TIBCO ActiveMatrix BusinessWorks™ Plug-in Development Kit

Developer's Guide

Version 6.3.1
June 2022

Contents

Overview

TIBCO ActiveMatrix BusinessWorks™ Plug-in Development Kit is a tool that speeds up the development of new plug-ins for TIBCO ActiveMatrix BusinessWorks™.

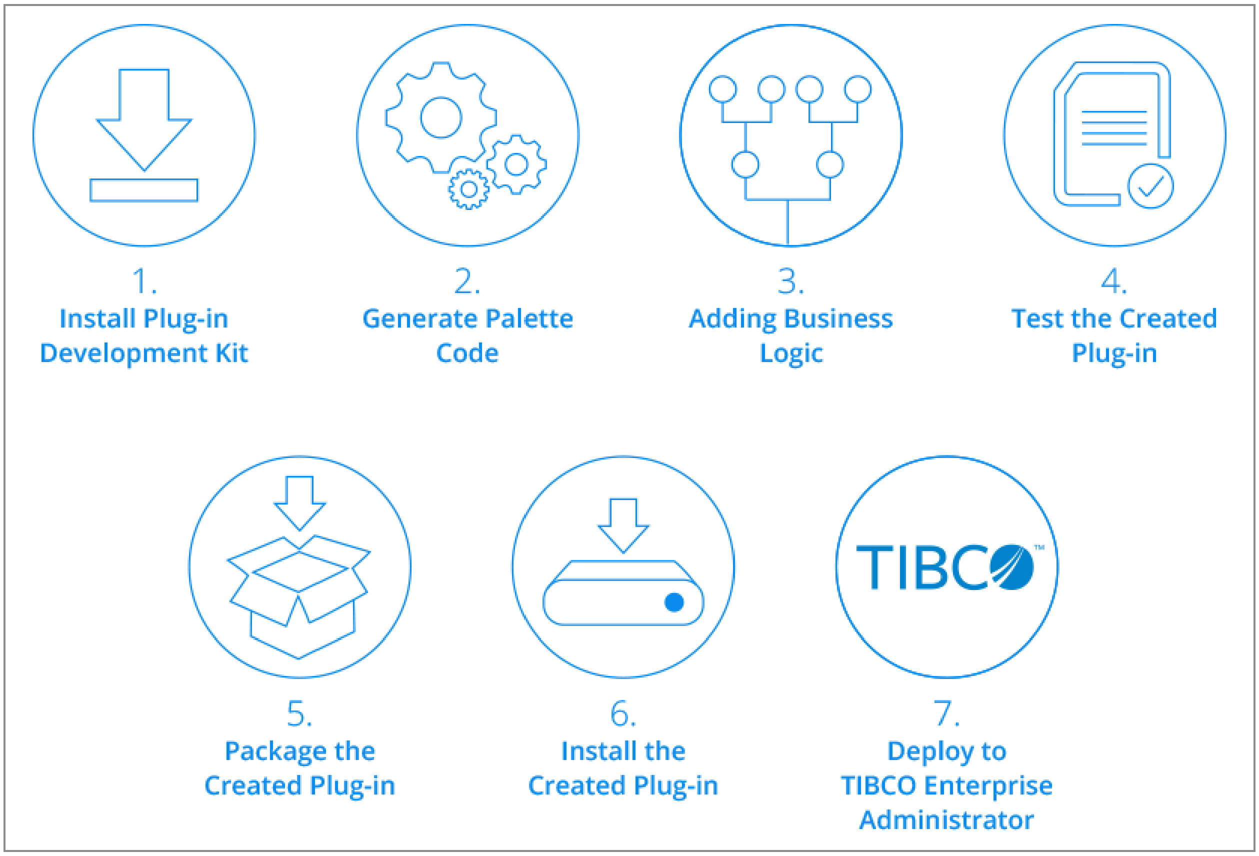
BusinessWork Plug-in Development Kit guides you through a set of wizards to create palettes and activities, and build scripts for a new plug-in. BusinessWorks Plug-in Development Kit is used to develop many of the plug-ins supported on ActiveMatrix BusinessWorks including plug-ins for Marketo, MDM, and SharePoint.

TIBCO ActiveMatrix BusinessWorks is a leading integration platform to integrate a wide variety of technologies and systems within enterprise and on cloud. ActiveMatrix BusinessWorks includes an Eclipse-based graphical user interface (GUI) based on TIBCO Business Studio™ for design, deployment, and testing of process flows. The processes designed in TIBCO Business Studio can be deployed to the BusinessWorks process engine. TIBCO ActiveMatrix BusinessWorks plug-ins extend the functions of ActiveMatrix BusinessWorks by adding more activities. A TIBCO ActiveMatrix BusinessWorks plug-in is designed to integrate third-party applications with ActiveMatrix BusinessWorks. See *TIBCO ActiveMatrix BusinessWorks Concepts* for more details about TIBCO ActiveMatrix BusinessWorks.

BusinessWorks Plug-in Development Kit works with TIBCO Business Studio to boost developers productivity by creating plug-ins that are not yet currently available for a platform. Once developed, the plug-ins created by BusinessWorks Plug-in Development Kit are installed and used exactly like the plug-in provided by TIBCO. BusinessWorks Plug-in Development Kit hides the complexity of plug-in development, generates the code to conform to BusinessWorks SDK specifications, and provides tooling to package, install, and document the plug-in.

As TIBCO is regularly developing and making many plug-ins, before you develop a new plug-in, check if TIBCO has already developed and made a plug-in available for your needs.

The following figure illustrates a complete workflow of creating and using a TIBCO ActiveMatrix BusinessWorks plug-in:


1.
Install Plug-in Development Kit
2.
Generate Palette Code
3.
Adding Business Logic
4.
Test the Created Plug-in
5.
Package the Created Plug-in
6.
Install the Created Plug-in
7.
Deploy to TIBCO Enterprise Administrator
TIBCO

BusinessWorks Plug-in Development Kit provides the following features:

- Creating a plug-in

  BusinessWorks Plug-in Development Kit provides a plug-in development wizard to guide developers in creating a plug-in step by step. During the process, you have to create a palette, add and configure activities, and add business logic.

  For more details, see Creating a Plug-in.

- Editing an existing plug-in

  BusinessWorks Plug-in Development Kit provides a plug-in editing wizard to guide developers in editing an existing plug-in that is created by using BusinessWorks Plug-in Development Kit. You can modify activity configurations, and add new activities to the plug-in.

  For more details, see Editing a Plug-in.

- Generating an installer

  BusinessWorks Plug-in Development Kit provides the functionality to package a created plug-in as a TIBCO Eclipse plug-in, which is an installation package for the

Eclipse provisioning platform.

For more details, see Creating an Installer for a Plug-in.

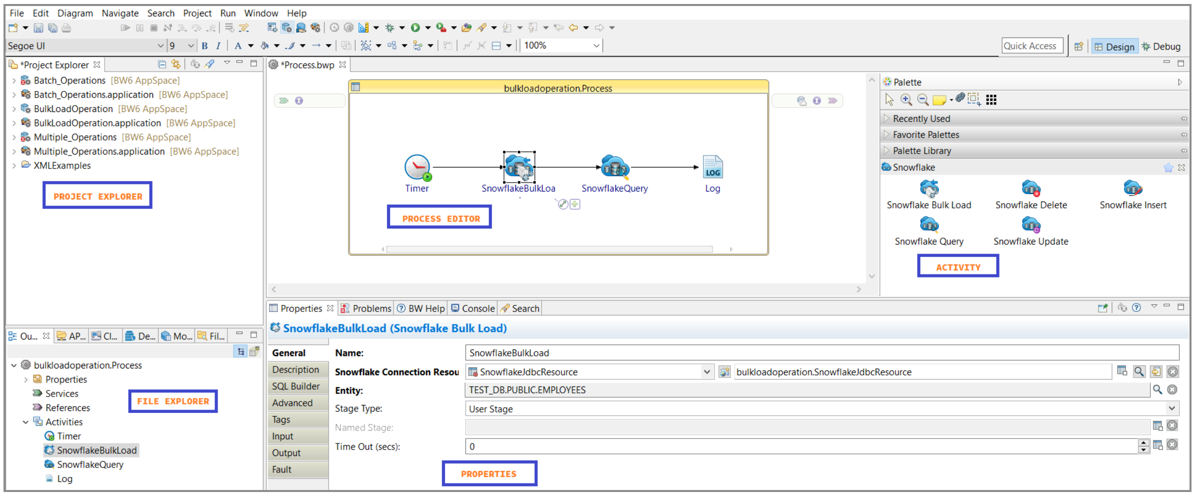
- Documentation template

  BusinessWorks Plug-in Development Kit generates a documentation template based on the plug-in that you create. You can update this template to guide your plug-in users.

  For more details, see Creating Documentation.

Overview of TIBCO Business Studio for BusinessWorks

TIBCO Business Studio for BusinessWorks is an Eclipse-based integration development environment that is used to design, develop, and test ActiveMatrix BusinessWorks applications. The studio provides a workbench in which you can create, manage, and navigate resources in your workspace. A *workspace* is the central location on your computer where all data files are stored.



SnowflakeBulkLoad (Snowflake Bulk Load)

Name: SnowflakeBulkLoad

Snowflake Connection Resou: SnowflakeJdbcResource bulkloadoperation.SnowflakeJdbcResource

Entity: TEST_DB.PUBLIC.EMPLOYEES

Stage Type: User Stage

Named Stage:

Time Out (secs): 0

The following table introduces the workbench UI elements highlighted in the image:

UI Element	Description
Menu	Contains menu items such as File, Edit, Navigate, Search, Project, Run, Window, and Help.
Toolbar	Contains buttons for frequently used commands such as: • New • Save

UI Element	Description
	- Enable/Disable Business Studio Capabilities - Create a new BusinessWorks Application Module - Debug - Run
Perspectives	Contains an initial set and layout of views that are required to perform a certain task. TIBCO Business Studio for BusinessWorks launches the Design perspective by default. Use the Design perspective when designing a process and the Debug perspective when testing and debugging a process. To change the perspective, select **Window > Open Perspective > *perspective_name*** from the main menu. Or, you can click the icon at the top right-hand side of the workbench and select the perspective to open.
Views	Lists the resources and helps you navigate within the workbench. For example, the Project Explorer view displays the ActiveMatrix BusinessWorks applications, modules, and other resources in your workspace, and the Properties view displays the properties for the selected resource. To open a view, select **Window > Show View > *view_nameview_name*** from the main menu.
Editors	Provides a canvas to configure, edit, or browse a resource. Double-click a resource in a view to open the appropriate editor for the selected resource. For example, double-click on a process `(MortgageAppConsumer.bwp)` in the Project Explorer view to open the process in the editor.
Palette	Contains a set of widgets and a palette library. A *palette* groups activities that perform similar tasks, and provides quick access to activities when configuring a process.

Getting Started

You can begin to use BusinessWorks Plug-in Development Kit by creating a HelloWorld plug-in.

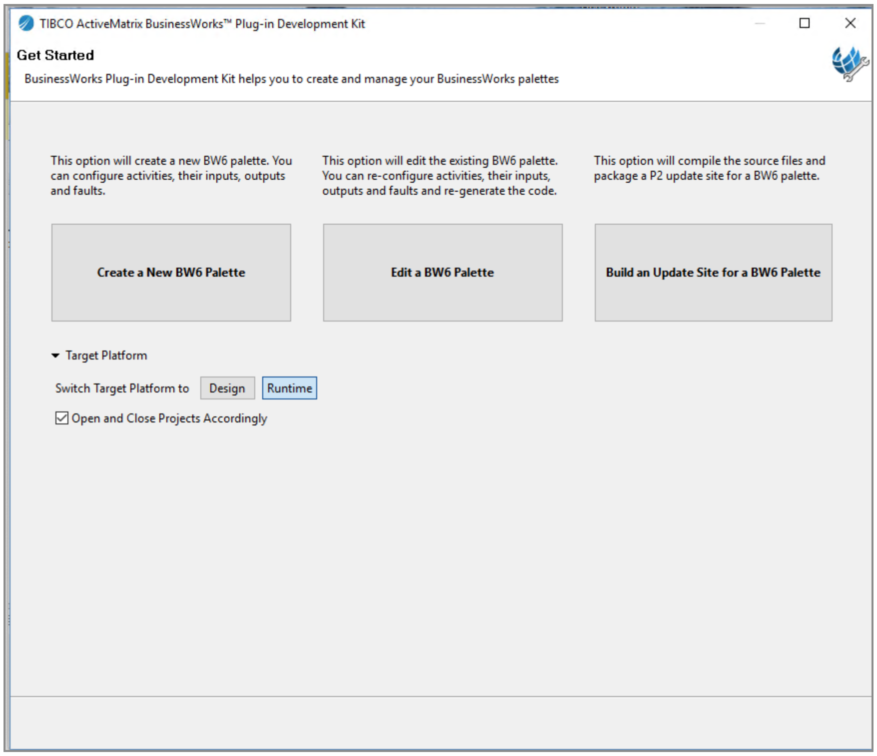
For simplicity, the HelloWorld plug-in contains only one activity, which responds to the "Hello World" string.

Defining a HelloWorld Palette

The first step in creating the HelloWorld plug-in is to create a SayHello activity in the HelloWorld palette.

Procedure

1. Open TIBCO Business Studio in one of the following ways:
    - Microsoft Windows: click **Start > All Programs > TIBCO > *TIBCO_HOME* > TIBCO Business Studio *version_number* > Studio for Designers**.
    - Linux: run the TIBCO Business Studio executable located in the `TIBCO_HOME/studio/version_number/eclipse` directory.
    - Mac OS: run the TIBCO Business Studio executable located in the `TIBCO_HOME/studio/version_number/eclipse` directory.
2. Open the BusinessWorks Plug-in Development Kit wizard in one of the following ways:
    - From the menu, click **File > New > Create a new or modify BW Plug-in Project**.
    - On the toolbar, click .
    - Press Alt+T.



TIBCO ActiveMatrix BusinessWorks™ Plug-in Development Kit

Get Started

BusinessWorks Plug-in Development Kit helps you to create and manage your BusinessWorks palettes

This option will create a new BW6 palette. You can configure activities, their inputs, outputs and faults.

This option will edit the existing BW6 palette. You can re-configure activities, their inputs, outputs and faults and re-generate the code.

This option will compile the source files and package a P2 update site for a BW6 palette.

☑ Open and Close Projects Accordingly

3. In the Get Started dialog box, click **Create New BW6 Palette** to create a palette:

   a. In the **Palette name** field, enter HelloWorld as the palette name.

   b. In the **Palette package name** field, enter com.tibco.bw as the package name.

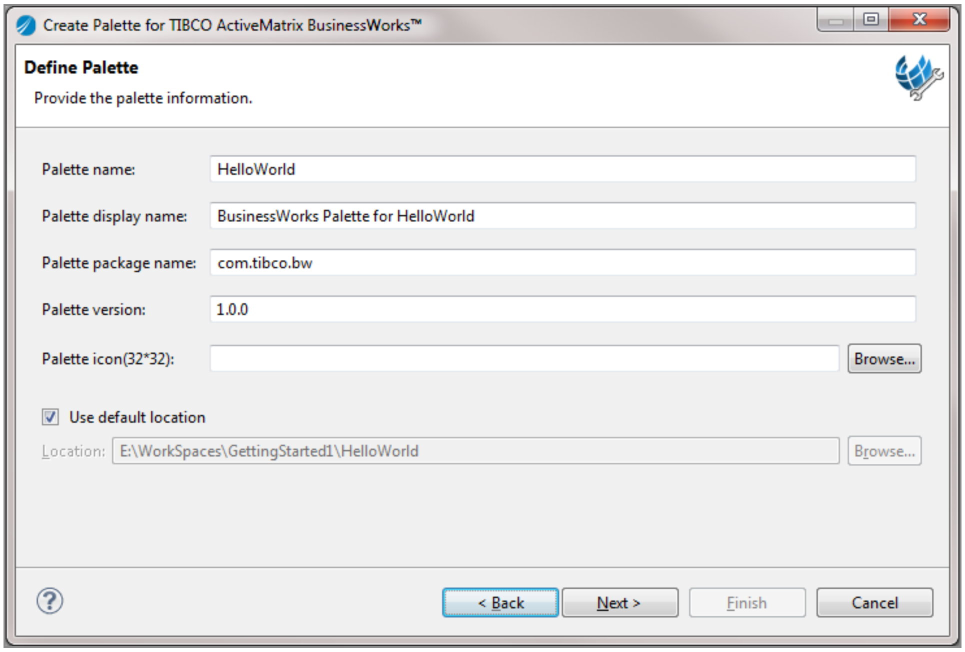
   c. In the **Palette version** field, enter the palette version that you want to use.

      By default, the palette version is 1.0.0.

   d. By default, 📷 is used as the palette icon. If you want to use another icon, click **Browse** to locate a palette icon.

      The size of the icon cannot be less than 32x32.

   e. Use the default location as the project folder. Click **Next**.



Define Palette

Provide the palette information.

Palette name: HelloWorld

Palette display name: BusinessWorks Palette for HelloWorld

Palette package name: com.tibco.bw

Palette version: 1.0.0

Palette icon(32*32): Browse...

☑ Use default location

Location: E:\WorkSpaces\GettingStarted1\HelloWorld Browse...

4. In the Define Activities, Process Starters and Signal-In dialog box, add and configure an activity:

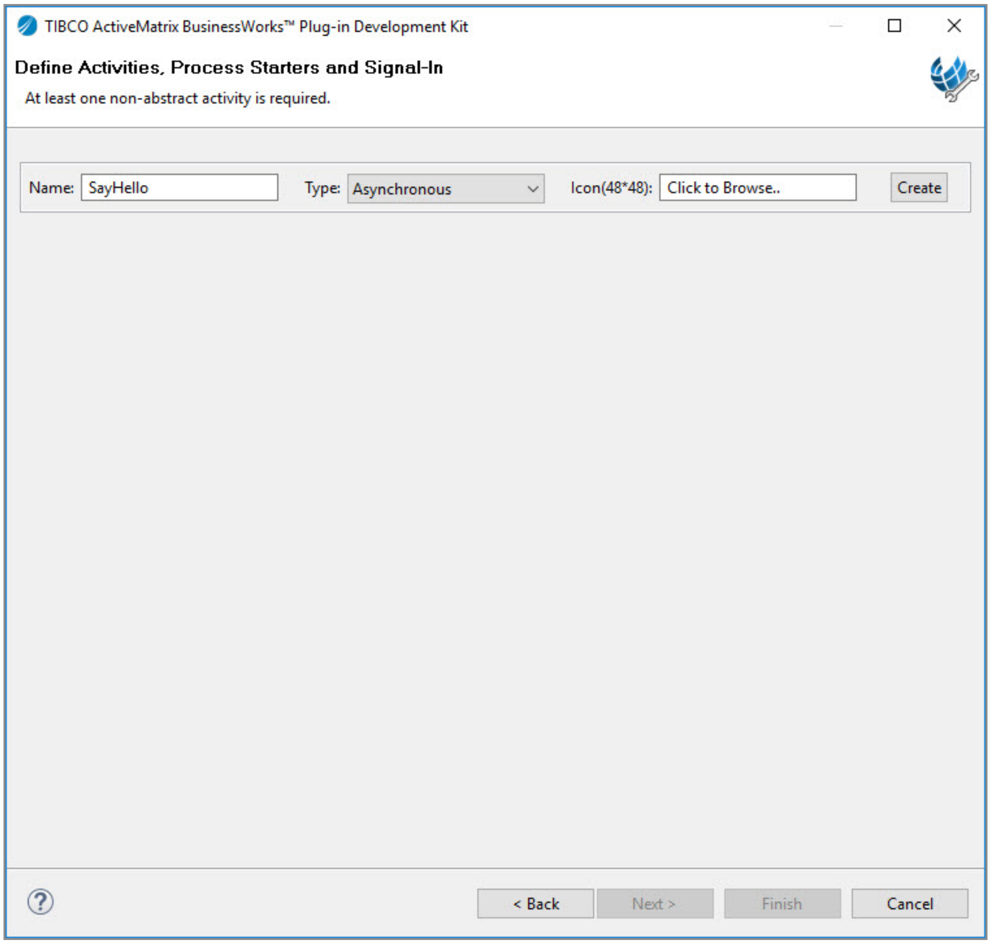
   a. In the **Name** field, enter SayHello as the activity name.

   b. From the **Type** list, select **Asynchronous** as the activity type.

   c. By default, 📷 is used as the activity icon. If you want to change the activity icon, click the text box next to the **Icon** field, and select an icon to represent the activity.

      The supported image formats are .jpg, .gif, .png, and .jpeg. The size of the icon must be 48x48 or larger. BusinessWorks Plug-in Development Kit automatically generates the icon in size of 16x16, 32x32, and 48x48.

      The size of the icon cannot be less than 48x48.



TIBCO ActiveMatrix BusinessWorks™ Plug-in Development Kit

Define Activities, Process Starters and Signal-In

At least one non-abstract activity is required.

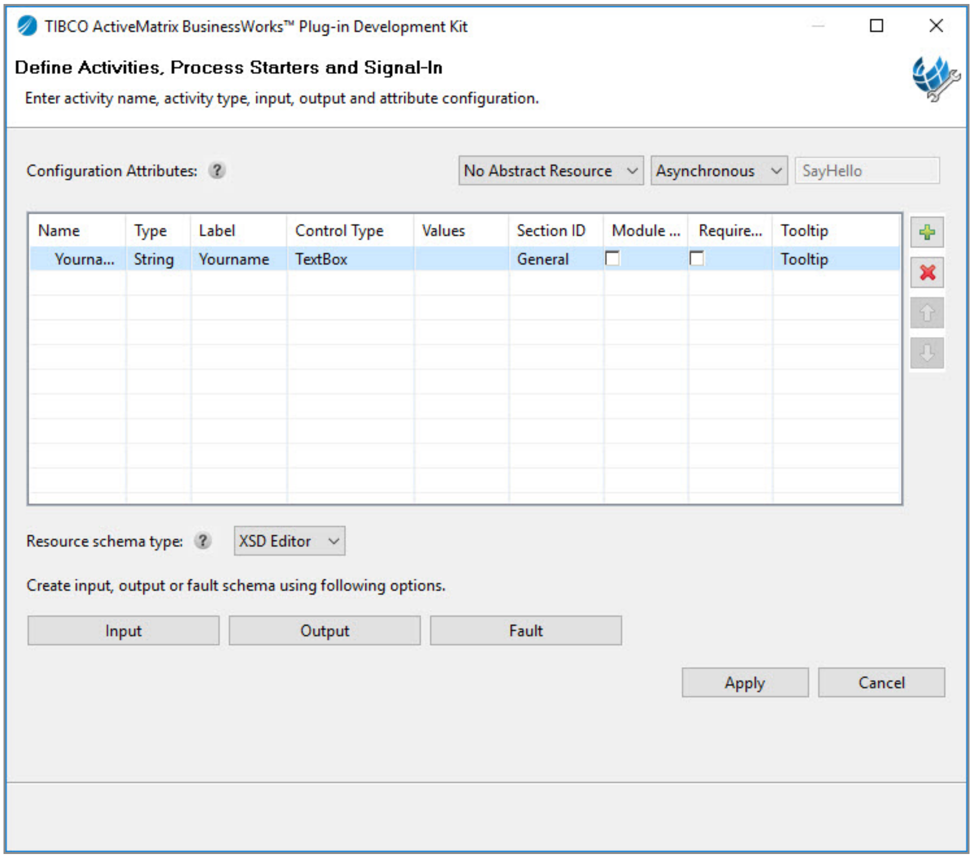
Name: SayHello

Type: Asynchronous

Icon(48*48): Click to Browse..

d. Click **Create** to configure the activity. Click ➕ to add attributes for the **SayHello** activity and configure the attribute as follows:

- Enter SayHello as the value of the **Name** attribute.
- Select String as the value of the **Type** attribute.
- Enter a name that you want to set as the value of the **Label** attribute.
- Select TextBox as the value of the **Control Type** attribute.
- Select General as the value of the **Section ID** attribute.



TIBCO ActiveMatrix BusinessWorks™ Plug-in Development Kit

Define Activities, Process Starters and Signal-In

Enter activity name, activity type, input, output and attribute configuration.

Configuration Attributes: No Abstract Resource | Asynchronous | SayHello

Name	Type	Label	Control Type	Values	Section ID	Module ...	Require...	Tooltip
Yourna...	String	Yourname	TextBox		General	□	□	Tooltip

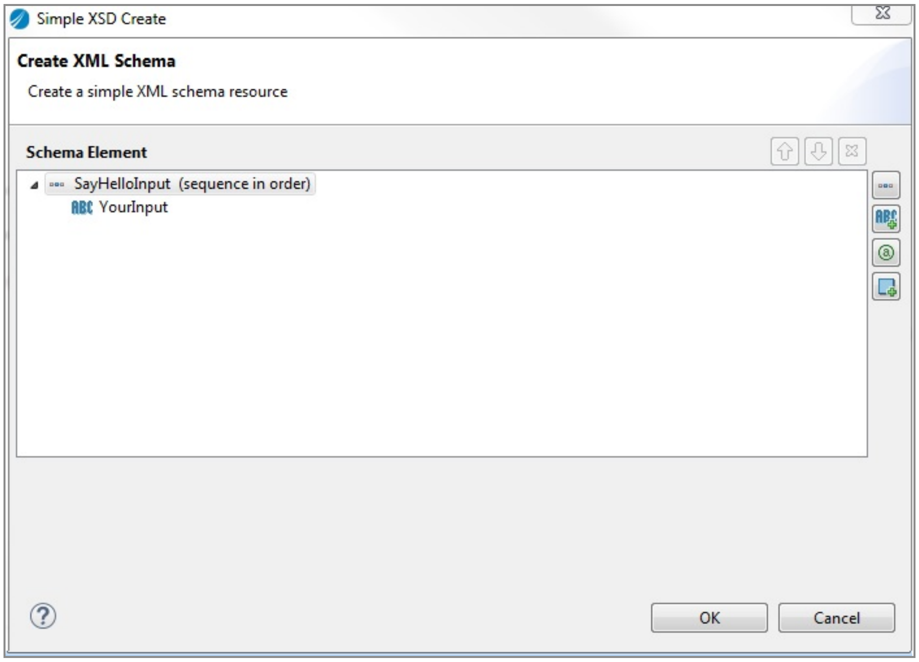
Resource schema type: XSD Editor

Create input, output or fault schema using following options.

e. From the **Resource schema type** list, select **XSD Editor**.

f. Click **Input** to create an input schema for the activity.

g. In the Create XML Schema dialog box, click to add a primitive element, and then set the element name to YourInput. Click **OK**.



Simple XSD Create

Create XML Schema

Create a simple XML schema resource

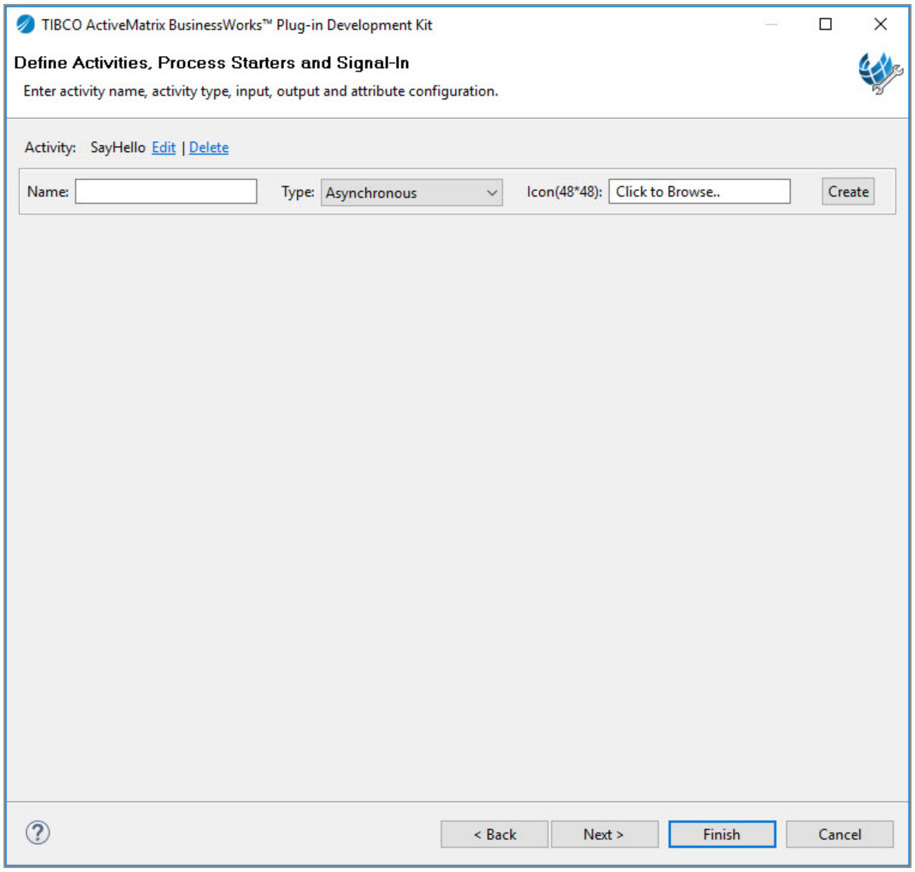
Schema Element

- SayHelloInput (sequence in order)
  - YourInput

h. Click **Output** to create an output schema for the activity. In the Create XML Schema dialog box, click ABC to add a primitive element, and then set the element name to YourOutput. Click **OK**.

5. Click **Apply** to save your activity configurations.

   You are now brought back to the Define Activities, Process Starters and Signal-In dialog box. The configured SayHello activity is displayed.



TIBCO ActiveMatrix BusinessWorks™ Plug-in Development Kit

Define Activities, Process Starters and Signal-In

Enter activity name, activity type, input, output and attribute configuration.

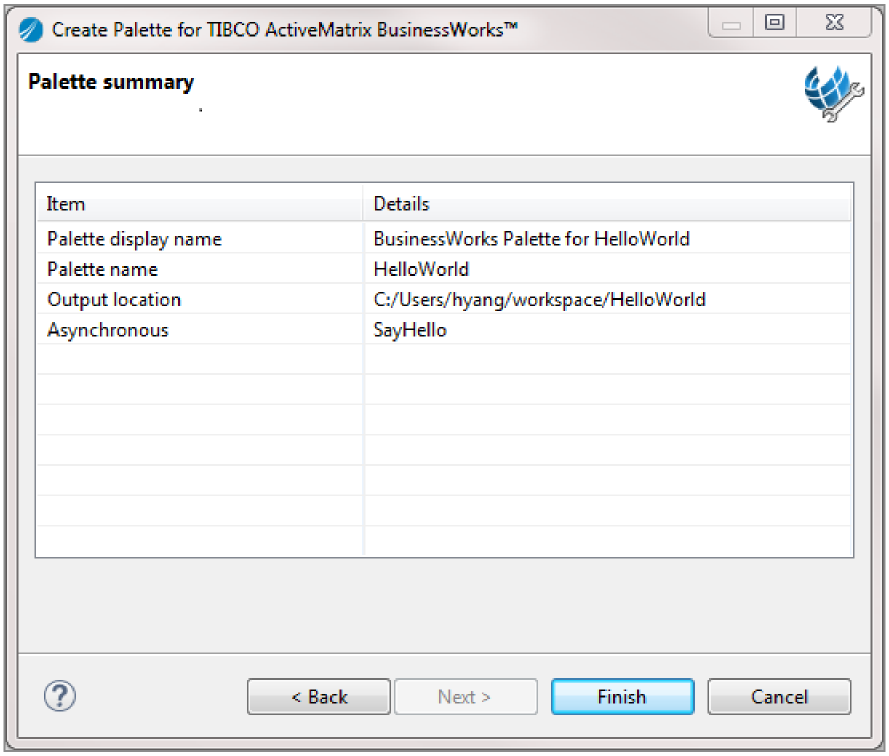
Activity: SayHello Edit | Delete

Name:

Type: Asynchronous

Icon(48*48): Click to Browse..

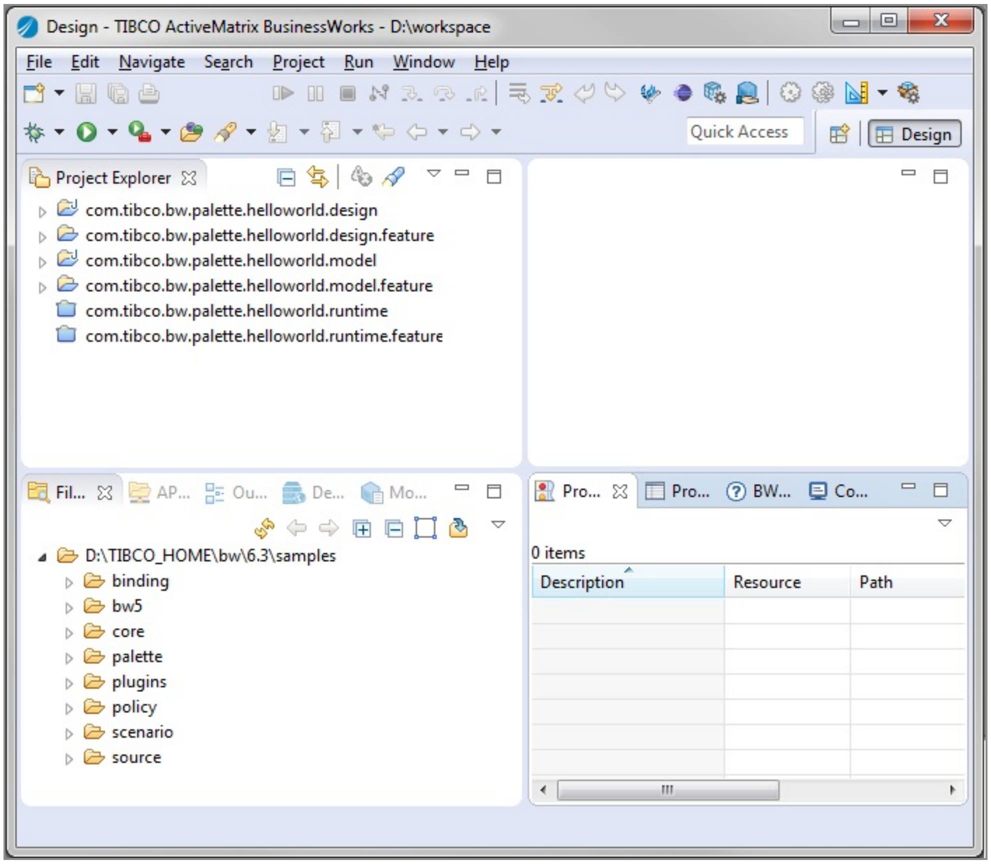
6. In the Define Activities, Process Starters and Signal-In dialog box, click **Next** to verify the palette and activity configurations.
7. In the "Palette summary" dialog box, review the palette information and click **Finish** to generate the code for the HelloWorld palette and the SayHello activity.



Item	Details
Palette display name	BusinessWorks Palette for HelloWorld
Palette name	HelloWorld
Output location	C:/Users/hyang/workspace/HelloWorld
Asynchronous	SayHello

Result

When the code generation is completed, the design-time, model, and runtime bundles and features are displayed in TIBCO Business Studio.


Design - TIBCO ActiveMatrix BusinessWorks - D:\workspace
File Edit Navigate Search Project Run Window Help
Quick Access
Design
Project Explorer
com.tibco.bw.palette.helloworld.design
com.tibco.bw.palette.helloworld.design.feature
com.tibco.bw.palette.helloworld.model
com.tibco.bw.palette.helloworld.model.feature
com.tibco.bw.palette.helloworld.runtime
com.tibco.bw.palette.helloworld.runtime.feature
Fil... AP... Ou... De... Mo...
D:\TIBCO_HOME\bw\6.3\samples
binding
bw5
core
palette
plugins
policy
scenario
source
Pro... Pro... BW... Co...
0 items
Description
Resource
Path

What to do next

Adding Business Logic

Adding Business Logic

After configuring the plug-in palette and activity, you can add business logic for the SayHello activity.

Before you begin

Ensure that you have generated a HelloWorld palette and a SayHello activity, as described in Defining a HelloWorld Palette.

Procedure

1. In TIBCO Business Studio, from the menu, click **Run > Run Configurations** to launch

a child TIBCO Business Studio:

   a. In the "Create, manage, and run configurations" dialog box, double-click **Eclipse Application** in the left panel, and then click **New_configuration**.

   b. Click **Run** to launch a child TIBCO Business Studio configured with the bw-runtime platform.

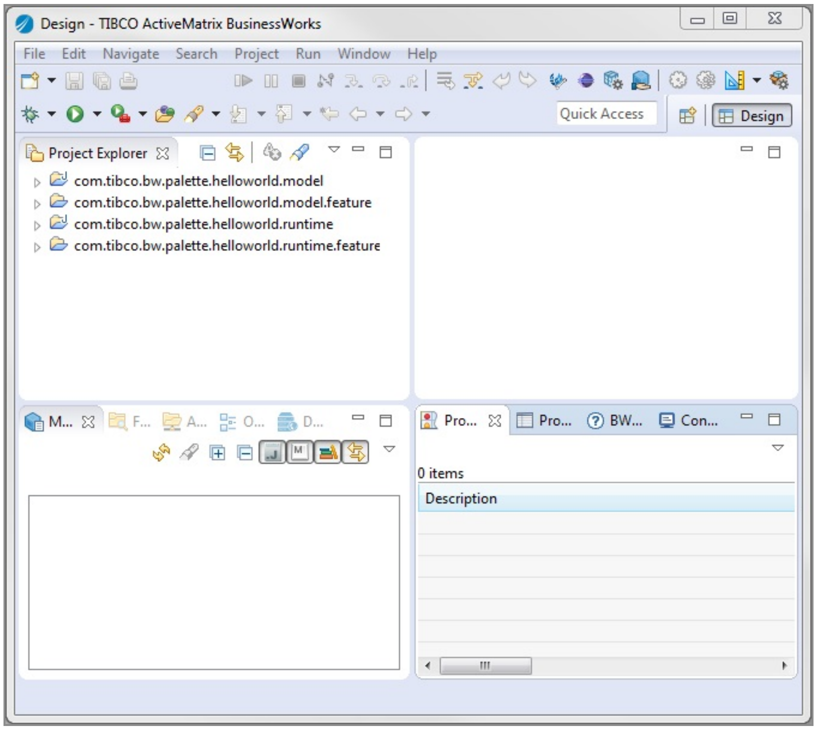
2. In the child TIBCO Business Studio, click **File > Import** to import the features and bundles of runtime and model to the child TIBCO Business Studio:

   a. In the Select dialog box, expand the **General** folder and select **Existing Projects into Workspace**. Click **Next**.

   b. In the Import Projects dialog box, click **Browse** to locate the project folder that contains the HelloWorld plug-in project.

   c. Click **Deselect All**, and then select the runtime and model bundles, and features. Click **Finish**.

   The bundles and features of runtime and model are loaded in the child TIBCO Business Studio.


Design - TIBCO ActiveMatrix BusinessWorks
File
Edit
Navigate
Search
Project
Run
Window
Help
Quick Access
Design
Project Explorer
com.tibco.bw.palette.helloworld.model
com.tibco.bw.palette.helloworld.model.feature
com.tibco.bw.palette.helloworld.runtime
com.tibco.bw.palette.helloworld.runtime.feature
M...
F...
A...
O...
D...
Pro...
Pro...
BW...
Con...
0 items
Description

3. In the Project Explorer view, expand the **com.tibco.bw.palette.helloworld.runtime** folder, and then click **src > com.tibco.bw.palette.helloworld.runtime >**

SayHelloAsynchronousActivity.java.

4. In the `SayHelloAsychronousActivity.java` file, add the `evalOutput()` method in `SayHelloActivityExecutor` class.
5. In the `evalOutput()` method, enter Hello World as the output text.

   This method is used to generate the output structure for an activity by passing the string value to the output structure.

   ```
   protected <A> N evalOutput(N inputData, ProcessingContext<N>
   processingContext, Object data) throws Exception {

          SayHelloOutput sayHelloOutput = new SayHelloOutput();
          sayHelloOutput.setOutput("String Value");
          // begin-custom-code
          // add your own business code here
   sayHelloOutput.setOutput("Hello World");
          // end-custom-code
   N outputNode = PaletteUtil.parseObjtoN(SayHelloOutput.class,
   sayHelloOutput, processingContext,
   activityContext.getActivityOutputType().getTargetNamespace(),
   "SayHelloOutput");
          return outputNode;
   }
   ```

6. Save the `SayHelloAsychronousActivity.java` file.

What to do next

Running the HelloWorld Plug-in

Running the HelloWorld Plug-in

To check whether the HelloWorld plug-in works as expected, you can create a business process using the created SayHello activity.

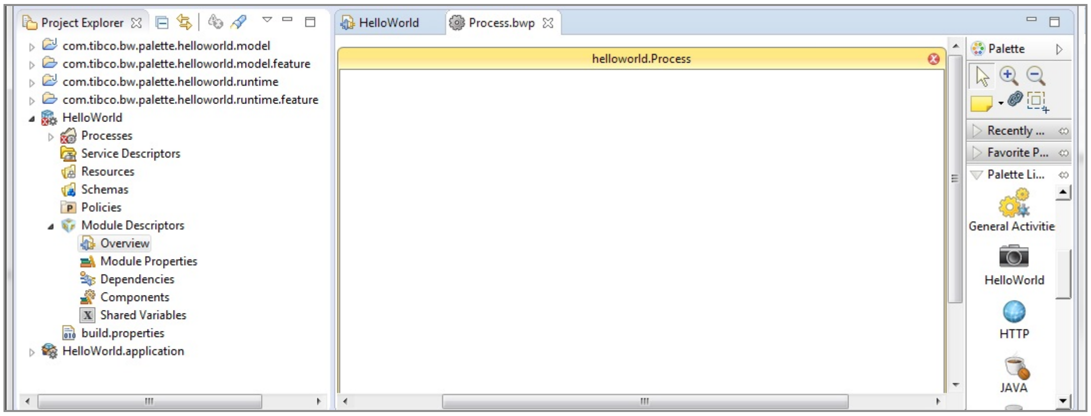
Before you begin

Ensure that you have added business logic, as described in Adding Business Logic.

Procedure

1. In the child TIBCO Business Studio, click **File > New > BusinessWorks Resources** from the menu.
2. In the "Select a wizard" dialog box, click **BusinessWorks Application Module**. Click **Next**.
3. In the Project dialog box, enter HelloWorld in the **Project name** field. Click **Finish** to create a HelloWorld project.

   The helloworld process editor is displayed.

4. From the General Activities palette, select and drop a Timer activity to the process editor. From the HelloWorld palette, select and drop a SayHello activity to the process editor.
5. Drag the icon to create a transition between the Timer activity and the SayHello activity.

   You can also click in the Palette view to create a transition.

   > **Note:** The icon is displayed only when you select a Timer activity.

6. In the process editor, double-click the SayHello activity:
   1. In the **General** tab, enter AAA in the **YourName** field.
   2. In the **Input** tab, enter AAA as the value of the YourInput element.
7. On the toolbar, click to save your configurations.
8. From the menu, click **Run > Debug Configurations**. In the "Create, manage, and run configurations" dialog box, click **BusinessWorks Application > BWApplication** in the

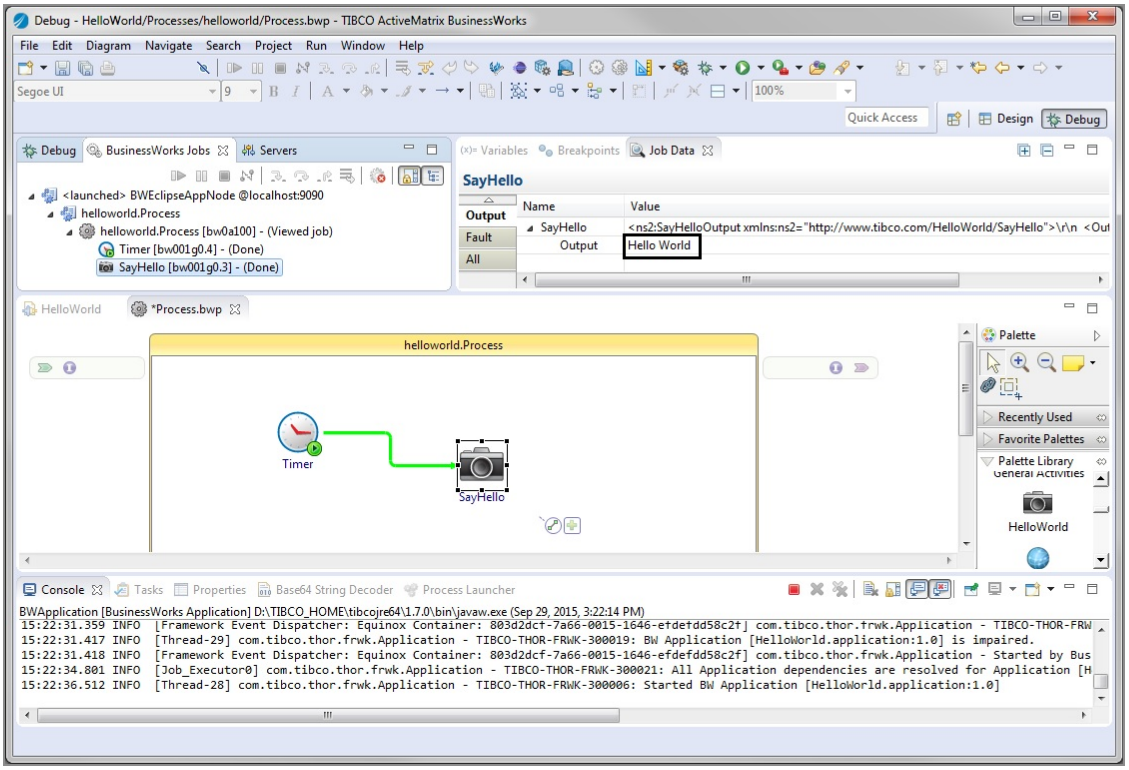
left panel, and then select **HelloWorld.application** in the **Applications** tab.

9. Click **Debug** to start the HelloWorld process.

Result

In the Debug perspective, click the **BusinessWorks Jobs** tab, and then click **helloworld.Process > SayHello**. Click the **Output** tab in the Job Data view.

Hello World is displayed in the **Output** tab.



SayHello

	Name	Value
Output	SayHello	<ns2:SayHelloOutput xmlns:ns2="http://www.tibco.com/HelloWorld/SayHello">\r\n <Out
Fault	Output	Hello World
All		

```
BWApplication [BusinessWorks Application] D:\TIBCO_HOME\tibcojre64\1.7.0\bin\javaw.exe (Sep 29, 2015, 3:22:14 PM)
15:22:31.359 INFO  [Framework Event Dispatcher: Equinox Container: 803d2dcf-7a66-0015-1646-efdefdd58c2f] com.tibco.thor.frwk.Application - TIBCO-THOR-FRW
15:22:31.417 INFO  [Thread-29] com.tibco.thor.frwk.Application - TIBCO-THOR-FRWK-300019: BW Application [HelloWorld.application:1.0] is impaired.
15:22:31.418 INFO  [Framework Event Dispatcher: Equinox Container: 803d2dcf-7a66-0015-1646-efdefdd58c2f] com.tibco.thor.frwk.Application - Started by Bus
15:22:34.801 INFO  [Job_Executor0] com.tibco.thor.frwk.Application - TIBCO-THOR-FRWK-300021: All Application dependencies are resolved for Application [H
15:22:36.512 INFO  [Thread-28] com.tibco.thor.frwk.Application - TIBCO-THOR-FRWK-300006: Started BW Application [HelloWorld.application:1.0]
```

What to do next

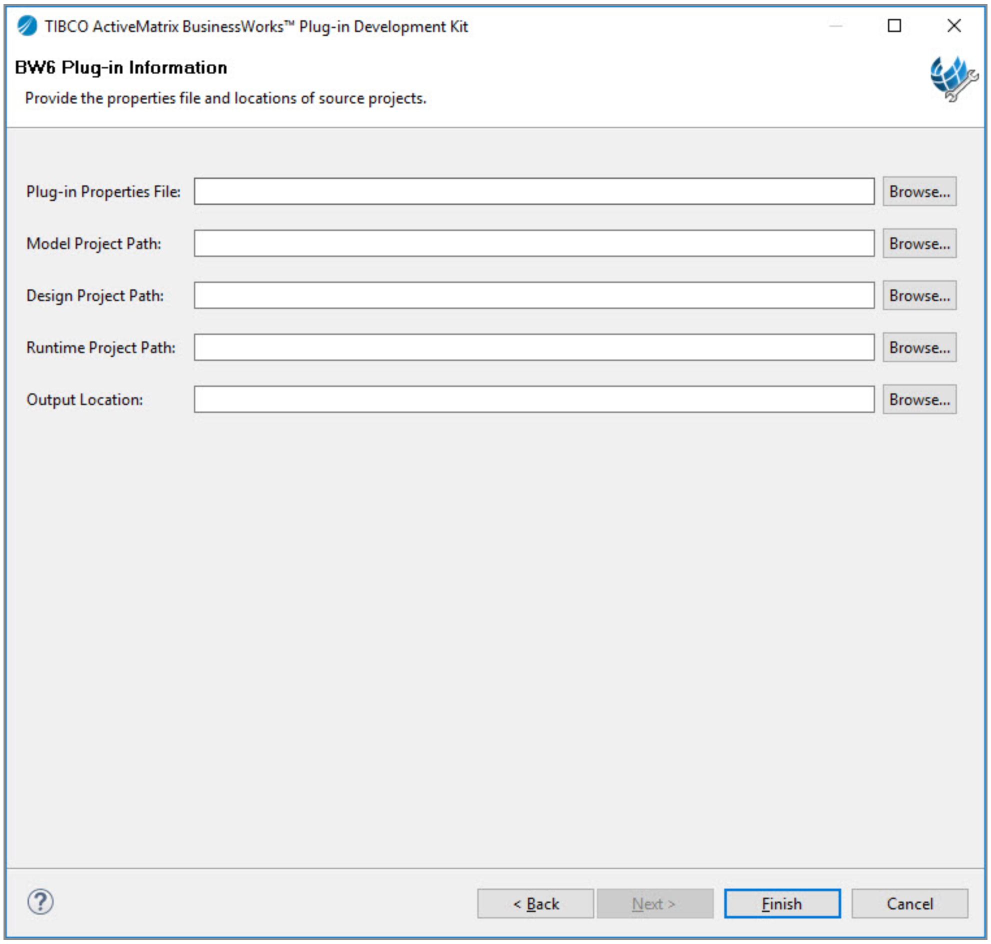
Packaging the HelloWorld Plug-in

Packaging the HelloWorld Plug-in

You can generate a provisioning platform P2 installer for the created HelloWorld plug-in.

Procedure

1. Open the BusinessWorks Plug-in Development Kit wizard in one of the following ways:
   - From the menu, click **File > New > Create a new or modify a BusinessWorks Plug-in project**.
   - On the toolbar, click .
   - Press Alt+T.
2. In the Get Started dialog box, Click **Build an Update Site for a BW6 Palette**.
3. In the BW6 Plug-in Information dialog box, provide the information for **Plug-in Properties File**.

4. Click **Browse** to locate the `bw6_devkit_configuration.properties` file that is located in the HelloWorld project folder.

   The paths for **Model Project Path**, **Design Project Path**, **Runtime Project Path**, and

Output Location will be automatically selected. You can change the path for all of them. But it is recommended that you do not change the paths.

5. Click **Finish** to package the plug-in.

What to do next

Using a Plug-in

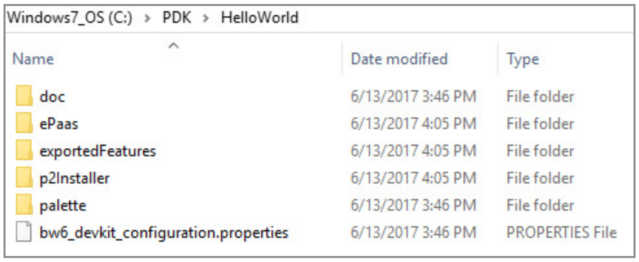
Generated Code

The following folders are generated for a created plug-in:

- The `doc` folder contains a documentation template generated according to the created palette and activity. You can update this template and use it as the online help for the created activities. See Creating Documentation for more details.
- The `ePass` folder contains the bwce runtime `zip` file for the plug-in which you have created.
- The `exportedFeatures` folder contains the bundles that the newly added source build produces.

  > **Note:** These files can be used for troubleshooting purposes only.

- The `p2Installer` folder contains the actual installable site.
- The `palette` folder contains the generated design, model, and runtime bundles. BusinessWorks Plug-in Development Kit also generates a Java file corresponding to each created activity. Each Java file contains a class where you can add your business logic. See Plug-in Bundles for more details.
- The `bw6-devkit_configuration.properties` file contains plug-in project-related information. This file is required when editing a plug-in and generating an installer.



Windows7_OS (C:) › PDK › HelloWorld

Name	Date modified	Type
doc	6/13/2017 3:46 PM	File folder
ePaas	6/13/2017 4:05 PM	File folder
exportedFeatures	6/13/2017 4:05 PM	File folder
p2Installer	6/13/2017 4:05 PM	File folder
palette	6/13/2017 3:46 PM	File folder
bw6_devkit_configuration.properties	6/13/2017 3:46 PM	PROPERTIES File

Plug-in Bundles

A *bundle* is a collection of files (resources and code) of the created plug-in. Each bundle contains different files and is associated with different functions.

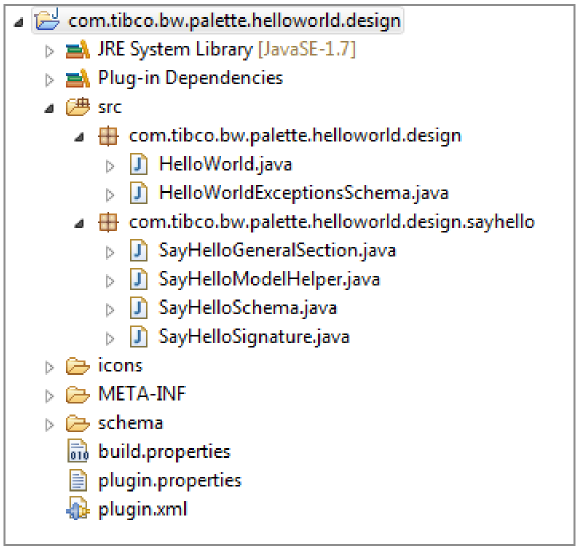
After adding and configuring activities for your plug-in, you have to change the target platform to design time. When the code generation is completed, the following bundles are generated and displayed in the Project Explorer view:

- Design-Time Bundle
- Model Bundle
- Runtime Bundle

Design-Time Bundle

The design-time bundle contains the code related to activity configurations.

Open the parent TIBCO Business Studio configured with the running platform, and then expand the design-time bundle in the Project Explorer view, the following files and folders are displayed:



```
com.tibco.bw.palette.helloworld.design
    JRE System Library [JavaSE-1.7]
    Plug-in Dependencies
    src
        com.tibco.bw.palette.helloworld.design
            HelloWorld.java
            HelloWorldExceptionsSchema.java
        com.tibco.bw.palette.helloworld.design.sayhello
            SayHelloGeneralSection.java
            SayHelloModelHelper.java
            SayHelloSchema.java
            SayHelloSignature.java
    icons
    META-INF
    schema
    build.properties
    plugin.properties
    plugin.xml
```

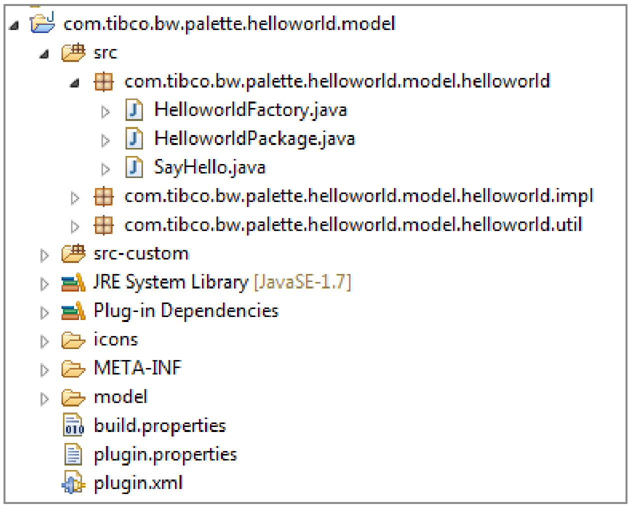
Folders and Files	Description
`src`	Contains the source code of the design-time configurations. A palette package is created, which contains the following Java files: • The *`Palette_Name`*`.java` file contains the methods to access your plug-

Folders and Files	Description
	in project. • The *Palette_Name*ExceptionsSchema.java file contains the fault schema related methods. **Note:** This file is generated only when you do *not* configure any fault schema. Besides, a separate package is created corresponding to each activity. Each activity package contains the following Java files: • The *ActivityName*General/Advanced/*customized*Section.java file contains the source code of the **General** tab, the **Advanced** tab, and the ***customized*** tab, including the source code for the GUI elements configured in these sections. • The *ActivityName*ModelHelper.java file contains the source code to initiate a model. • The *ActivityName*Schema.java file contains the input and output schema. **Note:** This file is generated when using an XSD file or a WSDL file or using XSD Editor to generate the activity input and output. • The *ActivityName*Signature.java file contains the source code of the **Input**, **Output**, and **Fault** tabs.
icons	Contains the palette icon to be displayed in the Palette view.
META-INF	Contains a MANIFEST.MF file that provides information regarding the plug-in bundle and package.
schema	Contains the input, output, and fault schema files of each activity.
plugin.xml	Describes how the plug-in extends the platform, what extensions you can use, and how the plug-in implements its functionality.

Model Bundle

The model bundle contains the code related to the data model and the implementation validator of the plug-in activities.

Open the parent TIBCO Business Studio configured with the running platform, and then expand the model bundle in the Project Explorer view, the following files and folders are displayed:



```
com.tibco.bw.palette.helloworld.model
  src
    com.tibco.bw.palette.helloworld.model.helloworld
      HelloworldFactory.java
      HelloworldPackage.java
      SayHello.java
    com.tibco.bw.palette.helloworld.model.helloworld.impl
    com.tibco.bw.palette.helloworld.model.helloworld.util
  src-custom
  JRE System Library [JavaSE-1.7]
  Plug-in Dependencies
  icons
  META-INF
  model
  build.properties
  plugin.properties
  plugin.xml
```

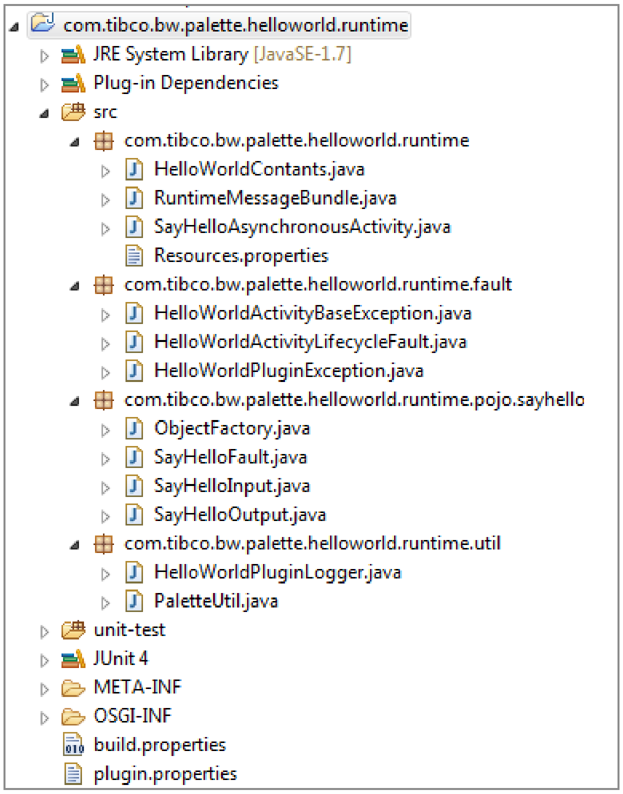
Folders and Files	Description
`src`	Contains the source code of the data model. Use the interface and corresponding implementation files of the activity to create a data model.
`src-custom`	Contains the implementation code of the activity validator.
`icons`	Contains the activity icon to be displayed in the Palette view and process editor.
`META-INF`	Contains a `MANIFEST.MF` file that provides information regarding the plug-in bundle and package.
`model`	Contains a `.ecore` file that describes the data model structure and a `.genmodel` file that generates the code of the data model.

Folders and Files	Description
`plugin.xml`	Describes how the plug-in extends the platform, what extensions you can use, and how the plug-in implements its functionality.

Runtime Bundle

The runtime bundle contains the code related to business logic implementation and unit test.

Open the child TIBCO Business Studio configured with the bw-runtime platform, and then expand the runtime bundle in the Project Explorer view, the following files and folders are displayed:



```
com.tibco.bw.palette.helloworld.runtime
  JRE System Library [JavaSE-1.7]
  Plug-in Dependencies
  src
    com.tibco.bw.palette.helloworld.runtime
      HelloWorldContants.java
      RuntimeMessageBundle.java
      SayHelloAsynchronousActivity.java
      Resources.properties
    com.tibco.bw.palette.helloworld.runtime.fault
      HelloWorldActivityBaseException.java
      HelloWorldActivityLifecycleFault.java
      HelloWorldPluginException.java
    com.tibco.bw.palette.helloworld.runtime.pojo.sayhello
      ObjectFactory.java
      SayHelloFault.java
      SayHelloInput.java
      SayHelloOutput.java
    com.tibco.bw.palette.helloworld.runtime.util
      HelloWorldPluginLogger.java
      PaletteUtil.java
  unit-test
  JUnit 4
  META-INF
  OSGI-INF
  build.properties
  plugin.properties
```

Folders and Files	Description
`src`	Contains the source code to implement the business logic of each activity. • The runtime package contains the following files: ◦ The *PaletteName*`Contants.java` file contains the defined constants used in the code. ◦ The `RuntimeMessageBundle.java` file contains the bundle message definition. ◦ The *ActivityNameActivityType*`.java` file contains the business logic of the activity, including the logic information on how to get configurations and input of the activity, and how to generate output according to design-time configurations. ◦ The `Resources.properties` file contains the implementation of bundle messages and error codes. • The fault package contains the following files: ◦ The *PaletteName*`ActivityBaseException.java` file contains the errors specified for the synchronous and asynchronous activities. ◦ The *PaletteName*`ActivityLifecycleFault.java` file contains initialization errors. ◦ The *PaletteName*`PluginException.java` file contains the error defined by BusinessWorks Plug-in Development Kit by default. ◦ The *FaultName*`Fault.java` file contains the generated fault schema according to the design-time configurations. Use the `private <N, A> N constructErrData ()` method and the `public QName getFaultElementQName()` method to add other faults at run time. The `private <N, A> N constructErrData ()` method is generated only when you add a customized fault at design time. If you do not specify any fault schema at design time, this file is not generated. **Note:** The *PaletteName*`ActivityBaseException.java` file is not available for the signal-in and process starter activities. • The pojo package contains Java Architecture for XML Binding (JAXB)

Folders and Files	Description
	generated common Java classes according to the output configurations. You can assign values to these POJO classes. - The util package contains the following files: ○ The *PaletteName*PluginLogger.java file contains the generated logging utility. ○ The PaletteUtil.java file contains the generated output utility.
unit-test	Contains the source code for unit tests.
META-INF	Contains a MANIFEST.MF file that provides information regarding the plug-in bundle and package.
OSGI-INF	Contains configuration files of OSGI bundles. Each activity corresponds to a bundle file.

Design-Time Class Specification

Use the design-time classes to generate input and output schema, and GUI elements.

The following design-time classes are generated when generating the palette code:

- The com.*companyname*.bw.palette.*palette_name*.design package contains the following classes:
  - [PaletteName]
  - [PaletteName]ExceptionsSchema
- The com.*companyname*.bw.palette.*palette_name*.design.*activity_name* package contains the following classes:
  - [ActivityName]General/Advanced/customizedSection
  - [ActivityName]ModelHelper
  - [ActivityName]Schema

  - [ActivityName]Signature

[PaletteName]

This class is used to access your plug-in project.

The `[PaletteName]` class contains the following methods:

Methods	Description
public void start()	This method is called when the design-time bundle is initialized by Eclipse. This is a BusinessWorks 6 life-cycle method.
public void stop()	This method is called when the design-time bundle is stopped by Eclipse. This is a BusinessWorks 6 life-cycle method.
public static *palette_name* getDefault()	Use this method to get a plug-in instance. This is a BusinessWorks 6 life-cycle method.

[PaletteName]ExceptionsSchema

This class is used to get the fault schema related information.

The `[PaletteName]ExceptionsSchema` class contains the following methods:

Methods	Description
protected InputStream getSchemaAsInputStream ()	Use this method to get the input stream of the fault schema. This is a BusinessWorks 6 life-cycle method.

Methods	Description
public static List<XSDElementDeclaration> get*plug-in_name*FaultTypes()	Use this method to get the fault type.
private static List<XSDElementDeclaration> getFaultElements()	Use this method to get the fault elements.

[ActivityName]General/Advanced/*customized*Section

This class is used to configure the **General**, **Advanced**, and ***customized*** tabs of an activity.

The [ActivityName]General/Advanced/*customized*Section class contains the following methods:

Methods	Description
protected Class <?> getModelClass()	Use this method to specify a representation of a model object. This is a BusinessWorks 6 life-cycle method.
protected void initBindings()	Use this method to initialize control bindings to the activity input. This is a BusinessWorks 6 life-cycle method.
protected Composite doCreateControl()	Use this method to configure activity attributes in the **General**, **Advanced**, and ***customized*** tabs. This is a BusinessWorks 6 life-cycle method.

Note: If you want to get the attribute value, you have to add a `String attributes_value= activityConfig.getAttributeName()` method.

[ActivityName]ModelHelper

The `[ActivityName]ModelHelper` class contains a `public EObject createInstance()` method. This is a BusinessWorks 6 life-cycle method. You can use this method to create a model instance and set the initialization value for the model instance.

[ActivityName]Schema

This class is used to parse an XSD file and get the input, output, and fault type according to the defined elements. This class is called by the `[ActivityName]Signature` class.

The `[ActivityName]Schema` class contains the following methods:

Methods	Description
public static XSDElementDeclaration getInputType()	Use this method to get the input type.
public static XSDElementDeclaration getOutputType()	Use this method to get the output type.
public static List<XSDElementDeclaration> getFaultElements()	Use this method to get the fault element. **Note:** If you do not specify any fault schema at design time, this method is not generated.
protected InputStream getSchemaAsInputStream()	Use this method to get the input stream. This is a BusinessWorks 6 life-cycle method.

[ActivityName]Signature

This class is used to create input, output, and fault schema for an activity.

The `[ActivityName]Signature` class contains the following methods:

Methods	Description
public boolean hasInput()	Returns a value of false when an activity has no input. This is a BusinessWorks 6 life-cycle method.
public boolean hasOutput()	Returns a value of false when an activity has no output. This is a BusinessWorks 6 life-cycle method.
public XSDElementDeclaration getInputType()	Use this method to configure an input schema. This is a BusinessWorks 6 life-cycle method.
public XSDElementDeclaration getOutputType()	Use this method to configure an output schema. This is a BusinessWorks 6 life-cycle method.
public List<XSDElementDeclaration> getFaultTypes()	Use this method to configure a schema for the **Fault** tab. This is a BusinessWorks 6 life-cycle method.
private XSDSchema getCompiledSchema()	Use this method to configure the imported XSD schema. **Note:** This method is generated only when the XSD or WSDL file that you select for the activity imports other XSD schema.

Runtime Class Specification

Use runtime classes to add business logic for each activity.

The following runtime classes are generated:

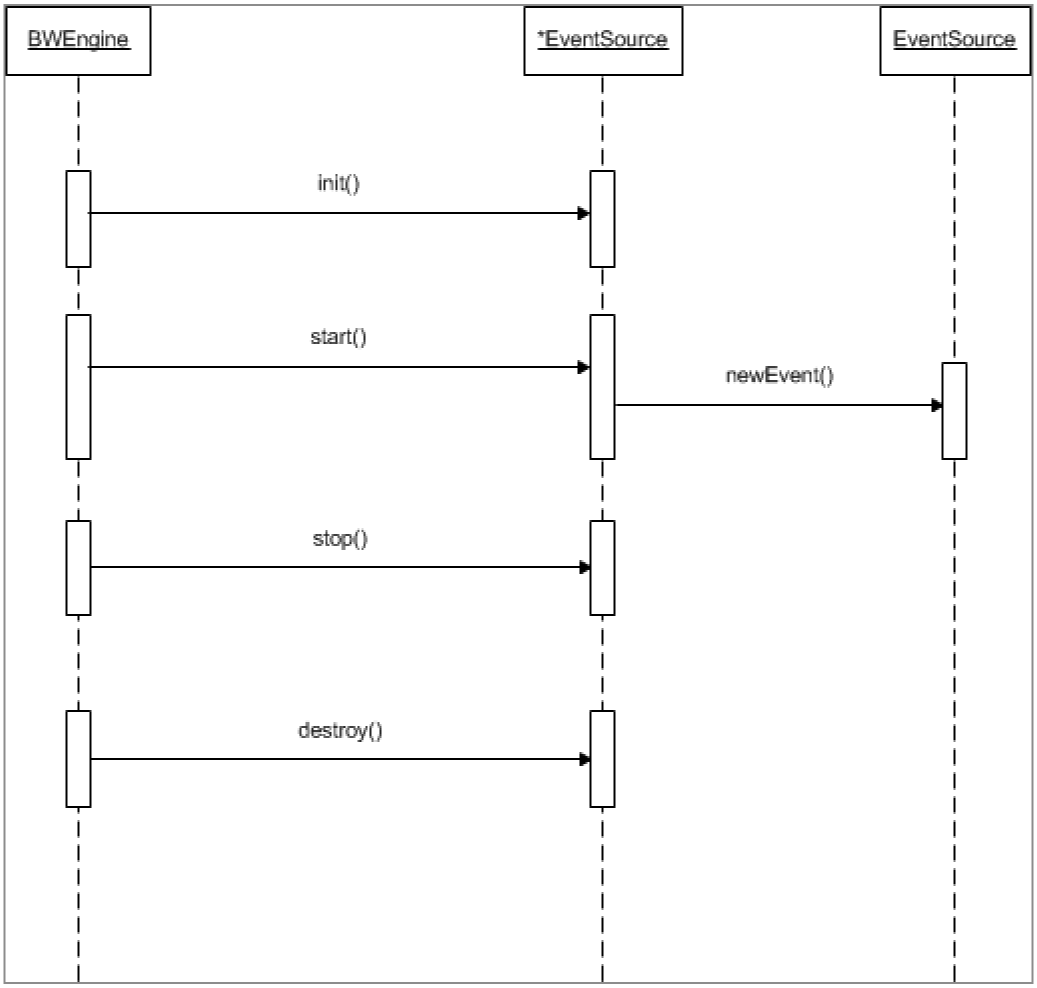
- [ActivityName]EventSource
- [ActivityName]AsynchronousActivity
- [ActivityName]SynchronousActivity

[ActivityName]EventSource

This class is used to add business logic for the process starter and signal-in activities.

Both the process starter and signal-in activities are event source activities and use the same `EventSource<N>` class.

The following sequence diagram illustrates how the methods in the `[ActivityName]EventSource<N>` class are invoked:



The `[ActivityName]EventSource<N>` class contains the following methods:

Methods	Description
public synchronized void destroy()	Use this method to release or clean resources held by a source, when an event source is destroyed.

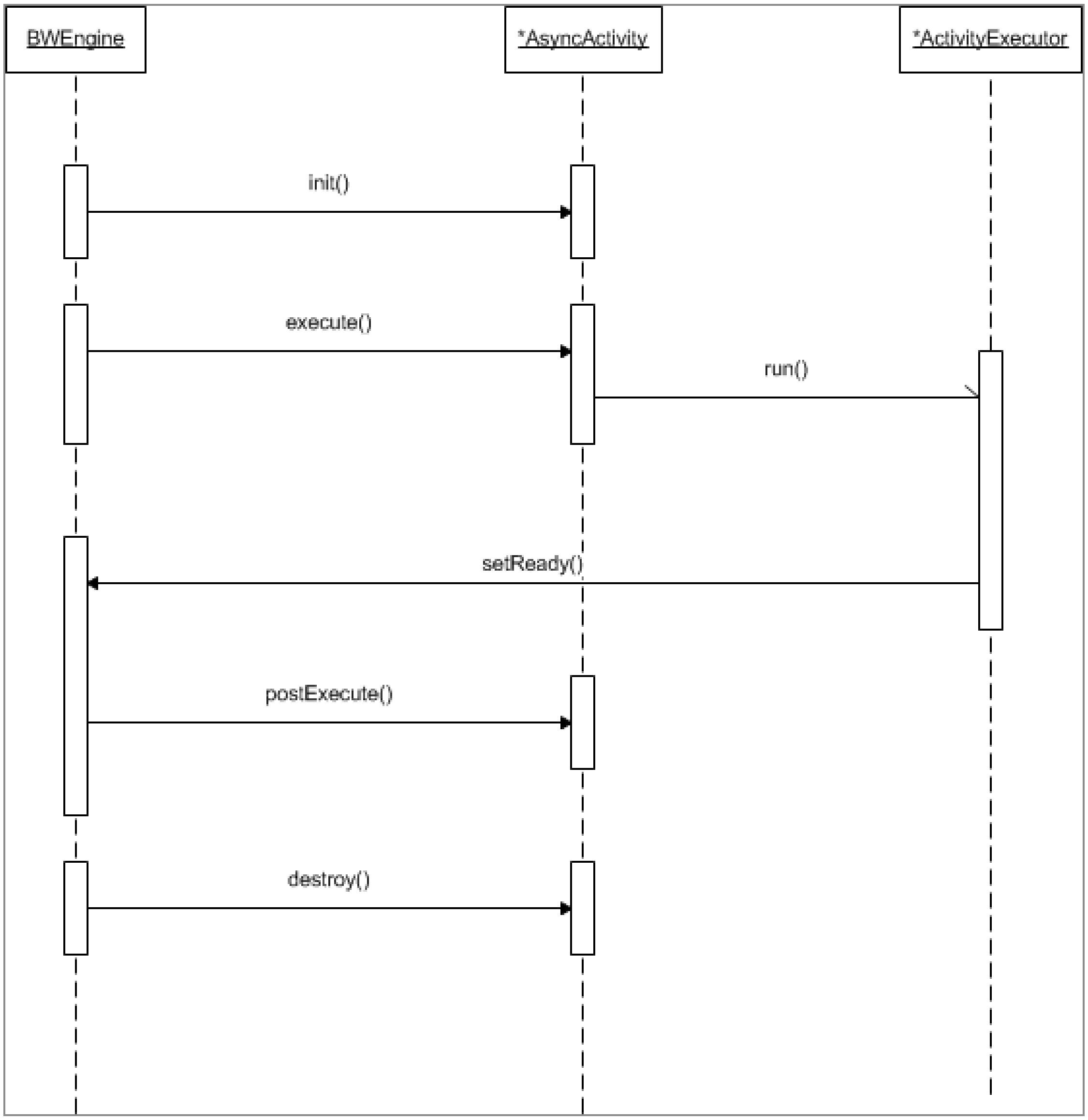
Methods	Description
	This is a BusinessWorks 6 life-cycle method.
public synchronized boolean isStarted()	Returns a boolean value indicating the status of an event source. This is a BusinessWorks 6 life-cycle method.
public synchronized void start()	Use this method to start an event source. You cannot start a new event until this method is called. Once this method is called, the event source uses the {@link EventSourceContext} interface to notify the BusinessWorks engine of a new event. You can use the `{@link EventSource#getEventSourceContext()}` method to get the {@link EventSource#getEventSourceContext()} object. This is a BusinessWorks 6 life-cycle method.
protected <A> N evalOutput()	Use this method to generate output when the business is completed.
protected N getOutputRootElement()	Use this method to get the root element of the output.
public synchronized void stop()	Use this method to stop the event source from processing new events. When this method is called, the event source cannot use the {@link EventSourceContext} interface to notify the BusinessWorks engine of a new event. **Caution:** Be careful when using this method to release or delete the resources that are used to start an event source. The `{@link EventSource#start()}` method cannot be called after the `{@link Event source#stop()}` method. This is a BusinessWorks 6 life-cycle method.
public void init()	Use this method to perform any required initialization.

Methods	Description
	The `<code>eventSourceKind</code>` argument of this method indicates that the event source is being initialized by a process starter activity or a signal-in activity. **Note:** Do not use this method to start an event source. You can start the event source until the `{@link Event source#start()}` method is called. This is a BusinessWorks 6 life-cycle method.

[ActivityName]AsynchronousActivity

This class is used to add business logic for an asynchronous activity.

The following sequence diagram illustrates how the methods in the `[ActivityName]AsynchronousActivity<N>` class are invoked:



The `[ActivityName]AsynchronousActivity<N>` class contains an internal class and the following methods:

Methods	Description
public void init()	Use this method to initialize the activity. This is a BusinessWorks 6 life-cycle method.
public void destroy()	Use this method to release or clean resources

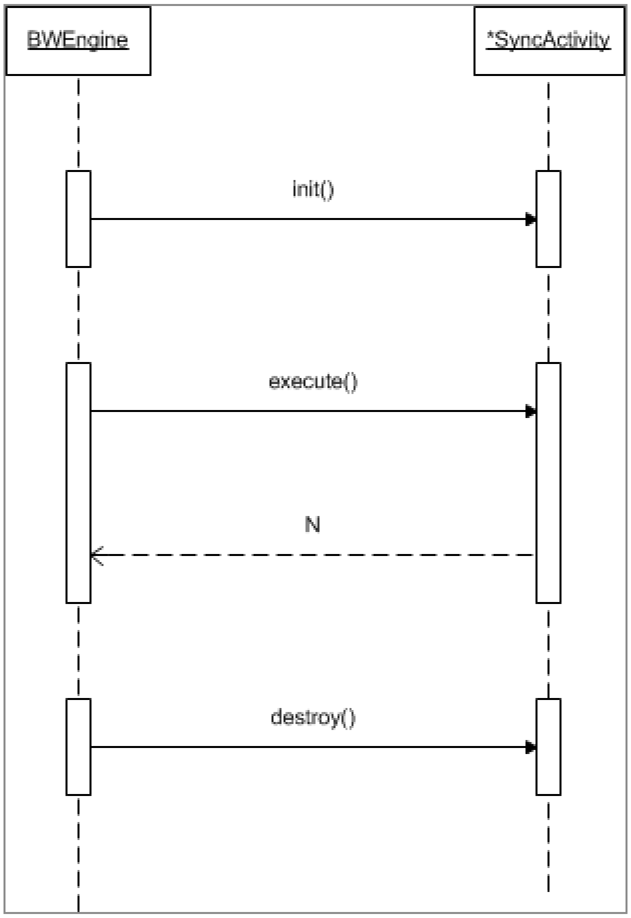
Methods	Description
	held by a source, when an event source is destroyed. This is a BusinessWorks 6 life-cycle method.
public void requestCancel()	Use this method to revoke the execution of an asynchronous activity, when the application is closed, or the asynchronous activity is still running within the waiting time that is specified in the `{@link AsyncActivityController#setPending}` method. This is a BusinessWorks 6 life-cycle method. This method sets the instance variable cancelRequested. Business logic in run() method may check it periodically and may end if the value is True
public void execute()	Use this method to execute an asynchronous activity. **Note:** Do not execute the activity and business logic in the same thread. This is a BusinessWorks 6 life-cycle method.
public N postExecute()	Use this method to complete the execution of an asynchronous activity, when the asynchronous activity issues a signal of completion by invoking the `{@link AsyncActivityCompletionNotifier#setReady}` method in a different thread. This is a BusinessWorks 6 life-cycle method.
protected <A> N evalOutput()	Use this method to generate output when the business is completed.

Methods	Description
protected <N> N getOutputRootElement()	Use this method to get the root element of the output.
public String getInputParameterStringValueByName()	Use this method to get the value of a string parameter according to the parameter name from the input. **Note:** This method cannot retrieve the value of a sub node.
public String getInputAttributeStringValueByName()	Use this method to get the value of an attribute according to the parameter name from the input.
public boolean getInputParameterBooleanValueByName()	Use this method to get the value of a boolean parameter according to the parameter name from the input.
The [ActivityName]AsynchActivityExecutor<A> internal class contains the following method:	
public void run()	Use this method to add business logic before calling the `evalOutput()` method. This method is executed in another thread.

[ActivityName]SynchronousActivity

This class is used to add business logic for a synchronous activity.

The following sequence diagram illustrates how the methods in the `[ActivityName]SynchronousActivity<N>` class are invoked:



The `[ActivityName]SynchronousActivity<N>` class contains the following methods:

Methods	Description
public void init()	Use this method to initialize the activity. This is a BusinessWorks 6 life-cycle method.
public void destroy()	Use this method to release or clean resources held by an event source, when an event source is destroyed. This is a BusinessWorks 6 life-cycle method.
public N execute()	Use this method to define the execution of a synchronous activity.

Methods	Description
	This is a BusinessWorks 6 life-cycle method.
protected <A> N evalOutput	Use this method to generate output when the business is completed.
protected N getOutputRootElement()	Use this method to get the root element of the output.
public String getInputParameterStringValueByName()	Use this method to get the value of a string parameter according to the parameter name from the input. **Note:** This method cannot retrieve the value of a sub node.
public String getInputAttributeStringValueByName()	Use this method to get the value of an attribute according to the parameter name from the input.
public boolean getInputParameterBooleanValueByName()	Use this method to get the value of a boolean parameter according to the parameter name from the input.

Target Platform

During the development of plug-ins, your plug-in depends on other plug-ins, for example, the SWT and JFace plug-ins. The set of plug-ins that you can use for your development is defined by the plug-ins in your workspace in addition to the plug-ins defined by your target platform. There are two target platforms, namely Design and Runtime.

When you launch TIBCO Business Studio, the Runtime platform is used by default.

- Design Platform

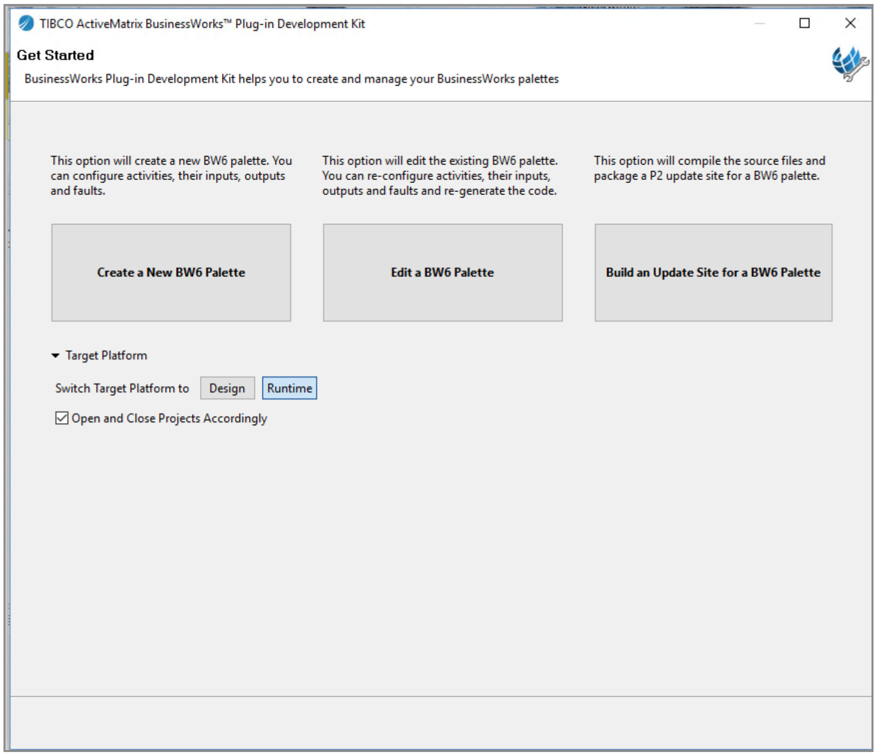
  This target platform is required for the design-time module. You can work on the UI-related features when the target platform is set to Design platform..

- Runtime Platform

  This target platform is required for the runtime module. You can work on the runtime related features when the target platform is set to Runtime platform.

To change the platform:

Click **File > New > Create a new or modify a BusinessWorks Plug-in project**, and then in the Get Started dialog box, expand **Target Platform** and click Design. The target platform in use is selected.



TIBCO ActiveMatrix BusinessWorks™ Plug-in Development Kit

Get Started

BusinessWorks Plug-in Development Kit helps you to create and manage your BusinessWorks palettes

This option will create a new BW6 palette. You can configure activities, their inputs, outputs and faults.

This option will edit the existing BW6 palette. You can re-configure activities, their inputs, outputs and faults and re-generate the code.

This option will compile the source files and package a P2 update site for a BW6 palette.

☑ Open and Close Projects Accordingly

Note: Design projects will not compile when runtime platform is selected and runtime projects will not compile when runtime platform is selected. The **Open and Close Projects Accordingly** parameter ensures that only projects appropriate to the selected platform are opened.

Creating a Plug-in

TIBCO ActiveMatrix BusinessWorks plug-ins extend the functions of ActiveMatrix BusinessWorks by adding more activities. A TIBCO ActiveMatrix BusinessWorks plug-in is designed to integrate third-party applications with ActiveMatrix BusinessWorks.

To create a plug-in, complete the following tasks:

1. Defining a Palette
2. Adding and Configuring Activities
3. Adding Business Logic

Defining a Palette

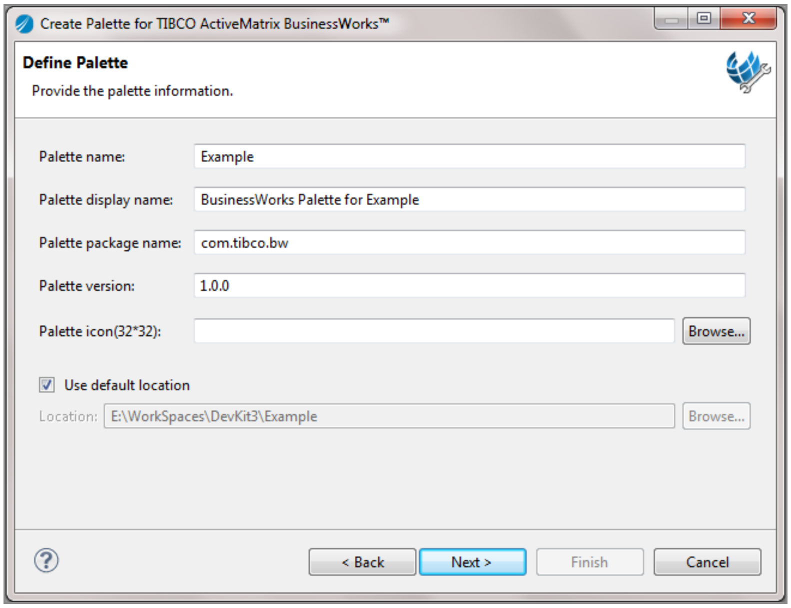
A *palette* groups the plug-in activities together.

Procedure

1. Open TIBCO Business Studio in one of the following ways:
   - Microsoft Windows: click **Start > All Programs > TIBCO > *TIBCO_HOME* > TIBCO Business Studio *version_number* > Studio for Designers**.
   - Linux: run the TIBCO Business Studio executable located in the `TIBCO_HOME/studio/version_number/eclipse` directory.
   - Mac OS: run the TIBCO Business Studio executable located in the `TIBCO_HOME/studio/version_number/eclipse` directory.
2. Open the BusinessWorks Plug-in Development Kit wizard in one of the following ways:
   - From the menu, click **File > New > Create a new or modify BW Plug-in Project**.
   - On the toolbar, click **Create a new or modify BW Plug-in Project**.

- Press Alt+T.

3. Click **Create New BW6 Palette** to start creating a plug-in.
4. In the Define Palette dialog box, configure a palette for the plug-in:



Create Palette for TIBCO ActiveMatrix BusinessWorks™

Define Palette

Provide the palette information.

Palette name: Example

Palette display name: BusinessWorks Palette for Example

Palette package name: com.tibco.bw

Palette version: 1.0.0

Palette icon(32*32):

☑ Use default location

Location: E:\WorkSpaces\DevKit3\Example

   a. In the **Palette name** field, enter a palette name.

      By default, Example is the palette name.

   b. In the **Palette display name** field, enter a display name for the palette.

      The display name is displayed as the palette name in TIBCO Business Studio.

   c. In the **Palette package name** field, enter a name of the package, which is the suffix of the generated package name.

   d. In the **Palette version** field, enter the palette version that you want to use.

      By default, the palette version is 1.0.0.

   e. In the **Palette icon** field, click **Browse** to locate an icon for the palette.

      By default, 📷 is used as the palette icon. The supported image formats are .jpg, .gif, .png, and .jpeg. The size of the icon must be 32x32 or larger. BusinessWorks Plug-in Development Kit automatically generates the icon in size of 16x16 and 32x32.

   f. By default, the **Use default location** check box is selected. The palette project is saved to the workspace that you use in current. If you want to change the location of the project, clear the check box and click **Browse** to select a new location.

5. Click **Next** to add activities for the created palette.

What to do next

Adding and Configuring Activities

Adding and Configuring Activities

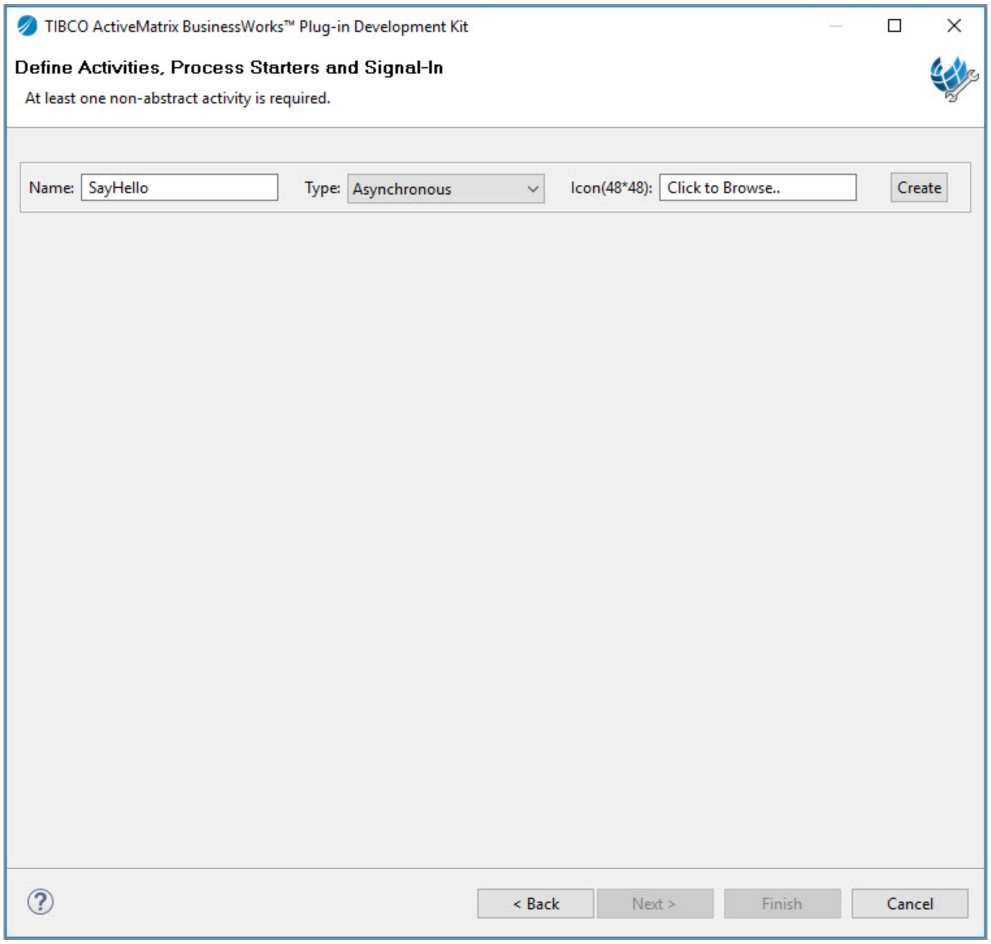
An activity is an individual unit of work in a process. Each activity is represented by an icon in TIBCO Business Studio and consists of a set of configurations.

Before you begin

Ensure that you have configured a palette, as described in Defining a Palette.

Procedure

1. In the **Name** field, type a name of the activity that you want to create.



TIBCO ActiveMatrix BusinessWorks™ Plug-in Development Kit

Define Activities, Process Starters and Signal-In

At least one non-abstract activity is required.

Name: SayHello Type: Asynchronous Icon(48*48): Click to Browse..

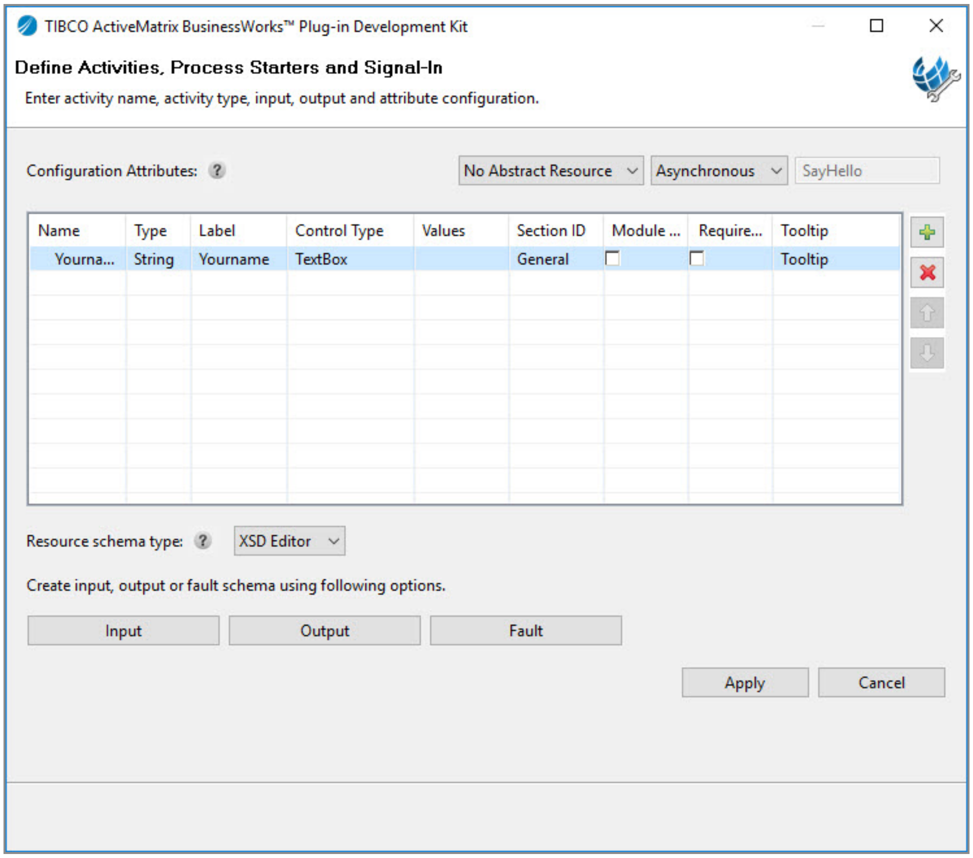
2. From the **Type** list, select an activity type.

   See Activity Types for more details.

3. By default, 📷 is used as the activity icon. If you want to change the activity icon, click the text box next to the **Icon** field, and select an icon to represent the activity.

   The supported image formats are .jpg, .gif, .png, and .jpeg. The size of the icon must be 48x48 or larger. BusinessWorks Plug-in Development Kit automatically generates the icon in size of 16x16, 32x32, and 48x48.

4. Click **Create** to configure the activity. Click + to add attributes for the activity. Each activity attribute contains the following properties:



Define Activities, Process Starters and Signal-In

Enter activity name, activity type, input, output and attribute configuration.

Configuration Attributes: No Abstract Resource | Asynchronous | SayHello

Name	Type	Label	Control Type	Values	Section ID	Module ...	Require...	Tooltip
Yourna...	String	Yourname	TextBox		General	□	□	Tooltip

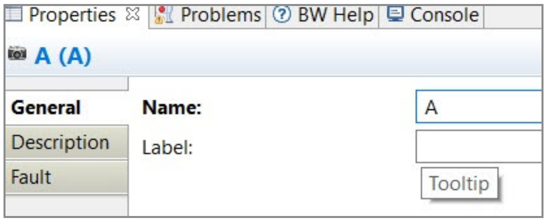
Resource schema type: XSD Editor

Create input, output or fault schema using following options.

The below table describes the attributes for the activity which you have created.

Attributes	Description
Name	Name of the activity.
Type	Type of the attribute. The following attribute types are supported: Int, Short, String, Float, Double, Long, Boolean, and Date. **Note:** The attribute of the Double, Long, Float, or Short type is not displayed in TIBCO Business Studio.
Label	Display name of the attribute. The display name is the field name

Attributes	Description
	displayed in TIBCO Business Studio.
Control Type	Control type of the attribute. The following types are supported: Hide, TextBox, FilePickerField, ComboViewer, Button, PasswordField, DirectoryPickerField, ActivityReferenceField, and PropertyField. BusinessWorks Plug-in Development Kit automatically selects a control type according to the selected attribute type. **Note:** If you set the control type of an attribute to **Hide**, the attribute of the activity is not displayed on UI. If you want to use shared resources, select **PropertyField** from **Control Type** list. For more details, see Using Shared Resources.
Values	Values to be displayed as default options. Use comma (,) to separate values when the control type is **ComboViewer**.
Section ID	Name of the section that the attribute belongs to. The following three options are supported: • General • Advanced • Any customized section name
Module Property	Whether the attribute supports the module property feature or not.
Required Field	Whether the attribute is required or not.
Tooltip	You can specify for activity fields. The content will be shown in the

Attributes	Description
	General tab of the Properties view.  Properties Problems BW Help Console A (A) General Description Fault Tooltip Name: A Label:

5. From the **Resource schema type** list, select a way to generate the input, output, and fault schema:

   - If you select **XSD/WSDL**, click **Browse** to locate an XSD file or a WSDL file. BusinessWorks Plug-in Development Kit automatically generates input, output, and fault schema according to the XSD or WSDL file. See Creating Schema with XSD/WSDL for more details.

   - If you select **Java Code**, BusinessWorks Plug-in Development Kit automatically generates a sample code of input and output.

   - If you select **XSD Editor**, you have to manually create input, output, and fault schema. See Creating Schema with XSD Editor for more details.

   - If you select **None**, no input and output are generated. BusinessWorks Plug-in Development Kit generates default schema for fault.

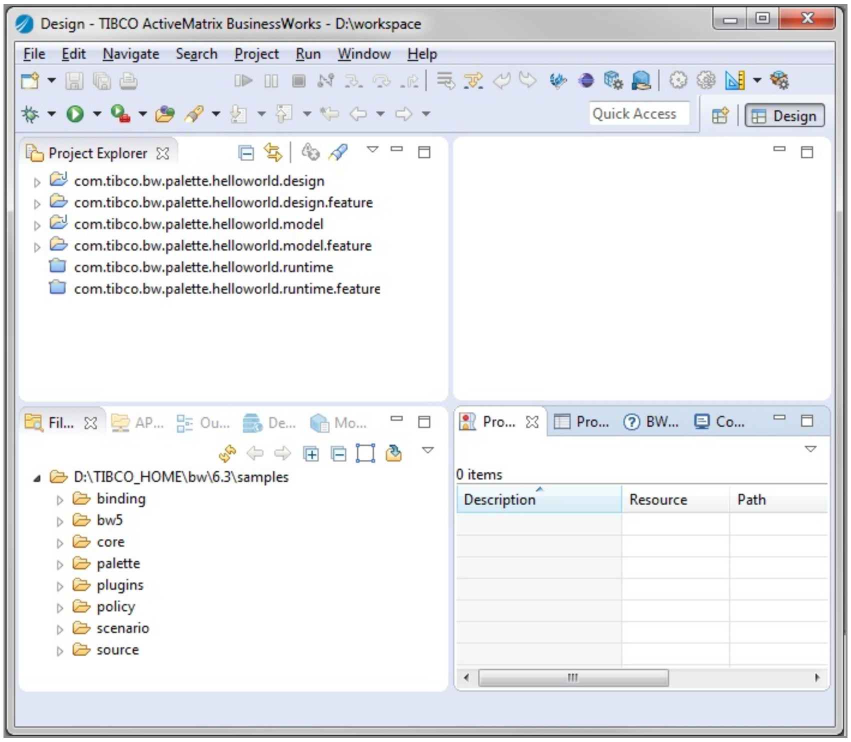
6. Click **Apply** to save your configurations.

   You are now directed to the Define Activities, Process Starters and Signal-In dialog box. Go back to In the Name field, type a name of the activity that you want to create. to add more activities.

7. Click **Finish** to generate code for the defined palette and activities.

Result

When the code generation is completed, the design-time, model, and runtime bundles and features are loaded in TIBCO Business Studio.


Design - TIBCO ActiveMatrix BusinessWorks - D:\workspace
File Edit Navigate Search Project Run Window Help
Quick Access
Design
Project Explorer
com.tibco.bw.palette.helloworld.design
com.tibco.bw.palette.helloworld.design.feature
com.tibco.bw.palette.helloworld.model
com.tibco.bw.palette.helloworld.model.feature
com.tibco.bw.palette.helloworld.runtime
com.tibco.bw.palette.helloworld.runtime.feature
Fil...
AP...
Ou...
De...
Mo...
D:\TIBCO_HOME\bw\6.3\samples
binding
bw5
core
palette
plugins
policy
scenario
source
Pro...
Pro...
BW...
Co...
0 items
Description
Resource
Path

What to do next

Adding Business Logic

Using Shared Resources

TIBCO ActiveMatrix BusinessWorks Plug-in activities can refer to shared resources. These resources provide centralized configuration and functionality. An activity contains a resource selector field that links the activity with an existing shared resource.

At runtime, the activity can get access to the resource and execute business logic irrespective of how a resource is configured. If a resource is modified, the new values will be in effect in an activity that is linked to that resource instance.

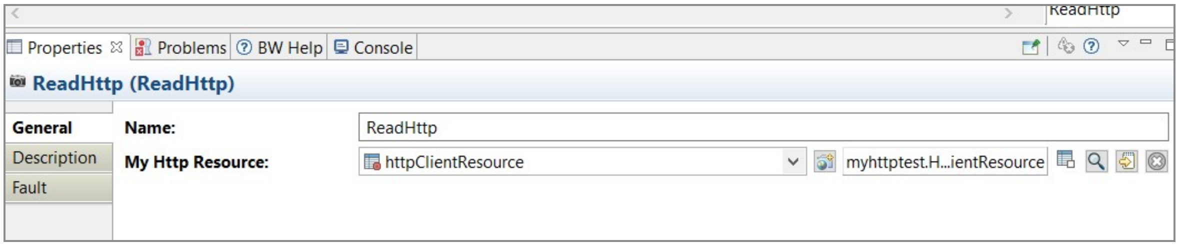
Code Generation

Using TIBCO ActiveMatrix BusinessWorks Plug-in Development Kit, you can establish the links to the shared resources in custom activity. When you define attributes of your activity, you can create an attribute with Property Field from the Control Type list. Select a type of a resource from the Values list. The available standard BusinessWorks resources are:

HTTP Client, JDBC Connection, JMS Connection, TCP Connection, FTL Realm Server Connection, SSL Client Configuration, Java Global Instance

When a PropertyField is added, the generated code will be affected in several ways:

- A resource selector will be added at Design time.



ReadHttp (ReadHttp)

Name: ReadHttp

My Http Resource: httpClientResource myhttptest.H...ientResource

- Generated plug-ins will contain appropriate dependencies in their `MANIFEST.MF` files.
- Runtime code will be generated in such a way that a resource instance is injected into the activity and object instances that are needed to carry out a resource-related action, will be added. Business logic code can utilize the resource.

HTTP Client

Generated Objects

```
@Property(name = "myHttpResource")
public HTTPClientDriverFactory myHttpResource;
private HttpClient myHttpResourceHttpClient;
```

`myHttpResourceHttpClient` is an instance of `org.apache.commons.httpclient.HttpClient`.

Business Logic Example

```
class ReadHttpActivityExecutor implements Runnable {
       private final AsyncActivityCompletionNotifier notifier;
       /**
        * <!-- begin-custom-doc -->
        *
        * <!-- end-custom-doc -->
        * @generated
        */
       @Override
       public void run() {
       if(getActivityLogger().isDebugEnabled()) {
       activityLogger.debug(RuntimeMessageBundle.DEBUG_PLUGIN_ACTIVITY_METHOD_
CALLED
                       ,new Object[] { "Executor run()"
                       ,activityContext.getActivityName()
                       ,activityContext.getProcessName()
                       ,activityContext.getDeploymentUnitName()
                       ,activityContext.getDeploymentUnitVersion() });
               String serializedNode = XMLUtils.serializeNode(inputData,
processContext.getXMLProcessingContext());
       activityLogger.debug(RuntimeMessageBundle.DEBUG_PLUGIN_ACTIVITY_INPUT,
new Object[] {activityContext.getActivityName(), serializedNode});
               }

               try {



                                       // begin-custom-code
                                       // add your own business code here
                               GetMethod getMethod = new GetMethod
("http://jsonplaceholder.typicode.com/posts/1");
       try {
       // myHttpResourceHttpClient is created by PDK for the myHttpResource
field
               myHttpResourceHttpClient.executeMethod(getMethod);
               } catch (IOException e) {
               e.printStackTrace();
               throw new ActivityFault(activityContext, new LocalizedMessage(
               RuntimeMessageBundle.ERROR_OCCURED_RETRIEVE_RESULT, new Object[]
{activityContext.getActivityName(), e.toString()}));
               }
               String responseBodyAsString = getMethod.getResponseBodyAsString();
               System.out.println("Response body: \n" + responseBodyAsString);

                       // end-custom-code



       N output = null;
       SerializableXMLDocument<N> wrapper = new SerializableXMLDocument<N>
(processContext.getXMLProcessingContext(), output);
       notifier.setReady(wrapper);
       } catch (Exception e) {
```

```
       e.printStackTrace();
       notifier.setReady(e);
       }
       }
       }
```

JDBC Connection

Generated Objects

```
@Property(name = "myJDBCConnection")
public DataSource myJDBCConnection;
```

`myJDBCConnection` is an instance of `javax.sql.DataSource` that represents the selected resource. You can use it for database access.

Business Logic Example

```
class MySelectActivityExecutor implements Runnable {
       /**
        * <!-- begin-custom-doc -->
        *
        * <!-- end-custom-doc -->
        * @generated
        */
       @Override
       public void run() {
               if(getActivityLogger().isDebugEnabled()) {
               activityLogger.debug(RuntimeMessageBundle.DEBUG_PLUGIN_ACTIVITY_
METHOD_CALLED
                               ,new Object[] { "Executor run()"
                               ,activityContext.getActivityName()
                               ,activityContext.getProcessName()
                               ,activityContext.getDeploymentUnitName()
                       ,activityContext.getDeploymentUnitVersion() });
               String serializedNode = XMLUtils.serializeNode(inputData,
processContext.getXMLProcessingContext());
       activityLogger.debug(RuntimeMessageBundle.DEBUG_PLUGIN_ACTIVITY_INPUT,
new Object[] {activityContext.getActivityName(), serializedNode});
```

```
        }

                        try {


                                        // begin-custom-code
                                        // add your own business code here

                try {
                // myJDBCConnection is a generated instance to be used for
                // JDBC access
                Connection connection = myJDBCConnection.getConnection();
                java.sql.Statement s = connection.createStatement();
                // execute a query and print results
                ResultSet rs = s.executeQuery("select * from mytable");
                while (rs.next()) {
                // print some fields
                System.out.print(rs.getInt(1) + " ");
                System.out.println(rs.getString(2));
                }

                } catch (SQLException e) {
                throw new ActivityFault(activityContext, e);
                }
                // end-custom-code


        N output = null;
        SerializableXMLDocument<N> wrapper = new SerializableXMLDocument<N>(
                processContext.getXMLProcessingContext(), output);
                notifier.setReady(wrapper);
                } catch (Exception e) {
                e.printStackTrace();
                notifier.setReady(e);
                        }
                }
        }
```

JMS Connection

Generated Objects

```
@Property(name = "myJmsConnection")
public ConnectionFactory myJmsConnection;
```

`myJmsConnection` is an instance of `javax.jms.ConnectionFactory` that represents the selected resource. You can use it to communicate with a JMS server.

Business Logic Example

```
class MyJMSActivityExecutor implements Runnable {
       /**
        * <!-- begin-custom-doc -->
        *
        * <!-- end-custom-doc -->
        * @generated
        */
       @Override
       public void run() {
       if(getActivityLogger().isDebugEnabled()) {
       activityLogger.debug(RuntimeMessageBundle.DEBUG_PLUGIN_ACTIVITY_METHOD_
CALLED
                               ,new Object[] { "Executor run()"
                               ,activityContext.getActivityName()
                               ,activityContext.getProcessName()
                               ,activityContext.getDeploymentUnitName()
                               ,activityContext.getDeploymentUnitVersion() });
               String serializedNode = XMLUtils.serializeNode(inputData,
processContext.getXMLProcessingContext());
       activityLogger.debug(RuntimeMessageBundle.DEBUG_PLUGIN_ACTIVITY_INPUT,
new Object[] {activityContext.getActivityName(), serializedNode});
                       }

                       try {


       // begin-custom-code
       // add your own business code here
       try {
       // create a server connection as per a JMS Connection Resource
       Connection connection = myJmsConnection.createConnection();
       connection.start();
       Session session = connection.createSession(false, Session.AUTO_
ACKNOWLEDGE);
       TemporaryQueue tempQueue = session.createTemporaryQueue();
       MessageProducer producer = session.createProducer(tempQueue);
       TextMessage message = session.createTextMessage();
       // send some messages
               for (int i = 0; i < 3; i++) {
                   message.setText("This is message " + (i + 1));
                   System.out.println("Sending message: " + message.getText());
                   producer.send(message);
               }
       // receive some messages
               MessageConsumer consumer = session.createConsumer(tempQueue);
               for (int i = 0; i < 3; i++) {
               Message receivedMessage = consumer.receive();
               System.out.println("Received message: " + receivedMessage.getBody
(String.class));
               }
               } catch (JMSException e) {
               throw new ActivityFault(activityContext, e);
                       }
                                       // end-custom-code



               N output = null;
       SerializableXMLDocument<N> wrapper = new SerializableXMLDocument<N>
(processContext.getXMLProcessingContext(), output);
                               notifier.setReady(wrapper);
                       } catch (Exception e) {
                               e.printStackTrace();
                               notifier.setReady(e);
                       }
               }
       }
```

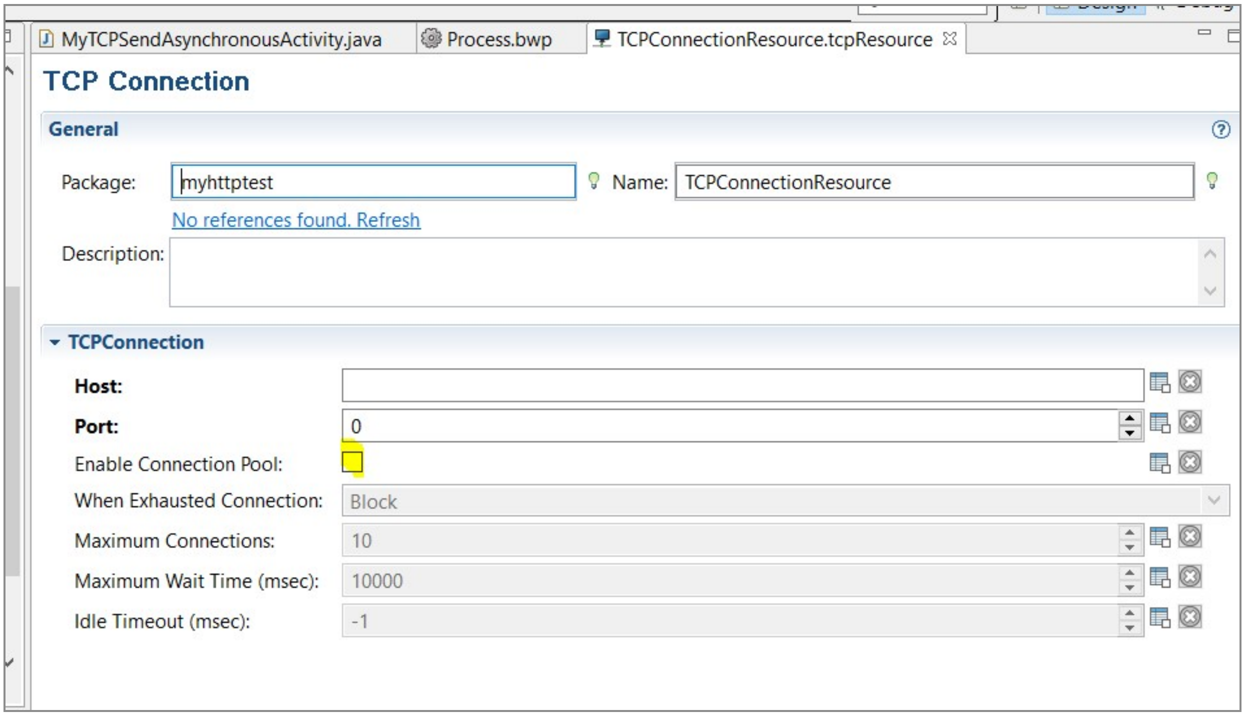
TCP Connection

Generated Objects

```
@Property(name = "MyTCPConnection")
public TCPConnectionAdminObject myTCPConnection;
```

`myTCPConnection` is an instance of `com.tibco.bw.sharedresource.tcpconnection.runtime.TCPConnectionAdminObject` that represents the selected resource. You can use it to obtain configuration attributes from the resource.

Important: A TCP Connection pooling (Enable Connection Pool under TCPConnection section of the resource configuration) cannot be used from a custom activity. You can only obtain the resource configuration values, but all socket access needs to be implemented in the activity.


MyTCPSendAsynchronousActivity.java
Process.bwp
TCPConnectionResource.tcpResource
TCP Connection
General
TCPConnection

Package: myhttptest

Name: TCPConnectionResource

No references found. Refresh

Description:

Host:

Port: 0

Enable Connection Pool: □

When Exhausted Connection: Block

Maximum Connections: 10

Maximum Wait Time (msec): 10000

Idle Timeout (msec): -1

Business Logic Example

```
class MyTCPSendActivityExecutor implements Runnable {

       /**
        * <!-- begin-custom-doc -->
        *
        * <!-- end-custom-doc -->
        * @generated
        */
       @Override
       public void run() {
       if(getActivityLogger().isDebugEnabled()) {
       activityLogger.debug(RuntimeMessageBundle.DEBUG_PLUGIN_ACTIVITY_METHOD_
CALLED
               ,new Object[] { "Executor run()"
               ,activityContext.getActivityName()
               ,activityContext.getProcessName()
               ,activityContext.getDeploymentUnitName()
               ,activityContext.getDeploymentUnitVersion() });

String serializedNode = XMLUtils.serializeNode
(inputData,processContext.getXMLProcessingContext());
       activityLogger.debug(RuntimeMessageBundle.DEBUG_PLUGIN_ACTIVITY_INPUT,
new Object[] {activityContext.getActivityName(), serializedNode});
                       }

                       try {


               // begin-custom-code
               // add your own business code here

               Socket s = null;
               try {
               // get the destination host and port from the resource
               String host = myTCPConnection.getHost();
               Integer port = Integer.valueOf(myTCPConnection.getPort());
               // create a connected socket
               s = new Socket(host, port);
               // send a some data
               OutputStream os = s.getOutputStream();
               os.write(new String("Hello World!").getBytes());
       }       catch (IOException e) {
                       throw new ActivityFault(activityContext, e);
                       } finally
                       {
                       if (s != null){
                       s.close();
                       }
                       }

                       // end-custom-code

                       N output = null;
                               SerializableXMLDocument<N> wrapper = new
SerializableXMLDocument<N>(processContext.getXMLProcessingContext(),
output);
                       notifier.setReady(wrapper);
               }                       catch (Exception e) {
                                       e.printStackTrace();
                                       notifier.setReady(e);
                       }
               }
       }
```

FTL Realm Server Connection

Generated Objects

```
@Property(name = "MyFtlServerRealm")
public FTLRealmServerConnectionResource myFtlServerRealm;
private Realm myFtlServerRealmFtlRealm;
```

`myFtlServerRealmFtlRealm` is an instance of `com.tibco.bw.sharedresource.ftl.rumntime`. `FTLRealmServerConnectionResource` and `myFtlServerRealmFtlRealm` are instances of `com.tibco.ftl.Realm`. You can use these objects to communicate with FTL.

Business Logic Example

```
class MyFtlSendActivityExecutor implements Runnable {
       /**
        * <!-- begin-custom-doc -->
        *
        * <!-- end-custom-doc -->
        * @generated
        */
       @Override
       public void run() {
       if(getActivityLogger().isDebugEnabled()) {
       activityLogger.debug(RuntimeMessageBundle.DEBUG_PLUGIN_ACTIVITY_METHOD_
CALLED
               ,new Object[] { "Executor run()"
               ,activityContext.getActivityName()
               ,activityContext.getProcessName()
               ,activityContext.getDeploymentUnitName()
               ,activityContext.getDeploymentUnitVersion() });
               String serializedNode = XMLUtils.serializeNode(inputData,
processContext.getXMLProcessingContext());
       activityLogger.debug(RuntimeMessageBundle.DEBUG_PLUGIN_ACTIVITY_INPUT,
new Object[] {activityContext.getActivityName(), serializedNode});
                       }

                       try {


               // begin-custom-code
               // add your own business code here
               String endpoint = myFtlServerRealm.getApplicationName();
               EventQueue eventQueue = null;
               Publisher pub = null;
               Message msg = null;
               try {
               eventQueue = myFtlServerRealmFtlRealm.createEventQueue
(FTL.createProperties());
               pub = myFtlServerRealmFtlRealm.createPublisher(endpoint);
               msg = myFtlServerRealmFtlRealm.createMessage(null);
               msg.setString("message", "Hello World!");
               pub.send(msg);
               eventQueue.dispatchNow();
               } catch (FTLException e) {
               throw new ActivityFault(activityContext, e);
               } finally {
               if (eventQueue != null) {
               eventQueue.destroy();
               }
               }
               // end-custom-code


               N output = null;
               SerializableXMLDocument<N> wrapper = new SerializableXMLDocument<N>
(processContext.getXMLProcessingContext(), output);
               notifier.setReady(wrapper);
               } catch (Exception e) {
               e.printStackTrace();
               notifier.setReady(e);
                       }
               }
       }
```

SSL Client Configuration

Generated Objects

```
        @Property(name = "MySSLClientConfiguration")
public SSLClientResource mySSLClientConfiguration;
private IdentityTrust mySSLClientConfigurationIdentityTrust;
```

`mySSLClientConfigurationIdentityTrust` is an instance of `com.tibco.trinity.runtime.base.provider.identity.IdentityTrust`. You can use to access resources.

Business Logic Example

```
class MySSLActivityExecutor implements Runnable {
       @Override
       public void run() {
       if(getActivityLogger().isDebugEnabled()) {
       activityLogger.debug(RuntimeMessageBundle.DEBUG_PLUGIN_ACTIVITY_METHOD_
CALLED
               ,new Object[] { "Executor run()"
               ,activityContext.getActivityName()
               ,activityContext.getProcessName()
               ,activityContext.getDeploymentUnitName()
               ,activityContext.getDeploymentUnitVersion() });
               String serializedNode = XMLUtils.serializeNode(inputData,
processContext.getXMLProcessingContext());
       activityLogger.debug(RuntimeMessageBundle.DEBUG_PLUGIN_ACTIVITY_INPUT,
new Object[] {activityContext.getActivityName(), serializedNode});
               }

       try {



               // begin-custom-code
               // add your own business code here

               // print aliases from the trust store
               try {
               Enumeration<String> aliases = mySSLClientConfigurationIdentityTrust
               .getTrustStore().aliases();
               while (aliases.hasMoreElements()) {
               System.out.println("***" + aliases.nextElement());
               }
               } catch (KeyStoreException e) {
               // TODO Auto-generated catch block
               throw new ActivityFault(activityContext, e);
               }
               // end-custom-code


               N output = null;
               SerializableXMLDocument<N> wrapper = new SerializableXMLDocument<N>
(processContext.getXMLProcessingContext(), output);
               notifier.setReady(wrapper);
               } catch (Exception e) {
               e.printStackTrace();
               notifier.setReady(e);
               }
               }
       }
```

JAVA Global Instance

Generated Objects

```
        @Property(name = "MyJavaGlobalInstance")
public JavaGlobalInstanceResource myJavaGlobalInstance;
```

`myJavaGlobalInstance` is an instance of `com.tibco.bw.sharedresource.java.runtime.JavaGlobalInstanceResource`. You can use it to get the java object contained in the referenced JAVA Global Instance resource. You can access that object via Java reflection.

Business Logic Example

```
class CallMyMethodActivityExecutor implements Runnable {
       @Override
       public void run() {
       if(getActivityLogger().isDebugEnabled()) {
       activityLogger.debug(RuntimeMessageBundle.DEBUG_PLUGIN_ACTIVITY_METHOD_
CALLED
               ,new Object[] { "Executor run()"
               ,activityContext.getActivityName()
               ,activityContext.getProcessName()
               ,activityContext.getDeploymentUnitName()
               ,activityContext.getDeploymentUnitVersion() });
               String serializedNode = XMLUtils.serializeNode(inputData,
processContext.getXMLProcessingContext());
       activityLogger.debug(RuntimeMessageBundle.DEBUG_PLUGIN_ACTIVITY_INPUT,
new Object[] {activityContext.getActivityName(), serializedNode});
               }

       try {


                       // begin-custom-code
                       // add your own business code here

               try {
               // get the object instance
               Object object = myJavaGlobalInstance.getDeclaredClassObject();
               // use reflection to call getHello() method
               Method method = object.getClass().getDeclaredMethod("getHello");
```

```
                Object result = method.invoke(object);
                System.out.println((String) result);
                } catch (Exception e) {
                throw new ActivityFault(activityContext, e);
                }

                // end-custom-code



        N output = null;
        SerializableXMLDocument<N> wrapper = new SerializableXMLDocument<N>
(processContext.getXMLProcessingContext(), output);
                notifier.setReady(wrapper);
                } catch (Exception e) {
                e.printStackTrace();
                notifier.setReady(e);
                        }
                }
        }
```

Activity Types

Activities are the individual units of work in a process.

Activities generally interact with an external system and perform a task. Activities that are used to connect the same external application can be grouped together.

BusinessWorks Plug-in Development Kit supports the following types of activities:

Synchronous Activities

Synchronous activities are *blocking* and they block the execution of the process until the activity task is completed.

Asynchronous Activities

Asynchronous activities are *non-blocking* and they perform a task asynchronously without blocking the execution of a process.

Process Starter Activities

Process starter activities are configured to react to events and trigger the execution of a process when an event occurs. Process starter activities only have outputs in addition to other general configurations.

Signal-In Activities

Signal-In activities wait for an asynchronous event in a process and then proceed to execute the process instance. Signal-In activities require conversations to be configured and are always blocking the execution.

For more details about the conversation, see the TIBCO ActiveMatrix BusinessWorks 6 documentation.

Abstract Activities

Abstract activities are not TIBCO ActiveMatrix BusinessWorks 6 activities. BusinessWorks Plug-in Development Kit uses this kind of activity to share common configurations. Abstract activities define the common properties that are shared by all the activities in a plug-in and require no input and output in addition to other general configurations.

Creating Schema with XSD/WSDL

BusinessWorks Plug-in Development Kit can parse the input, output, and fault schema from an XSD or a WSDL file.

Procedure

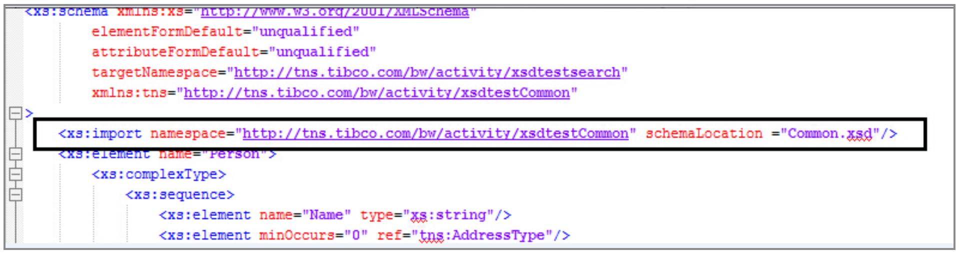
1. In the Define Activities, Process Starters and Signal-In dialog box, select **XSD/WSDL** from the **Resource schema type** list.
2. Click **Browse** to locate a predefined XSD or a WSDL file.
3. Select a predefined element that you want to use as the input, output, or fault schema. If you want to define multiple faults, you have to add the fault element one by one.

   The XSD schema in the selected XSD or WSDL file must conform to the following rules:

   - Contains a namespace.
   - Does not contain an `include` element.
   - If the XSD schema uses the `import` element, ensure the `schemaLocation`

element is also used and the value of the `schemaLocation` element is the relative path of the imported schema.

- The value of the `elementFormDefault` attribute is unqualified.



```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
        elementFormDefault="unqualified"
        attributeFormDefault="unqualified"
        targetNamespace="http://tns.tibco.com/bw/activity/xsdtestsearch"
        xmlns:tns="http://tns.tibco.com/bw/activity/xsdtestCommon"
>
    <xs:import namespace="http://tns.tibco.com/bw/activity/xsdtestCommon" schemaLocation ="Common.xsd"/>
    <xs:element name="Person">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="Name" type="xs:string"/>
                <xs:element minOccurs="0" ref="tns:AddressType"/>
```

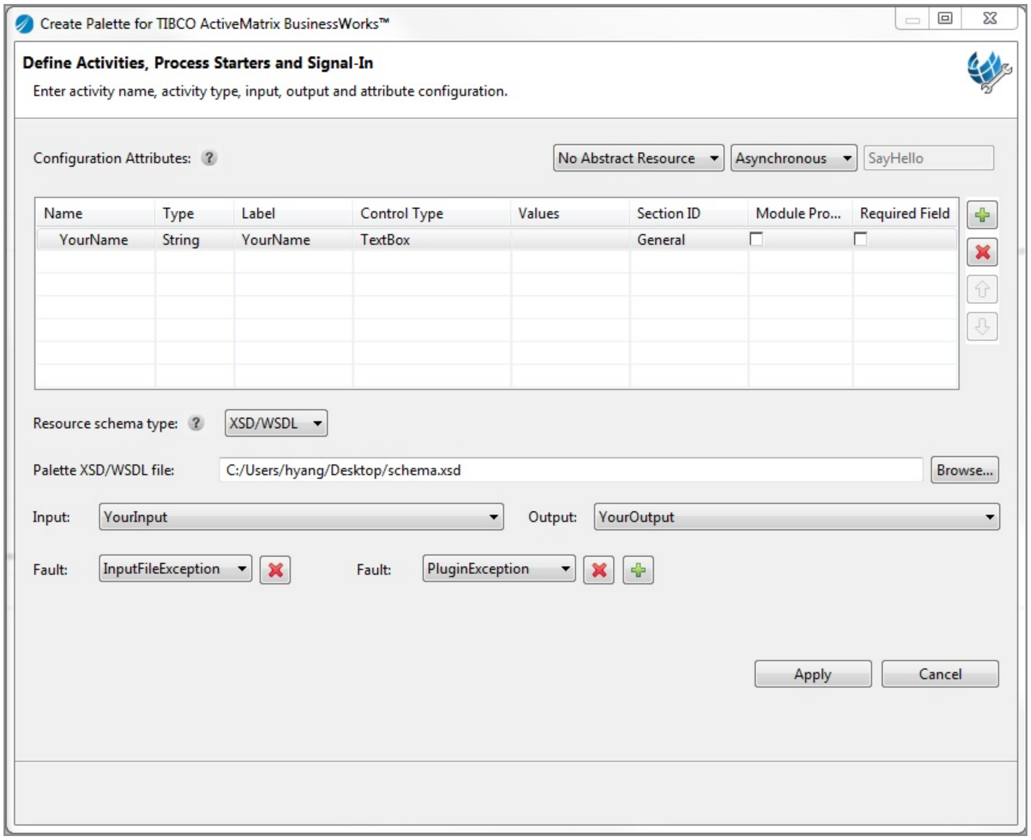
Use the following XSD file as an example:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<schema xmlns:tns="http://www.tibco.com/namespaces/tnt/plugins/example"
xmlns="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.tibco.com/namespaces/tnt/plugins/example">
<element name="YourInput" type="tns:YourInputType"/>
<complexType name="YourInputType">
    <sequence>
        <element maxOccurs="1" minOccurs="0" name="YourName"
type="string"/>
     </sequence>
</complexType>

<element name="YourOutput" type="tns:YourOutputType"/>
<complexType name="YourOutputType">
    <sequence>
        <element maxOccurs="1" minOccurs="0" name="Output"
type="string"/>
    </sequence>
</complexType>

<element name="PluginException" type="tns:PluginExceptionType"/>
<complexType name="PluginExceptionType">
    <sequence>
        <element maxOccurs="1" minOccurs="0"
name="PluginfileExceptionMessage" type="string"/>
    </sequence>
</complexType>

<element name="InputFileException" type="tns:InputFileExceptionType"/>
<complexType name="InputFileExceptionType">
    <sequence>
        <element maxOccurs="1" minOccurs="1" name="fileInfo"
```

```
type="tns:fileInfoType">
        </element>
    </sequence>
</complexType>
<complexType name="fileInfoType">
    <sequence>
        <element maxOccurs="1" minOccurs="1" name="fullName"
type="string">
        </element>
        <element maxOccurs="1" minOccurs="1" name="fileName"
type="string">
        </element>
        <element maxOccurs="1" minOccurs="1" name="location"
type="string">
        </element>
        <element maxOccurs="1" minOccurs="0" name="configuredFileName"
type="string">
        </element>
        <element maxOccurs="1" minOccurs="1" name="type" type="string">
        </element>
        <element maxOccurs="1" minOccurs="1" name="readProtected"
type="boolean">
        </element>
        <element maxOccurs="1" minOccurs="1" name="writeProtected"
type="boolean">
        </element>
        <element maxOccurs="1" minOccurs="1" name="size" type="long">
        </element>
        <element maxOccurs="1" minOccurs="1" name="lastModified"
type="string"/>
    </sequence>
</complexType>
</schema>
```

When adding and configuring activities in the Define Activities, Process Starters and Signal-In dialog box, use the example XSD file to configure the activity schema:

- Select the `YourInput` element as the activity input.
- Select the `YourOutput` element as the output.
- Select the `InputFileException` element as one fault.
- Select the `PluginException` element as another fault.



Create Palette for TIBCO ActiveMatrix BusinessWorks™

Define Activities, Process Starters and Signal-In

Enter activity name, activity type, input, output and attribute configuration.

Configuration Attributes: No Abstract Resource | Asynchronous | SayHello

Name	Type	Label	Control Type	Values	Section ID	Module Pro...	Required Field
YourName	String	YourName	TextBox		General	□	□

Resource schema type: XSD/WSDL

Palette XSD/WSDL file: C:/Users/hyang/Desktop/schema.xsd

Input: YourInput

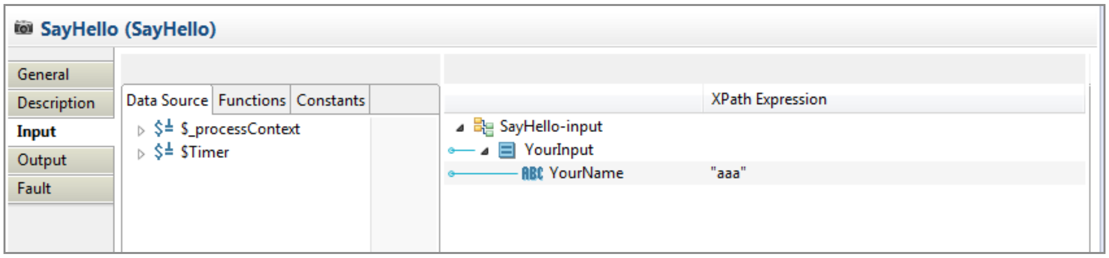
Output: YourOutput

Fault: InputFileException

Fault: PluginException

After generating the activity, click the **Input**, **Output**, and **Fault** tabs accordingly to check the schema:

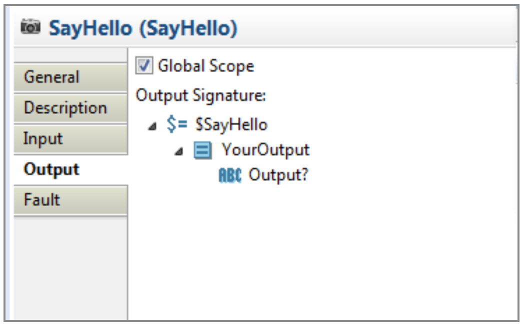
The following example is the generated input schema based on the selected XSD file:



SayHello (SayHello)

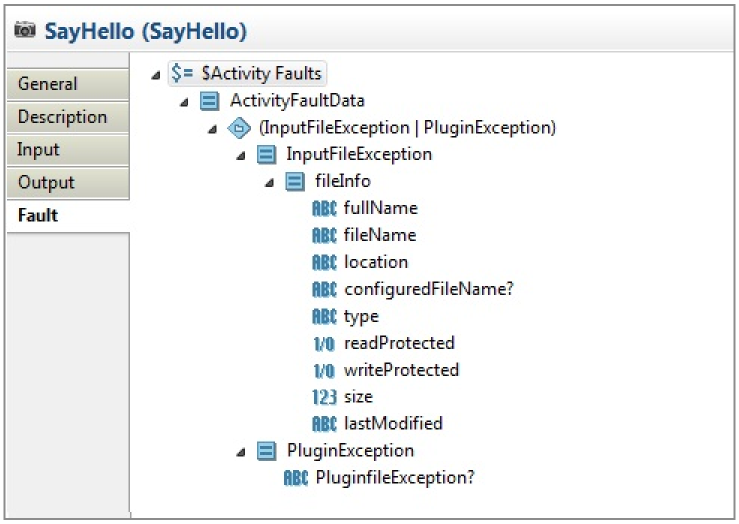
	XPath Expression
SayHello-input	
YourInput	
YourName	"aaa"

The following example is the generated output schema based on the selected XSD file:



☑ Global Scope

The following example is the generated fault schema based on the selected XSD file:



Creating Schema with XSD Editor

You can use the XSD editor to configure the input, output, and fault schema of an activity.

When configuring an activity, the root element of input and output is created by default. You can add primitive elements, anonymous complex elements, any elements, and attributes to this root element.

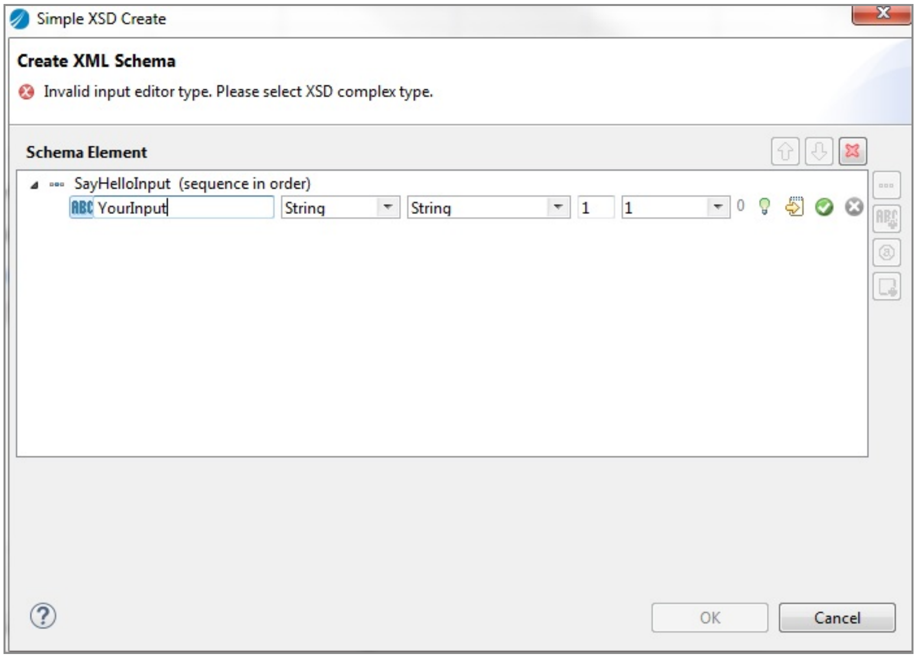
Note: If you want to specify multiple exceptions, add the fault schema one by one.

Note: The Choose Element Type function and the number of reference function are not supported when the element type is `anyType`.

Take the input of the SayHello activity as an example.

Procedure

1. In the Simple XSD Create dialog box, click **SayHelloInput**.
2. Click ABC to add a primitive element.
3. Click the added primitive element and enter YourInput as the element name. Click ✓ to save your configurations.



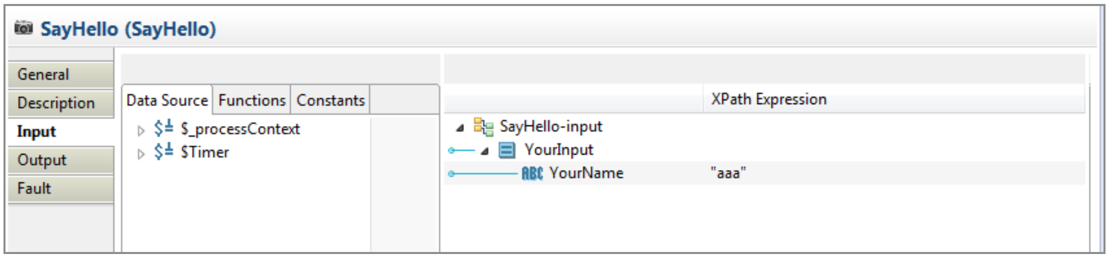
Create XML Schema

Invalid input editor type. Please select XSD complex type.

4. Click the **SayHelloInput (sequence in order)** root element, and then click **OK** to apply the configurations.

Result

After generating the activity, the input structure of the SayHello activity is as shown in the following figure:



Adding Business Logic

At run time, you can add your own logic for each activity.

- For a synchronous activity, add your business logic in the [ActivityName]SynchronousActivity class.
- For an asynchronous activity, add your business logic in the [ActivityName]AsynchronousActivity class.
- For a process starter activity or a signal-in activity, add your business logic in the [ActivityName]EventSource class.

Before you begin

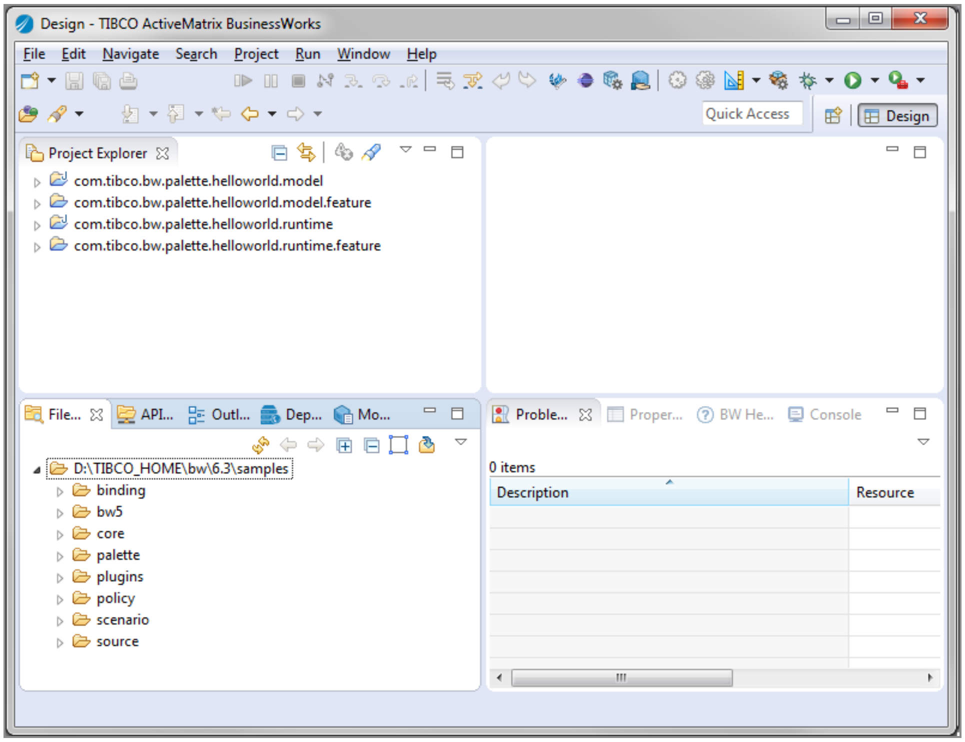
Ensure that you have generated a palette and activities of the plug-in, as described in Defining a Palette and Adding and Configuring Activities.

Procedure

1. In the parent TIBCO Business Studio with the running platform selected, click **Run > Run Configurations**.
2. In the "Create, manage, and run configurations" dialog box, double-click **Eclipse Application** in the left panel to create a new Eclipse application.

   You can also use an existing Eclipse application.
3. Click **Debug** to launch a child TIBCO Business Studio configured with the bw-runtime platform.
4. In the child TIBCO Business Studio, click **File > Import** to import the model and runtime bundles and features:
   a. In the Select dialog box, expand the **General** folder and select **Existing Projects into Workspace**. Click **Next**.
   b. In the Import Projects dialog box, click **Browse** to locate the project folder that contains the plug-in project.
   c. Click **Deselect All**, and then select the runtime and model bundles and features. Click **Finish**.

   The runtime and model bundles are loaded in the child TIBCO Business Studio.


Design - TIBCO ActiveMatrix BusinessWorks
File Edit Navigate Search Project Run Window Help
Quick Access
Design
Project Explorer
com.tibco.bw.palette.helloworld.model
com.tibco.bw.palette.helloworld.model.feature
com.tibco.bw.palette.helloworld.runtime
com.tibco.bw.palette.helloworld.runtime.feature
File...
API...
Outl...
Dep...
Mo...
D:\TIBCO_HOME\bw\6.3\samples
binding
bw5
core
palette
plugins
policy
scenario
source
Proble...
Proper...
BW He...
Console
0 items
Description
Resource

5. Click the runtime bundle, and then click the **src** folder to add the business logic.

   Plain Ordinary Java Object (POJO) class is supported for implementation of source code. You can assign values to POJO classes, and output is automatically generated.

   See Runtime Class Specification and Runtime Bundle for more details.

6. If you have specified multiple faults when configuring activities at design time, BusinessWorks Plug-in Development Kit only generates the first fault schema at run time. You have to add other faults manually at run time:

   a. Expand the runtime bundle and find the **src** folder.

   b. Expand the **com.*company_name*.bw.palette.*palette_name*.runtime.fault** package.

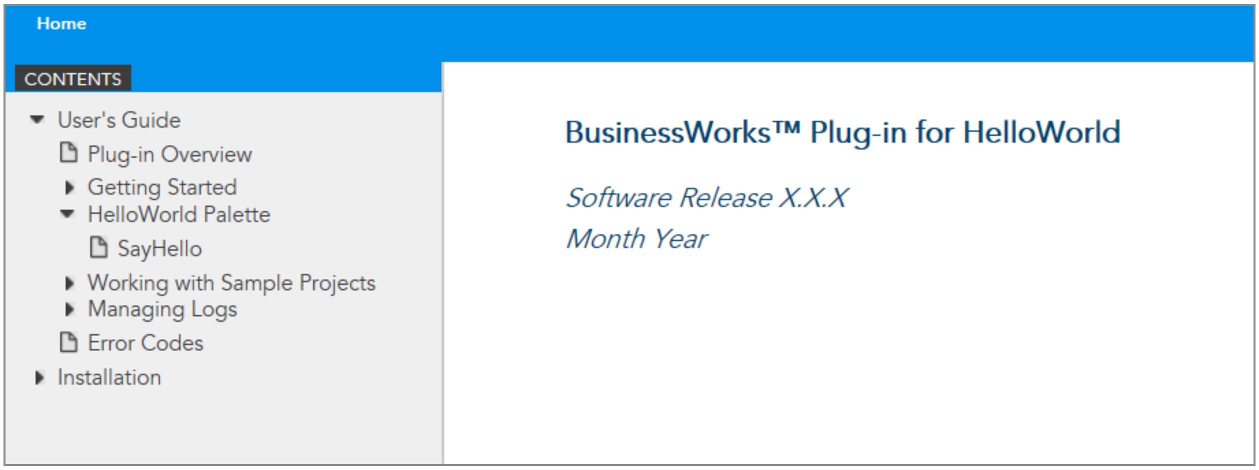
   c. In the **com.*company_name*.bw.palette.*palette_name*.runtime.fault** package, make a copy of the `fault_nameFault.java` file.

   d. Open the copied file and build the other faults by using the `private <N, A> N constructErrData ()` method and the `public QName getFaultElementQName ()` method.

Creating Documentation

BusinessWorks Plug-in Development Kit generates a documentation template based on the activities that you have created. You can update the documentation template to guide your plug-in users, and configure the documentation for online help.

A doc folder is generated under the project folder after generating the plug-in activities. Open the `index.html`, the created documentation template is displayed.



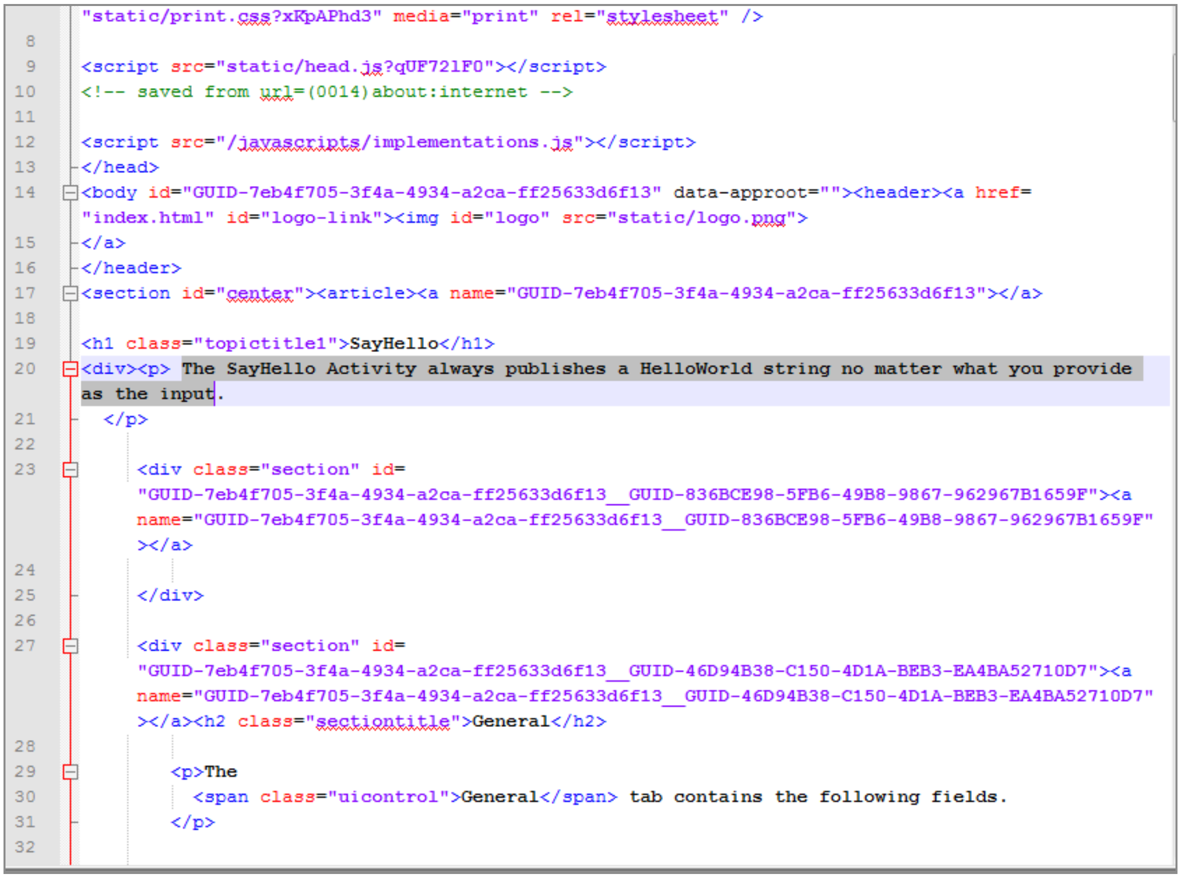
BusinessWorks™ Plug-in for HelloWorld

Software Release X.X.X

Month Year

The documentation template contains a user's guide and an installation guide. In the user's guide, BusinessWorks Plug-in Development Kit automatically creates reference topics for the created palette and activities.

You can update the documentation template based on your requirements. For example, add a description for the SayHello activity.

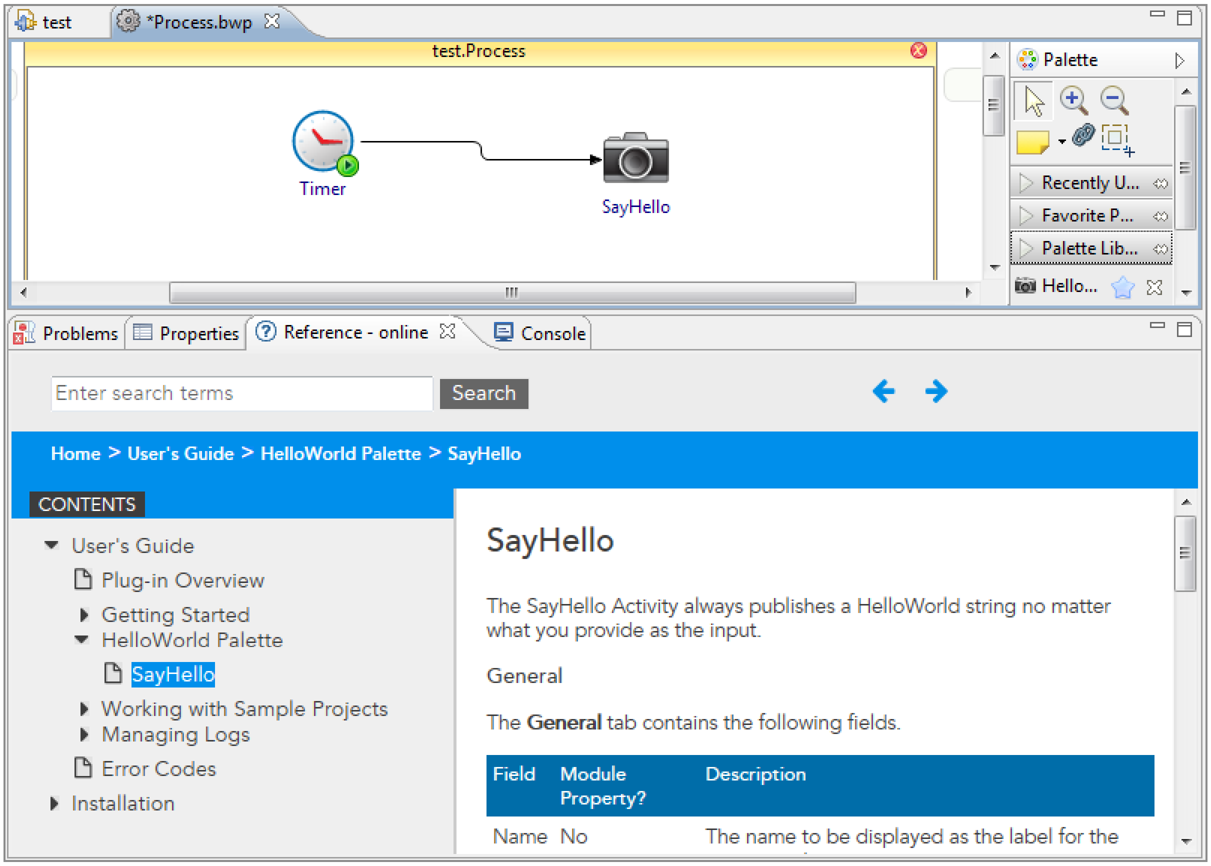


```
"static/print.css?xKpAPhd3" media="print" rel="stylesheet" />

<script src="static/head.js?qUF721F0"></script>
<!-- saved from url=(0014)about:internet -->

<script src="/javascripts/implementations.js"></script>
</head>
<body id="GUID-7eb4f705-3f4a-4934-a2ca-ff25633d6f13" data-approot=""><header><a href=
"index.html" id="logo-link"><img id="logo" src="static/logo.png">
</a>
</header>
<section id="center"><article><a name="GUID-7eb4f705-3f4a-4934-a2ca-ff25633d6f13"></a>

<h1 class="topictitle1">SayHello</h1>
<div><p> The SayHello Activity always publishes a HelloWorld string no matter what you provide
as the input.
  </p>

     <div class="section" id=
     "GUID-7eb4f705-3f4a-4934-a2ca-ff25633d6f13__GUID-836BCE98-5FB6-49B8-9867-962967B1659F"><a
     name="GUID-7eb4f705-3f4a-4934-a2ca-ff25633d6f13__GUID-836BCE98-5FB6-49B8-9867-962967B1659F"
     ></a>

     </div>

     <div class="section" id=
     "GUID-7eb4f705-3f4a-4934-a2ca-ff25633d6f13__GUID-46D94B38-C150-4D1A-BEB3-EA4BA52710D7"><a
     name="GUID-7eb4f705-3f4a-4934-a2ca-ff25633d6f13__GUID-46D94B38-C150-4D1A-BEB3-EA4BA52710D7"
     ></a><h2 class="sectiontitle">General</h2>

        <p>The
          <span class="uicontrol">General</span> tab contains the following fields.
        </p>

```

After updating the documentation, you can enable the online help function for the created palette, as described in How to Add Online Help for a Palette.

After configuring the online help settings, in the process editor, right-click the activity and click **Help > Reference Page**. The help content for the activity is displayed in the Reference view.

⚠ **Caution:** Any changes to the documentation files will be lost if you edit the palette and re-generate the source code. Save the modified version and replace the re-generated files with it.



SayHello

The SayHello Activity always publishes a HelloWorld string no matter what you provide as the input.

General

The **General** tab contains the following fields.

Field	Module Property?	Description
Name	No	The name to be displayed as the label for the

Editing a Plug-in

You can update the activity attributes and schema, delete an activity, and add new activities.

Before editing a plug-in, see Merging Code for more details about how BusinessWorks Plug-in Development Kit merges the code.

Procedure

1. Open TIBCO Business Studio in one of the following ways:

   - Microsoft Windows: click **Start > All Programs > TIBCO > *TIBCO_HOME* > TIBCO Business Studio *version_number* > Studio for Designers**.
   - Linux: run the TIBCO Business Studio executable located in the `TIBCO_HOME/studio/version_number/eclipse` directory.
   - Mac OS: run the TIBCO Business Studio executable located in the `TIBCO_HOME/studio/version_number/eclipse` directory.

2. Open the BusinessWorks Plug-in Development Kit wizard in one of the following ways:

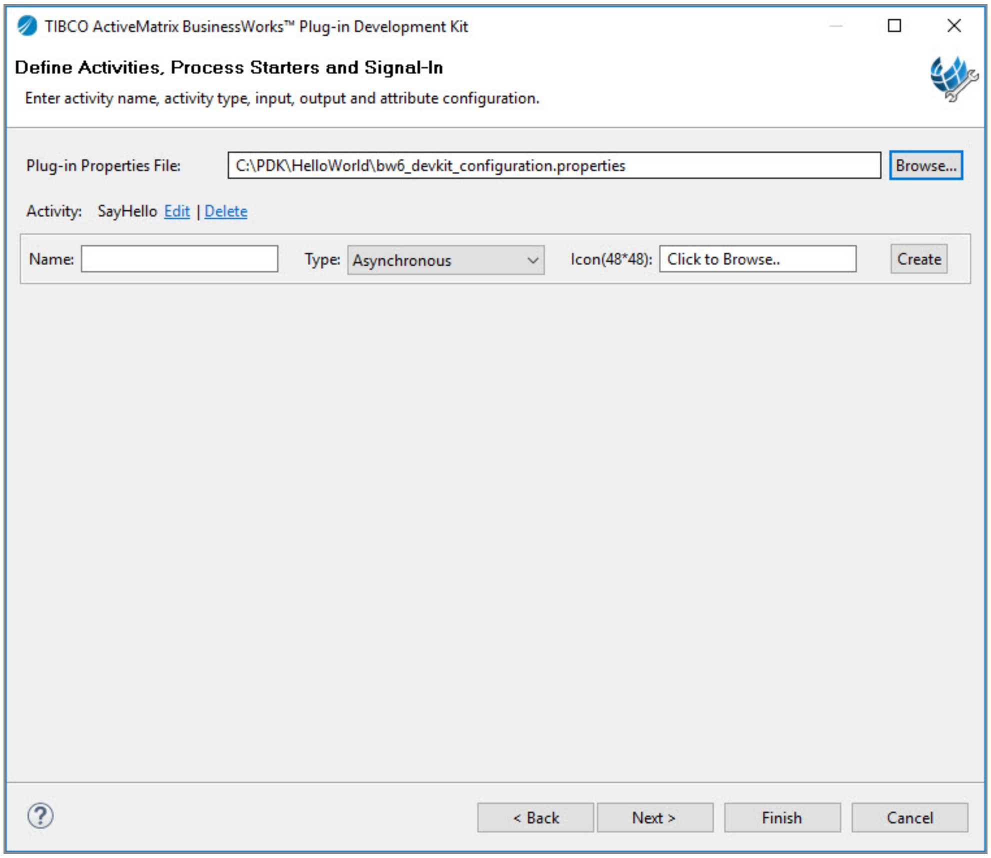
   - From the menu, click **File > New > Create a new or modify BW Plug-in Project**.
   - On the toolbar, click **Create a new or modify BW Plug-in Project**.
   - Press Alt+T.

3. Click **Edit a BW6 Palette**.

4. In the Define Activities, Process Starters and Signal-In dialog box, click **Browse** next to the **Plug-in Properties File** field to locate the `bw6_devkit_configuration.properties` file that contains the plug-in configurations.

   The `bw6_devkit_configuration.properties` file is located in the project folder of the plug-in.

   The activities of the plug-in are displayed in the Define Activities, Process Starters and Signal-In dialog box.



TIBCO ActiveMatrix BusinessWorks™ Plug-in Development Kit

Define Activities, Process Starters and Signal-In

Enter activity name, activity type, input, output and attribute configuration.

Plug-in Properties File: C:\PDK\HelloWorld\bw6_devkit_configuration.properties

Activity: SayHello

Name:

Type: Asynchronous

Icon(48*48): Click to Browse..

5. Click **Edit** to update an activity.

   See Editing an Activity for more details.

6. Click **Delete** to delete an activity.

   See Deleting an Activity or more details.

7. In the **Name** field, enter a name of the activity to be added and select an activity type, and then click **Go** to configure the activity.

   See Adding and Configuring Activities for more details.

8. Click **Finish** to generate activities.

9. To add business logic, see Adding Business Logic.

Merging Code

BusinessWorks Plug-in Development Kit regenerates the source code, and merges the added business logic to the source code when editing a plug-in.

Before editing the plug-in, ensure that you add your own code between the //begin-custom-code tag and the //end-custom-code tag. Otherwise, the added business logic is deleted when regenerating the source code.

Note: The custom tags are settled. You cannot add new custom tags and the code in the new custom tag cannot be merged.

For example, if you add your logic as follows, the added code is deleted after regenerating the code.

```
public EObject createInstance() {
        Get activity = Example2Factory.eINSTANCE.createGet();
        activity.setAttrName("b");
        return activity;
```

You have to add your logic between the //begin-custom-code tag and the //end-custom-code tag as follows:

```
 public EObject createInstance() {
        Get activity = Example2Factory.eINSTANCE.createGet();
        // begin-custom-code
        activity.setAttrName("b");
        // end-custom-code
        return activity;
```

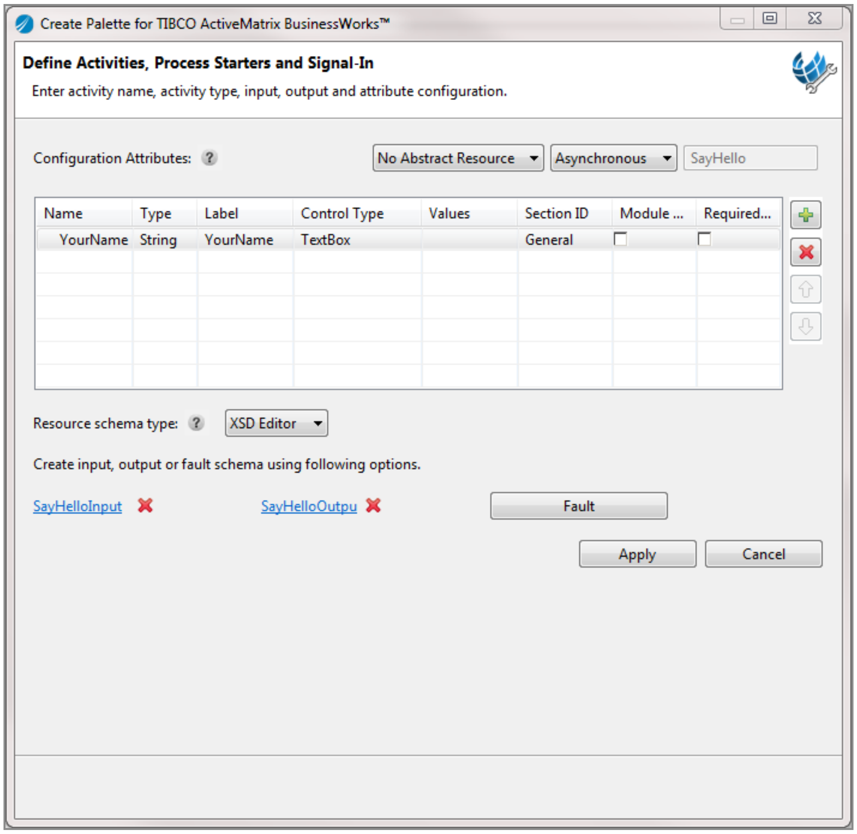
Editing an Activity

You can update the activity attributes and update the activity input, output, and fault schema. When you update an activity, BusinessWorks Plug-in Development Kit regenerates the source code of the activity.

When clicking **Edit**, the activity configurations are displayed in the Define Activities, Process Starters and Signal-In dialog box.

Important: It is not a good practice to change the activity type from Asynchronous/Synchronous to Signal-In/Process Starter and vice versa.

See Updating Schema for more information about how to update the input, output, or fault schema of an activity.



Create Palette for TIBCO ActiveMatrix BusinessWorks™

Define Activities, Process Starters and Signal-In

Enter activity name, activity type, input, output and attribute configuration.

Configuration Attributes: No Abstract Resource | Asynchronous | SayHello

Name	Type	Label	Control Type	Values	Section ID	Module ...	Required...
YourName	String	YourName	TextBox		General	☐	☐

Resource schema type: XSD Editor

Create input, output or fault schema using following options.

Updating Schema

BusinessWorks Plug-in Development Kit provides you a `[ActivityName]Signature` class to update the input and output schema after generating codes of activities.

Depending on the way that you select to generate the input, output, and fault schema, you can choose different methods to update the activity schema:

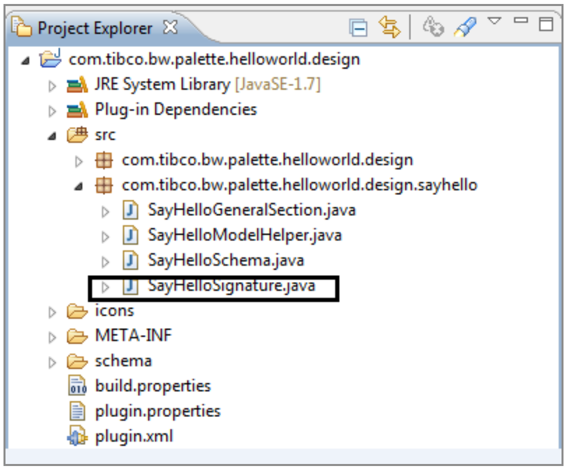
- If you use an XSD file or a WSDL file to define the input, output, and fault schema, you can update the XSD file or the WSDL file accordingly and regenerate the palette.
- If you use the default input, output, and fault schema generated by BusinessWorks Plug-in Development Kit, you can use the `[ActivityName]Signature` class to update the schema accordingly.
- If you use the XSD Editor to define the input, output, and fault schema, you can use the XSD Editor to update the schema accordingly.

The following example shows how to use the `[ActivityName]Signature` class to update the input of the SayHello activity to `YourName` and `Email`.

Procedure

1. In the parent TIBCO Business Studio with the running platform, click the design-time bundle.

2. Open the `SayHelloSignature.java` file that is located in the **src** folder.


Project Explorer
com.tibco.bw.palette.helloworld.design
JRE System Library [JavaSE-1.7]
Plug-in Dependencies
src
com.tibco.bw.palette.helloworld.design
com.tibco.bw.palette.helloworld.design.sayhello
SayHelloGeneralSection.java
SayHelloModelHelper.java
SayHelloSchema.java
SayHelloSignature.java
icons
META-INF
schema
build.properties
plugin.properties
plugin.xml

3. Find the `public XSDElementDeclaration getInputType()` method and enter YourName and Email as the input elements. Save your configurations.

   The following code example is generated when selecting **Java Code** from **Resource schema type** list:

```
public XSDElementDeclaration getInputType(final Configuration
config) {
    XSDElementDeclaration inputType = null;
    String namespace = createNamespace(new Object[] { TARGET_NS,
config, "Input" });
    XSDSchema inputSchema = XSDUtility.createSchema(namespace);

    XSDModelGroup rootInput = XSDUtility.addComplexTypeElement
(inputSchema, "ActivityInputType", "ActivityInputType",
XSDCompositor.SEQUENCE_LITERAL);
    XSDUtility.addSimpleTypeElement(rootInput, "YourName",
"string", 1, 1);
    XSDUtility.addSimpleTypeElement(rootInput, "Email", "string",
0, 1);

    inputSchema = compileSchema(inputSchema);
    inputType = inputSchema.resolveElementDeclaration
(getInputTypeRootName());
```

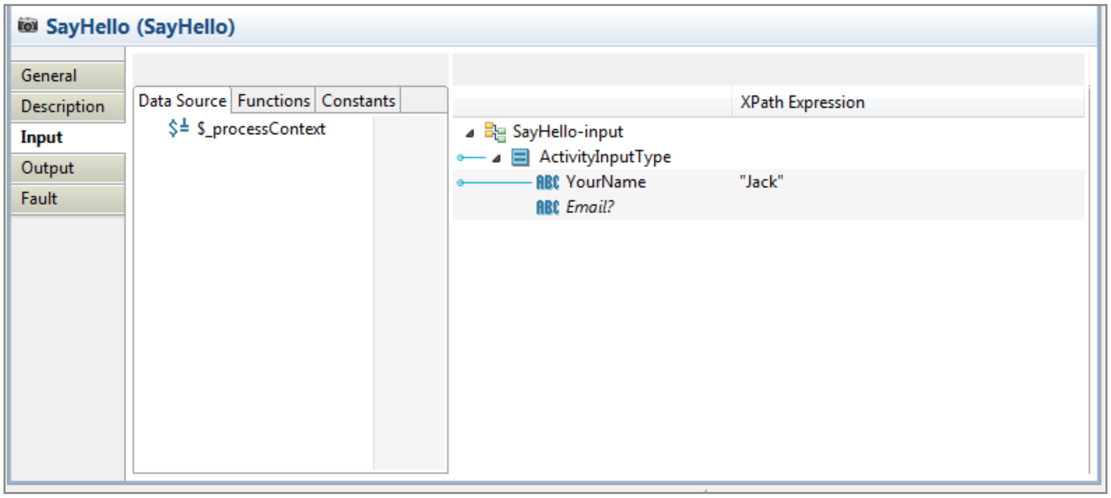
```
        // begin-custom-code
        // end-custom-code
        return inputType;
    }
```

4. From the menu, click **Run > Run Configurations**, and then double-click **Eclipse Application** in the left panel.
5. Click **Debug** to launch a child TIBCO Business Studio.

   Another TIBCO Business Studio workbench is launched.

Result

In the child TIBCO Business Studio, check the input of the SayHello activity. The input elements, `YourName` and `Email` are displayed.



Adding an Activity

You can add one or more activities to the palette by selecting the **Edit Existing BW6 Palette** option.

In the Define Activities, Process Starters and Signal-In dialog box, enter a name of the activity that you want to add, select the activity type, and provide an activity icon. Click **Go** to configure the activity.

See Adding and Configuring Activities for more details.

Deleting an Activity

You can delete existing activities by selecting the **Edit Existing BW6 Palette** option. BusinessWorks Plug-in Development Kit deletes the reference of an activity from the palette but does not delete the source code. You have to manually delete source code related to the activity.

Testing a Plug-in

After creating a plug-in, you can test your plug-in by checking the UI, running the plug-in, checking the online help, and so on.

Open TIBCO Business Studio and perform the following actions to test your plug-in:

- Click the created activity and check if the UI elements are displayed as designed, and that the input and output schema are correct.
- If you have configured the online help settings as described in How to Add Online Help for a Palette, right-click the created activity and click **Help > Reference Page** to check if the online help for the activity is displayed.
- Create a BusinessWorks process by using the created activity and run the process. Check if the result is expected.

Creating an Installer for a Plug-in

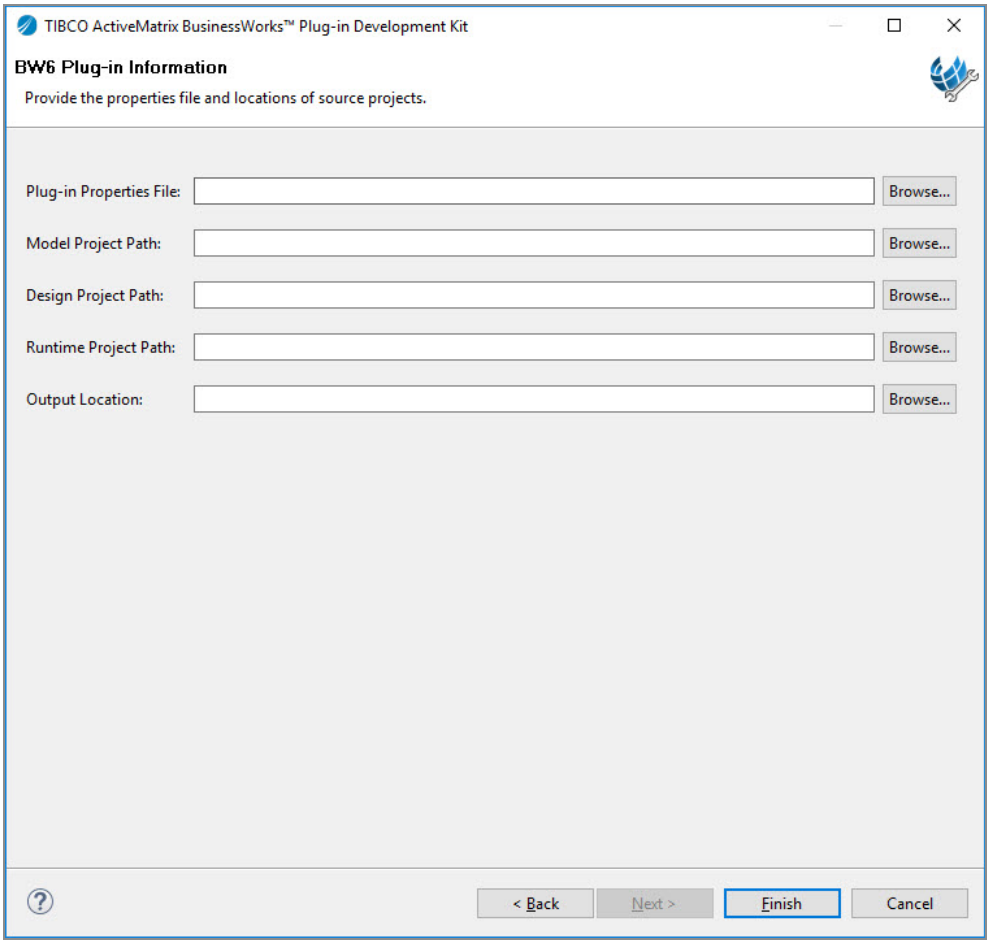
You can create an installer for the palette generated by BusinessWorks Plug-in Development Kit. This installer can be installed in TIBCO Business Studio through the Eclipse Provisioning system.

Generating an Installer

BusinessWorks Plug-in Development Kit can package a created plug-in as a P2 update site.

Procedure

1. Open the BusinessWorks Plug-in Development Kit wizard in one of the following ways:
   - From the menu, click **File > New > Create a new or modify a BusinessWorks Plug-in project**.
   - On the toolbar, click .
   - Press Alt+T.
2. In the Get Started dialog box, Click **Build an Update Site for a BW6 Palette**.
3. In the BW6 Plug-in Information dialog box, provide the information for **Plug-in Properties File**.



4. Click **Browse** to locate the `bw6_devkit_configuration.properties` file that is located in the created project folder.

   The paths for **Model Project Path**, **Design Project Path**, **Runtime Project Path**, and **Output Location** will be automatically selected. You can change the path for all of them. But it is recommended not to change the paths.

5. Click **Finish** to package the plug-in.

Result

The following folders and scripts are generated in the output folder:

- The `doc` folder contains a documentation template generated according to the created palette and activity. You can update this template and use it as the online help for the created activities. See Creating Documentation for more details.

- The `ePass` folder contains the bwce runtime `zip` file for the plug-in which you have created.

- The exportedFeatures folder contains the bundles that the newly added source build produces.

  > **Note:** These files can be used for troubleshooting purposes only.

- The p2Installer folder contains a P2 update site.

  You can use this installer to install the created palette in TIBCO Business Studio by using Eclipse Update Manager.

- The installation and uninstallation scripts for palette runtime: rinstall.sh, rinstall.bat, runinstall.sh, and runinstall.bat

For more information on installing and uninstalling the palette runtime, see Installing and Uninstalling a Created Plug-in in Studio and Installing and Uninstalling Runtime from Command Line

What to do next

Installing a Created Plug-in

Using a Plug-in

After you have tested the plug-in created by BusinessWorks Plug-in Development Kit, you can use the plug-in to complete a task by creating a business process and deploying the business process.

Before using the plug-in, see Installing a Created Plug-in to install the plug-in in TIBCO Business Studio.

Installing and Uninstalling a Created Plug-in in TIBCO Business Studio

BusinessWorks Plug-in Development Kit generates a P2 installer for a created plug-in. You can install and uninstall the created plug-in by using Eclipse Update Manager.

Installing a Created Plug-in

You can use Eclipse Update Manager to install a created BusinessWorks 6 plug-in in TIBCO Business Studio.

Before you begin

Ensure that you have generated a P2 installer as described in Creating an Installer for a Plug-in.

Procedure

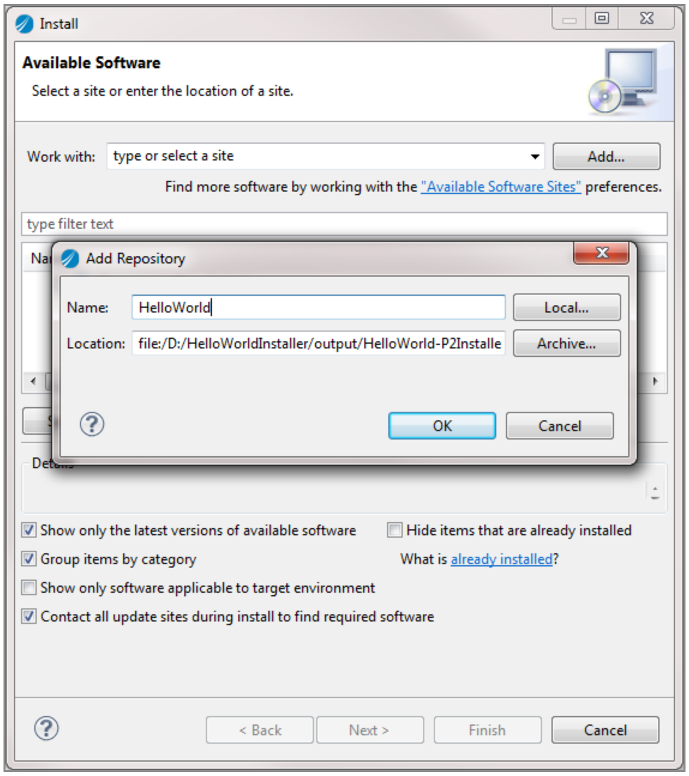
1. Open TIBCO Business Studio in one of the following ways:

    - Microsoft Windows: click **Start > All Programs > TIBCO > *TIBCO_HOME* > TIBCO Business Studio *version_number* > Studio for Designers**.
    - Linux: run the TIBCO Business Studio executable located in the *TIBCO_*

HOME/studio/*version_number*/eclipse directory.

- Mac OS: run the TIBCO Business Studio executable located in the *TIBCO_HOME*/studio/*version_number*/eclipse directory.

2. From the menu, click **Help > Install New Software**.
3. In the Available Software dialog box, click **Add**.
4. In the Add Repository dialog box, click **Local** and select the P2 installer folder that you have generated. Click **OK**.


Install
Available Software
Select a site or enter the location of a site.
Add...
Find more software by working with the "Available Software Sites" preferences.
Add Repository
Local...
Archive...
OK
Cancel
What is already installed?
< Back
Next >
Finish
Cancel

Work with: type or select a site

type filter text

Name: HelloWorld

Location: file:/D:/HelloWorldInstaller/output/HelloWorld-P2Installe

☑ Show only the latest versions of available software
□ Hide items that are already installed
☑ Group items by category
□ Show only software applicable to target environment
☑ Contact all update sites during install to find required software

5. Select the created plug-in that you want to install. Click **Next**.
6. In the Install Details dialog box, review the components to be installed. Click **Next**.
7. In the Review Licenses dialog box, click **I accept the terms of the license agreement**. Click **Finish** to install the plug-in.
8. During the installation, a security warning dialog box is displayed, click **OK** to complete the installation.
9. Click **Yes** when you are prompted to restart TIBCO Business Studio.

Uninstalling a Created Plug-in

Use Eclipse Update Manager to uninstall BusinessWorks Plug-in Development Kit or a plug-in that is created by using BusinessWorks Plug-in Development Kit.

Procedure

1. Open TIBCO Business Studio in one of the following ways:
   - Microsoft Windows: click **Start > All Programs > TIBCO > *TIBCO_HOME* > TIBCO Business Studio *version_number* > Studio for Designers**.
   - Linux: run the TIBCO Business Studio executable located in the `TIBCO_HOME/studio/version_number/eclipse` directory.
   - Mac OS: run the TIBCO Business Studio executable located in the `TIBCO_HOME/studio/version_number/eclipse` directory.
2. From the menu, click **Help > Installation Details** to open the Eclipse Update Manager.
3. In the **Installed Software** tab, click the plug-in that you want to uninstall, and then click **Uninstall**.
4. In the Uninstall Details dialog box, review the product to be uninstalled. Click **Finish** to uninstall the selected plug-in.
5. Click **Yes** when you are prompted to restart TIBCO Business Studio.

Installing and Uninstalling Runtime from Command Line

You can install and uninstall the runtime from the command line either by using installation and uninstallation scripts or by executing the Java command directly with the JAR file.

Installing and Uninstalling Runtime Using Scripts

You can install and uninstall only the runtime part of your palette using scripts provided as a part of the generated P2 installer. The target *TIBCO_HOME* does not require to have TIBCO Business Studio installed. It can be a runtime-only installation of TIBCO ActiveMatrix BusinessWorks.

Note:

- You can find the installation and uninstallation scripts at the root directory of a generated P2 installer.
- Ensure Java version is set to Java 11.

Installing Runtime Using Scripts

You can install the runtime of your palette into a specific `TIBCO_HOME` location by using the `rinstall.bat` or `rinstall.sh` scripts.

Procedure

1. Navigate to the `p2Installer` directory.
2. Run the following command to start the installation:
   - On Microsoft Windows: rinstall *TIBCO_HOME*
   - On Linux: `rinstall.sh TIBCO_HOME`

Uninstalling Runtime Using Scripts

You can uninstall the runtime of your palette into a specific `TIBCO_HOME` location by using the `runinstall.bat` or `runinstall.sh` scripts.

Procedure

1. Navigate to the `p2Installer` directory.
2. Run the following command to start the uninstallation:
   - On Microsoft Windows: runinstall *TIBCO_HOME*

- On Linux: runinstall.sh *TIBCO_HOME*

Installing and Uninstalling Runtime Using JAR File

You can install and uninstall only the runtime part of your palette by executing the Java command with the JAR file `com.tibco.tools.updatesite.devkit.actions_6.3.1.`*`nnn`*`.jar`. This file is provided as a part of a generated P2 installer. You can also navigate to `p2Installer\plugins` directory or *`TIBCO_HOME`*`\bw\palettes\devkit\`*`version`*`\design\plugins` directory to find this JAR file.

> **Note:**
> - This jar file is self-contained and does not require additional classpath specification.
> - Ensure Java version is set to Java 11.

Installing Runtime Using JAR File

Procedure

1. Run the following JAR file to install a palette runtime:

   ```
   java -jar com.tibco.tools.updatesite.devkit.actions_6.3.1.nnn.jar -ti -th
   TIBCO_HOME -p2 p2Installer directory
   ```

   Example:

   ```
   java -jar com.tibco.tools.updatesite.devkit.actions_6.3.1.nnn.jar -ti -th
   /var/my-tibco-home -p2 /johnsmith/workspace/MyPalette/p2Installer
   ```

Uninstalling Runtime Using JAR File.

Procedure

1. Run the following JAR file to uninstall a palette runtime:

   ```
   java -jar com.tibco.tools.updatesite.devkit.actions_6.3.1.nnn.jar -tu -th
   TIBCO_HOME -pname Palette Name
   ```

   Example:

```
java -jar com.tibco.tools.updatesite.devkit.actions_6.3.1.nnn.jar -tu -th
/var/my-tibco-home -pname MyPalette
```

Displaying Help Using JAR File

Procedure

1. Run the following JAR file to display help:

   ```
   java -jar com.tibco.tools.updatesite.devkit.actions_6.3.1.nnn.jar -help
   ```

Debugging the Plug-in

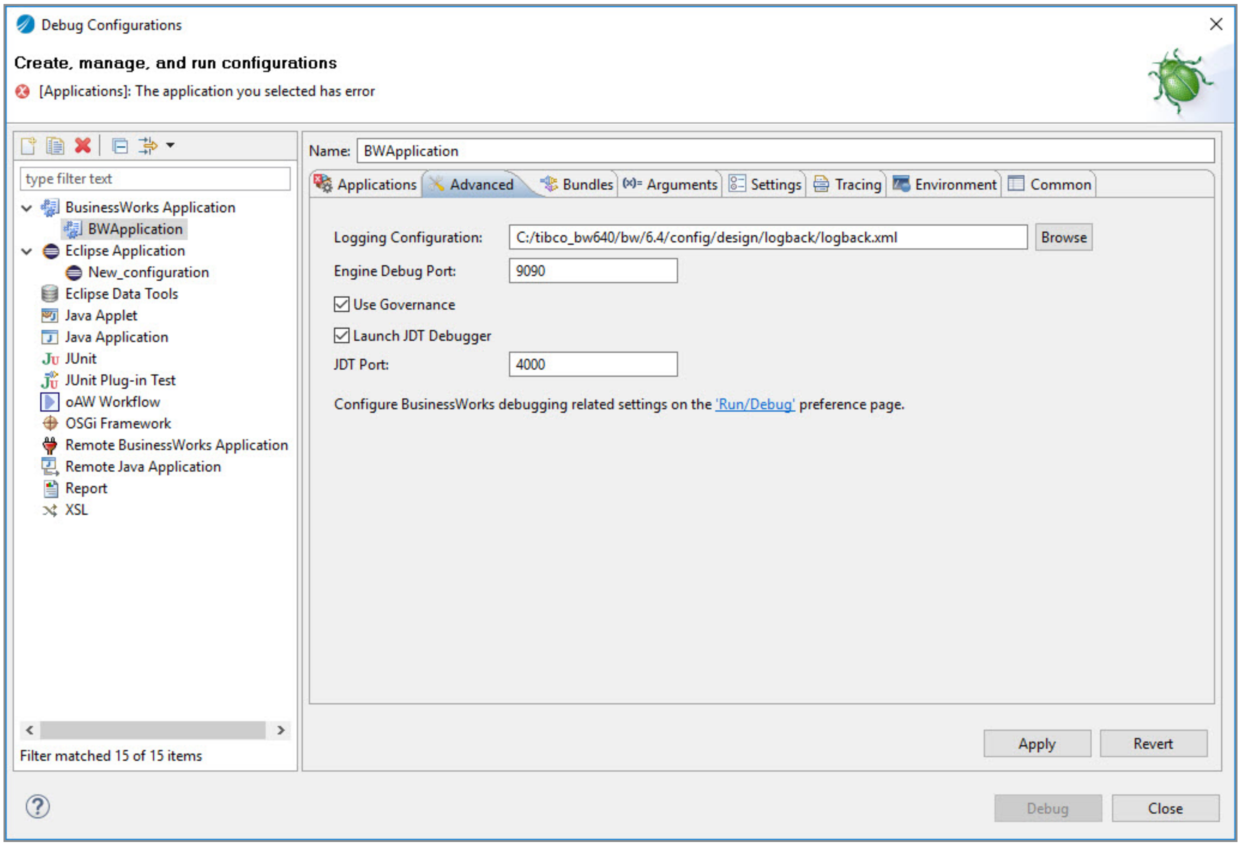
To debug the runtime of your created plug-in:

1. Generate the p2 site for your palette. See Generating an Installer
2. Install the plug-in on studio. See Installing a Created Plug-in

   > **Note:** Now, you can see your palette in the palette library and you can create a process with your activities.

3. You can set breakpoints in your runtime code.
4. Click **Run > Debug Configurations...**
5. In the Create, manage, and run configurations dialog box, select **BWAplications** under **BusinessWorks Application**.
6. In the Advanced tab, select **Launch JDT Debugger** check box.
7. Click **Apply**.

For more information about how to create a process, see *TIBCO ActiveMatrix BusinessWorks Application Deployment*.



Name: BWApplication

Logging Configuration: C:/tibco_bw640/bw/6.4/config/design/logback/logback.xml

Engine Debug Port: 9090

☑ Use Governance

☑ Launch JDT Debugger

JDT Port: 4000

Configure BusinessWorks debugging related settings on the 'Run/Debug' preference page.

Deploying an Application

You can deploy an application to TIBCO Enterprise Administrator. After deploying the application, you can manage the application in TIBCO Enterprise Administrator.

A typical workflow for deploying an application is:

1. Creating an application archive
2. Uploading an application archive
3. Deploying an application archive
4. Starting an application

For more details of how to deploy an application, see the TIBCO ActiveMatrix BusinessWorks 6 documentation.

Working with the Sample Projects

The plug-in packages sample projects with the P2 installer. The sample projects show how the plug-ins generated by BusinessWorks Plug-in Development Kit work.

After installing TIBCO ActiveMatrix BusinessWorks Plug-in Development Kit, you can locate the sample projects in the `TIBCO_HOME/bw/palettes/devkit/version_number/samples` directory. The sample projects contain processes, each process corresponds to a task.

- GSON

  You can convert Java object to JavaScript Object Notation (JSON).

- LinkedIn

  You can use the LinkedIn processes to share and post content by calling REST API.

GSON

You can convert Java object to JavaScript Object Notation (JSON).

1. Importing the GSON Sample Project
2. Importing the JavaToJSON Process
3. Running the JavaToJSON Process

Importing the GSON Sample Project

Before running the project, you must import the sample project to TIBCO Business Studio.

Procedure

1. Open TIBCO Business Studio in one of the following ways:

   - Microsoft Windows: click **Start > All Programs > TIBCO > *TIBCO_HOME* >**

   TIBCO Business Studio *version_number* > Studio for Designers.

   - Linux: run the TIBCO Business Studio executable located in the *TIBCO_HOME*/studio/*version_number*/eclipse directory.
   - Mac OS: run the TIBCO Business Studio executable located in the *TIBCO_HOME*/studio/*version_number*/eclipse directory.

2. From the menu, click **File > Import**. In the Select dialog box, expand the **General** folder and select the **Existing Projects into Workspace** item. Click **Next**.
3. Click **Browse** next to the **Select root directory** field to locate the sample. Clear the check boxes of runtime bundle and feature, click **Finish.**

   The sample project is located in the *TIBCO_HOME*/bw/palettes/devkit/*version_number*/samples/GSON directory.
4. Change the target platform to running platform for the design-time module.
   1. Click **Window > Preferences**, and then click **Plug-in Development > Target Platform**.
   2. Click **Add** to add a running platform for the design-time module.
   3. In the Target Definition dialog box, click **Default** to choose the running platform. Click **Next**.
   4. In the Target Content dialog box, click **Finish**.
   5. You are back to the Target Platform dialog box, select the **Running Platform (Active)** check box. Click **Apply**, and then click **OK**.

   By default, the target platform is bw-runtime after launching TIBCO Business Studio. See Target Platform for more details.
5. Click **Run > Run Configurations** to launch a child TIBCO Business Studio.
6. In the "Create, manage, and run configurations" dialog box, double-click **Eclipse Application** in the left panel to create a new Eclipse application.
7. Click **Run** to launch a child TIBCO Business Studio configured with the bw-runtime platform.
8. In the child TIBCO Business Studio, click **File > Import** to import the model and runtime bundles and features:
   1. In the Select dialog box, expand the **General** folder and select **Existing**

Projects into Workspace. Click **Next**.

b. In the Import Projects dialog box, click **Browse** to locate the project folder that contains the plug-in project.

c. Click **Deselect All**, and then select the runtime and model bundles and features. Click **Finish**.

Result

The sample project is imported to TIBCO Business Studio.

What to do next

Importing the JavaToJSON Process

Importing the JavaToJSON Process

Before running the JavaToJSON process, you must import this process to TIBCO Business Studio.

Before you begin

Ensure that you have imported the GSON sample project, as described in Importing the GSON Sample Project.

Procedure

1. In the child TIBCO Business Studio, click **File > Import**.
2. In the Select dialog box, expand the **General** folder and select the **Existing Studio Projects into Workspace** item. Click **Next.**
3. Click **Browse** next to the **Select archive file** field to locate the `Sample.zip` file.

   The `Sample.zip` file is located in the *`TIBCO_HOME`*`/bw/palettes/devkit/`*`version_number`*`/samples` directory.
4. In the Import Projects dialog box, click **Deselect All**, and then select the GSON sample project. Click **Finish**.

What to do next

Running the JavaToJSON Process

Running the JavaToJSON Process

Run the JavaToJSON process to convert Java object to JSON.

Before you begin

Ensure that you have imported the process, as described in Importing the JavaToJSON Process.

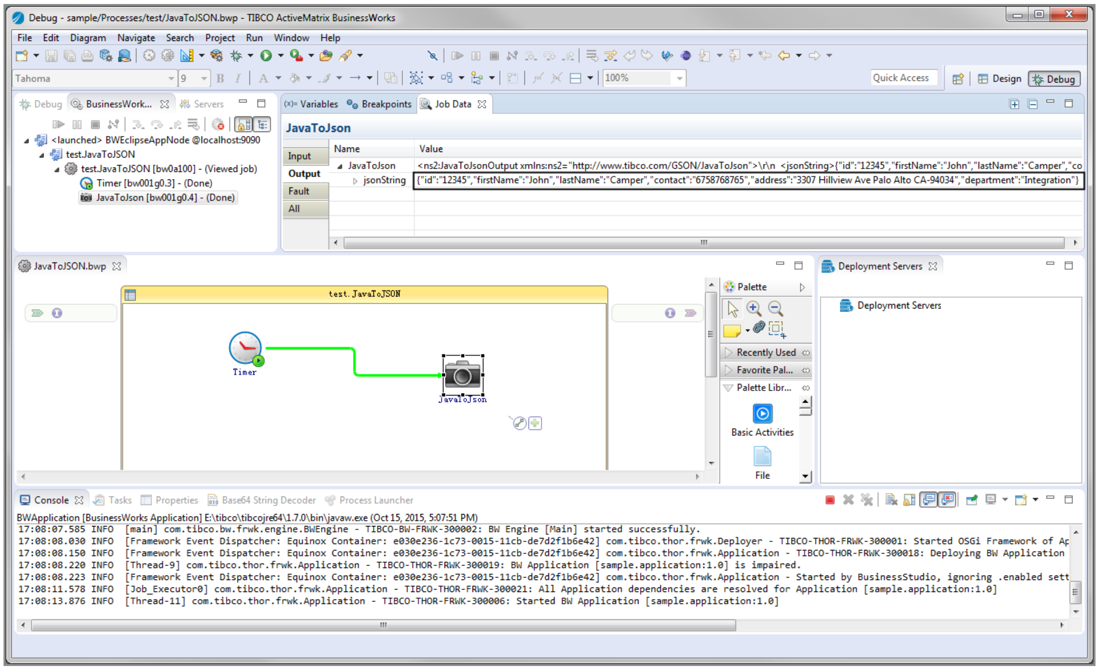
Procedure

1. In the Project Explorer view, expand **GSON-Sample > Processes > test > JavaToJSON.bwp**.
2. In the process editor, double-click the JavaToJson activity, enter the following parameters:
   a. Click the **General** tab, click 🔍 to create or choose a shared resource.
   b. Click the **Input** tab, enter a value in the required **address** field.

      You can also enter other optional values in other fields.
3. On the toolbar, click 💾 to save your configurations.
4. From the menu, click **Run > Debug Configurations**.
5. In the "Create, manage, and run configurations" dialog box, click **BusinessWorks Application > BWApplication** in the left panel, and then select **GSON-Sample.application** in the **Applications** tab in the right panel.
6. Click **Debug** to start the test process.

Result

In the Debug perspective, click the **BusinessWorks Jobs** tab, and then click **test.JavaToJSON > JavaToJSON**. Click the **Output** tab in the Job Data view. The output is displayed in the **Output** tab.



JavaToJson

	Name	Value
Input	JavaToJson	<ns2:JavaToJsonOutput xmlns:ns2="http://www.tibco.com/GSON/JavaToJson">\r\n <jsonString>{"id":"12345","firstName":"John","lastName":"Camper","co
Output	jsonString	{"id":"12345","firstName":"John","lastName":"Camper","contact":"6758768765","address":"3307 Hillview Ave Palo Alto CA-94034","department":"Integration"}
Fault		
All		

```
BWApplication [BusinessWorks Application] E:\tibco\tibcojre64\1.7.0\bin\javaw.exe (Oct 15, 2015, 5:07:51 PM)
17:08:07.585 INFO  [main] com.tibco.bw.frwk.engine.BWEngine - TIBCO-BW-FRWK-300002: BW Engine [Main] started successfully.
17:08:08.030 INFO  [Framework Event Dispatcher: Equinox Container: e030e236-1c73-0015-11cb-de7d2f1b6e42] com.tibco.thor.frwk.Deployer - TIBCO-THOR-FRWK-300001: Started OSGi Framework of Ap
17:08:08.150 INFO  [Framework Event Dispatcher: Equinox Container: e030e236-1c73-0015-11cb-de7d2f1b6e42] com.tibco.thor.frwk.Application - TIBCO-THOR-FRWK-300018: Deploying BW Application
17:08:08.220 INFO  [Thread-9] com.tibco.thor.frwk.Application - TIBCO-THOR-FRWK-300019: BW Application [sample.application:1.0] is impaired.
17:08:08.223 INFO  [Framework Event Dispatcher: Equinox Container: e030e236-1c73-0015-11cb-de7d2f1b6e42] com.tibco.thor.frwk.Application - Started by BusinessStudio, ignoring .enabled sett
17:08:11.578 INFO  [Job_Executor0] com.tibco.thor.frwk.Application - TIBCO-THOR-FRWK-300021: All Application dependencies are resolved for Application [sample.application:1.0]
17:08:13.876 INFO  [Thread-11] com.tibco.thor.frwk.Application - TIBCO-THOR-FRWK-300006: Started BW Application [sample.application:1.0]
```

LinkedIn

You can use the LinkedIn processes to share and post content by calling REST API.

1. Importing the LinkedIn Sample Project
2. Importing the LinkedIn Processes
3. Generating an Access Token and a Token Secret
4. Running the LinkedIn Processes

Importing the LinkedIn Sample Project

Before running the project, you must import the sample project to TIBCO Business Studio.

Procedure

1. Open TIBCO Business Studio in one of the following ways:
   - Microsoft Windows: click **Start** > **All Programs** > **TIBCO** > ***TIBCO_HOME*** **>**

TIBCO Business Studio *version_number* > Studio for Designers.

- Linux: run the TIBCO Business Studio executable located in the `TIBCO_HOME/studio/version_number/eclipse` directory.
- Mac OS: run the TIBCO Business Studio executable located in the `TIBCO_HOME/studio/version_number/eclipse` directory.

2. From the menu, click **File > Import**. In the Select dialog box, expand the **General** folder and select the **Existing Projects into Workspace** item. Click **Next.**
3. Click **Browse** next to the **Select root directory** field to locate the sample. Clear the check boxes of runtime bundle and feature, click **Finish.**

   The sample project is located in the `TIBCO_HOME/bw/palettes/devkit/version_number/samples/LinkedIn` directory.
4. Change the target platform to running platform for the design-time module.
   a. Click **Window > Preferences**, and then click **Plug-in Development > Target Platform**.
   b. Click **Add** to add a running platform for the design-time module.
   c. In the Target Definition dialog box, click **Default** to choose the running platform. Click **Next**.
   d. In the Target Content dialog box, click **Finish**.
   e. You are back to the Target Platform dialog box, select the **Running Platform (Active)** check box. Click **Apply**, and then click **OK**.

   By default, the target platform is bw-runtime after launching TIBCO Business Studio. See Target Platform for more details.
5. Click **Run > Run Configurations** to launch a child TIBCO Business Studio.
6. In the "Create, manage, and run configurations" dialog box, double-click **Eclipse Application** in the left panel to create a new Eclipse application.
7. Click **Run** to launch a child TIBCO Business Studio configured with the bw-runtime platform.
8. In the child TIBCO Business Studio, click **File > Import** to import the model and runtime bundles and features:
   a. In the Select dialog box, expand the **General** folder and select **Existing**

Projects into Workspace. Click **Next**.

b. In the Import Projects dialog box, click **Browse** to locate the project folder that contains the plug-in project.

c. Click **Deselect All**, and then select the runtime and model bundles and features. Click **Finish**.

Result

The sample project is imported to TIBCO Business Studio.

What to do next

Importing the LinkedIn Processes

Importing the LinkedIn Processes

Before running the LinkedIn processes, you must import these processes to TIBCO Business Studio.

Before you begin

Ensure that you have imported the LinkedIn sample project, as described in Importing the LinkedIn Sample Project.

Procedure

1. In the child TIBCO Business Studio, click **File > Import**.
2. In the Select dialog box, expand the **General** folder and select the **Existing Studio Projects into Workspace** item. Click **Next.**
3. Click **Browse** next to the **Select archive file** field to locate the `Sample.zip` file.

   The `Sample.zip` file is located in the *`TIBCO_HOME`*`/bw/palettes/devkit/`*`version_number`*`/samples` directory.
4. In the Import Projects dialog box, click **Deselect All**, and then select the LinkedIn sample project. Click **Finish**.

What to do next

Generating an Access Token and a Token Secret

Generating an Access Token and a Token Secret

You have to generate an access token and a token secret before running LinkedIn processes.

Before you begin

Ensure that you have installed Java version 1.7 or above on your machine.

Procedure

1. Log on to the LinkedIn website using your user name and password.

   You have to register first if you do not have a user name and password.

2. Go to the Developers page, and click **My Apps**. Click **Create Application** to create your applications.

3. Enter all the required information, and click **Submit** to receive the authentication keys of client ID and client secret.

4. In the **Default Application Permissions** area, select the **w_share** check box to share content by the application on LinkedIn. Click **Update**.

5. Extract the `LinkedinTokenGenerator.zip` file in the *TIBCO_HOME*`/bw/palettes/devkit/`*version_number*`/samples` directory to a temporary directory.

6. On the command line, navigate to the temporary directory where this tool is extracted, run the `javac -cp "lib/*" src/TokenGenerator.java` command, and then run the following command:

   - Microsoft Windows: `java -cp "src/;lib/*" TokenGenerator`
   - Linux: `java -cp "src/:lib/*" TokenGenerator`
   - Mac OS: `java -cp "src/:lib/*" TokenGenerator`

7. On the command line, enter the client ID and the client secret that you received in Step 3.

8. Go to the website link that is displayed on the command line, and click **Allow access** to allow access to your LinkedIn information.

   After authorizing the access, you receive a PIN code to grant access.

9. On the command line, enter this PIN code to generate the access token and the token secret.

Running the LinkedIn Processes

Run the LinkedIn processes to share and post content by the REST API.

Before you begin

Ensure that you have imported the processes, as described in Importing the LinkedIn Processes. You have to generate an access token and a token secret before running LinkedIn processes, see Generating an Access Token and a Token Secret for more details.

Procedure

1. You can run the following processes:
   - Running the Retrieve Process
   - Running the RetrieveDefaultProfile Process
   - Running the Update Process

Running the Retrieve Process

Run the Retrieve process to get the content, such as ID, the URL to the profile picture, the number of LinkedIn connections.

Before you begin

Ensure that you have imported the process, as described in Importing the LinkedIn Processes.

Procedure

1. In the Project Explorer view, expand **LinkedIn-Sample > Processes > test > Retrieve.bwp**.
2. In the process editor, double-click the Retrieve activity, enter the following parameters:
   a. Click the **General** tab, enter values in the **Client ID**, **Client Secret**, **Access**

Token, and **Token Secret** fields.

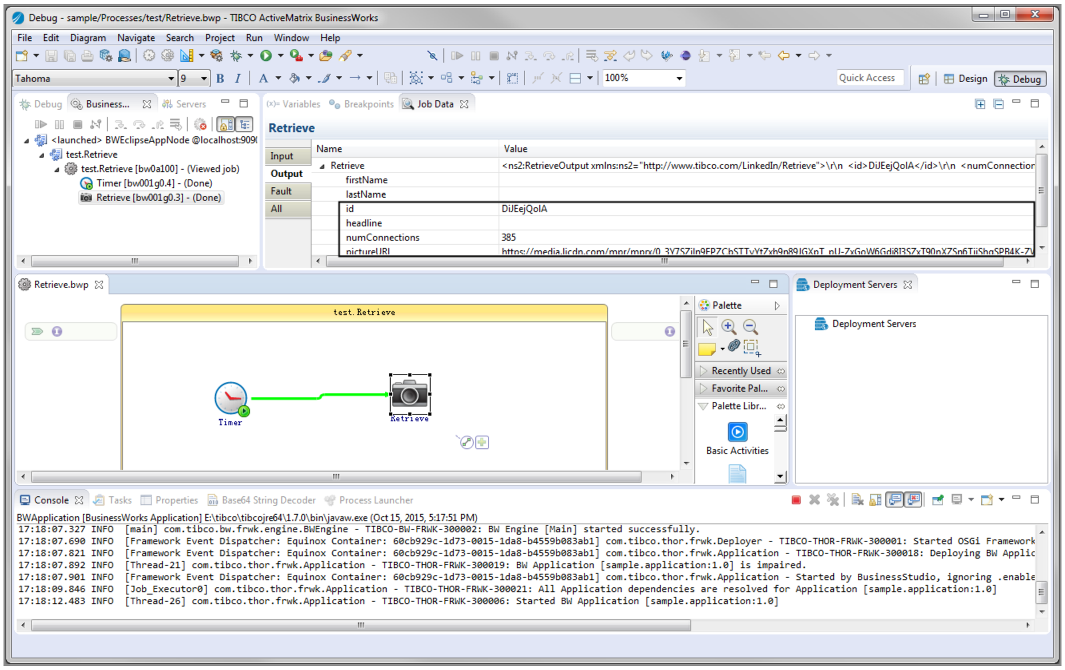
b. Click the **Input** tab, enter values in the **fields** field.

The following fields are supported: id, first-name, last-name, headline, num-connections, and picture-url.

3. On the toolbar, click 💾 to save your configurations.
4. From the menu, click **Run > Debug Configurations**.
5. In the "Create, manage, and run configurations" dialog box, click **BusinessWorks Application > BWApplication** in the left panel, and then select **LinkedIn-Sample.application** in the **Applications** tab in the right panel.
6. Click **Debug** to start the test process.

Result

In the Debug perspective, click the **BusinessWorks Jobs** tab, and then click **Test.Retrieve > Retrieve**. Click the **Output** tab in the Job Data view. The output is displayed in the **Output** tab.



<launched> BWEclipseAppNode @localhost:909(
- test.Retrieve
  - test.Retrieve [bw0a100] - (Viewed job)
    - Timer [bw001g0.4] - (Done)
    - Retrieve [bw001g0.3] - (Done)

Retrieve

Input | **Output** | Fault | All

Name	Value
Retrieve	<ns2:RetrieveOutput xmlns:ns2="http://www.tibco.com/LinkedIn/Retrieve">\r\n <id>DiJEejQolA</id>\r\n <numConnectior
firstName	
lastName	
id	DiJEejQolA
headline	
numConnections	385
pictureURL	https://media.licdn.com/mpr/mprx/0_3Y7SZiln9FPZCbSTTvYtZxh9n89JGXnT_nU-ZxGoW6Gdi8I3SZxT90nXZSn6TiiShqSPB4K-7V

BWApplication [BusinessWorks Application] E:\tibco\tibcojre64\1.7.0\bin\javaw.exe (Oct 15, 2015, 5:17:51 PM)

```
17:18:07.327 INFO  [main] com.tibco.bw.frwk.engine.BWEngine - TIBCO-BW-FRWK-300002: BW Engine [Main] started successfully.
17:18:07.690 INFO  [Framework Event Dispatcher: Equinox Container: 60cb929c-1d73-0015-1da8-b4559b083ab1] com.tibco.thor.frwk.Deployer - TIBCO-THOR-FRWK-300001: Started OSGi Framework
17:18:07.821 INFO  [Framework Event Dispatcher: Equinox Container: 60cb929c-1d73-0015-1da8-b4559b083ab1] com.tibco.thor.frwk.Application - TIBCO-THOR-FRWK-300018: Deploying BW Applic
17:18:07.892 INFO  [Thread-21] com.tibco.thor.frwk.Application - TIBCO-THOR-FRWK-300019: BW Application [sample.application:1.0] is impaired.
17:18:07.901 INFO  [Framework Event Dispatcher: Equinox Container: 60cb929c-1d73-0015-1da8-b4559b083ab1] com.tibco.thor.frwk.Application - Started by BusinessStudio, ignoring .enable
17:18:09.846 INFO  [Job_Executor0] com.tibco.thor.frwk.Application - TIBCO-THOR-FRWK-300021: All Application dependencies are resolved for Application [sample.application:1.0]
17:18:12.483 INFO  [Thread-26] com.tibco.thor.frwk.Application - TIBCO-THOR-FRWK-300006: Started BW Application [sample.application:1.0]
```

Running the RetrieveDefaultProfile Process

Run the RetrieveDefaultProfile process to get the default content.

Before you begin

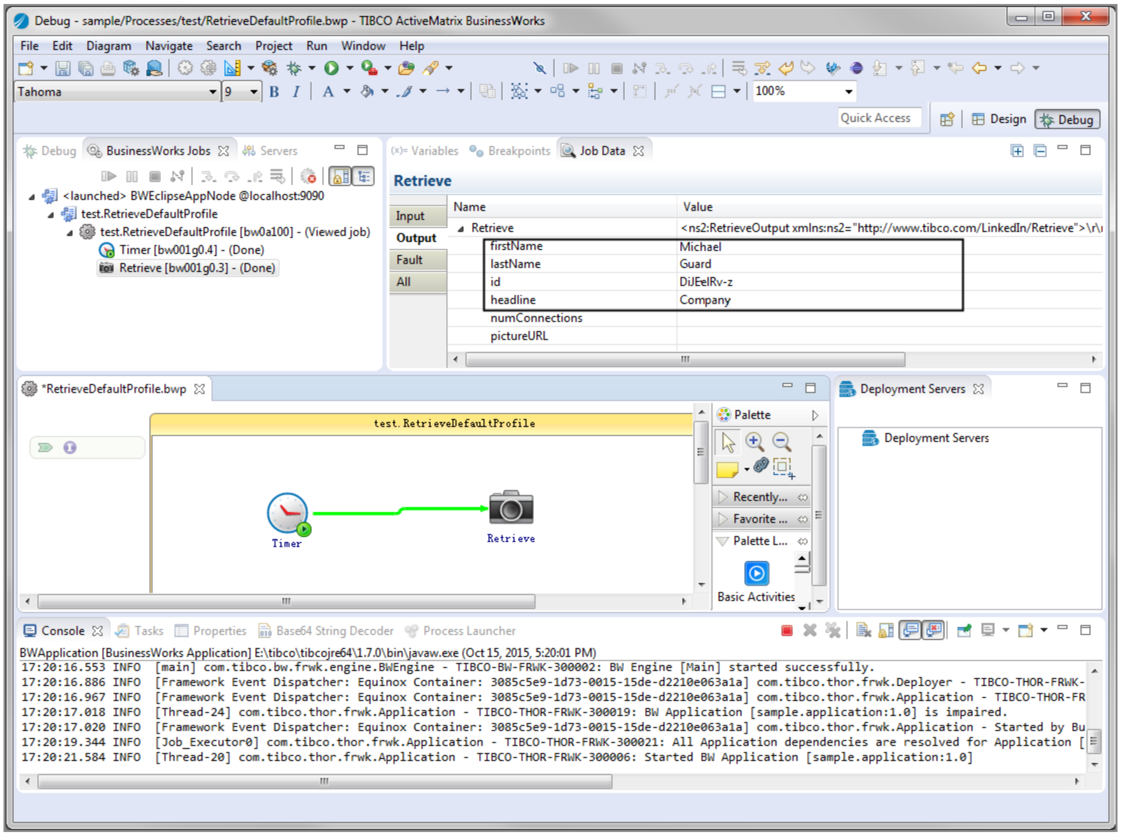
Ensure that you have imported the process, as described in Importing the LinkedIn Processes.

Procedure

1. In the Project Explorer view, expand **LinkedIn-Sample > Processes > test > RetrieveDefaultProfile.bwp**.
2. In the process editor, double-click the Retrieve activity, click the **General** tab, enter values in the **Client ID**, **Client Secret**, **Access Token**, and **Token Secret** fields.
3. On the toolbar, click 💾 to save your configurations.
4. From the menu, click **Run > Debug Configurations**.
5. In the "Create, manage, and run configurations" dialog box, click **BusinessWorks Application > BWApplication** in the left panel, and then select **LinkedIn-Sample.application** in the **Applications** tab in the right panel.
6. Click **Debug** to start the test process.

Result

In the Debug perspective, click the **BusinessWorks Jobs** tab, and then click **Test.RetrieveDefaultProfile > Retrieve**. Click the **Output** tab in the Job Data view. The output is displayed in the **Output** tab.



<launched> BWEclipseAppNode @localhost:9090
test.RetrieveDefaultProfile
test.RetrieveDefaultProfile [bw0a100] - (Viewed job)
Timer [bw001g0.4] - (Done)
Retrieve [bw001g0.3] - (Done)

Retrieve

Input | Output | Fault | All

Name	Value
Retrieve	<ns2:RetrieveOutput xmlns:ns2="http://www.tibco.com/LinkedIn/Retrieve">\r\
firstName	Michael
lastName	Guard
id	DiJEelRv-z
headline	Company
numConnections	
pictureURL	

```
BWApplication [BusinessWorks Application] E:\tibco\tibcojre64\1.7.0\bin\javaw.exe (Oct 15, 2015, 5:20:01 PM)
17:20:16.553 INFO  [main] com.tibco.bw.frwk.engine.BWEngine - TIBCO-BW-FRWK-300002: BW Engine [Main] started successfully.
17:20:16.886 INFO  [Framework Event Dispatcher: Equinox Container: 3085c5e9-1d73-0015-15de-d2210e063a1a] com.tibco.thor.frwk.Deployer - TIBCO-THOR-FRWK-
17:20:16.967 INFO  [Framework Event Dispatcher: Equinox Container: 3085c5e9-1d73-0015-15de-d2210e063a1a] com.tibco.thor.frwk.Application - TIBCO-THOR-FR
17:20:17.018 INFO  [Thread-24] com.tibco.thor.frwk.Application - TIBCO-THOR-FRWK-300019: BW Application [sample.application:1.0] is impaired.
17:20:17.020 INFO  [Framework Event Dispatcher: Equinox Container: 3085c5e9-1d73-0015-15de-d2210e063a1a] com.tibco.thor.frwk.Application - Started by Bu
17:20:19.344 INFO  [Job_Executor0] com.tibco.thor.frwk.Application - TIBCO-THOR-FRWK-300021: All Application dependencies are resolved for Application [
17:20:21.584 INFO  [Thread-20] com.tibco.thor.frwk.Application - TIBCO-THOR-FRWK-300006: Started BW Application [sample.application:1.0]
```

Running the Update Process

Run the Update process to post a message on the LinkedIn.

Before you begin

Ensure that you have imported the process, as described in Importing the LinkedIn Processes.

Procedure

1. In the Project Explorer view, expand **LinkedIn-Sample > Processes > test > Update.bwp**.

2. In the process editor, double-click the Update activity, enter the following parameters:

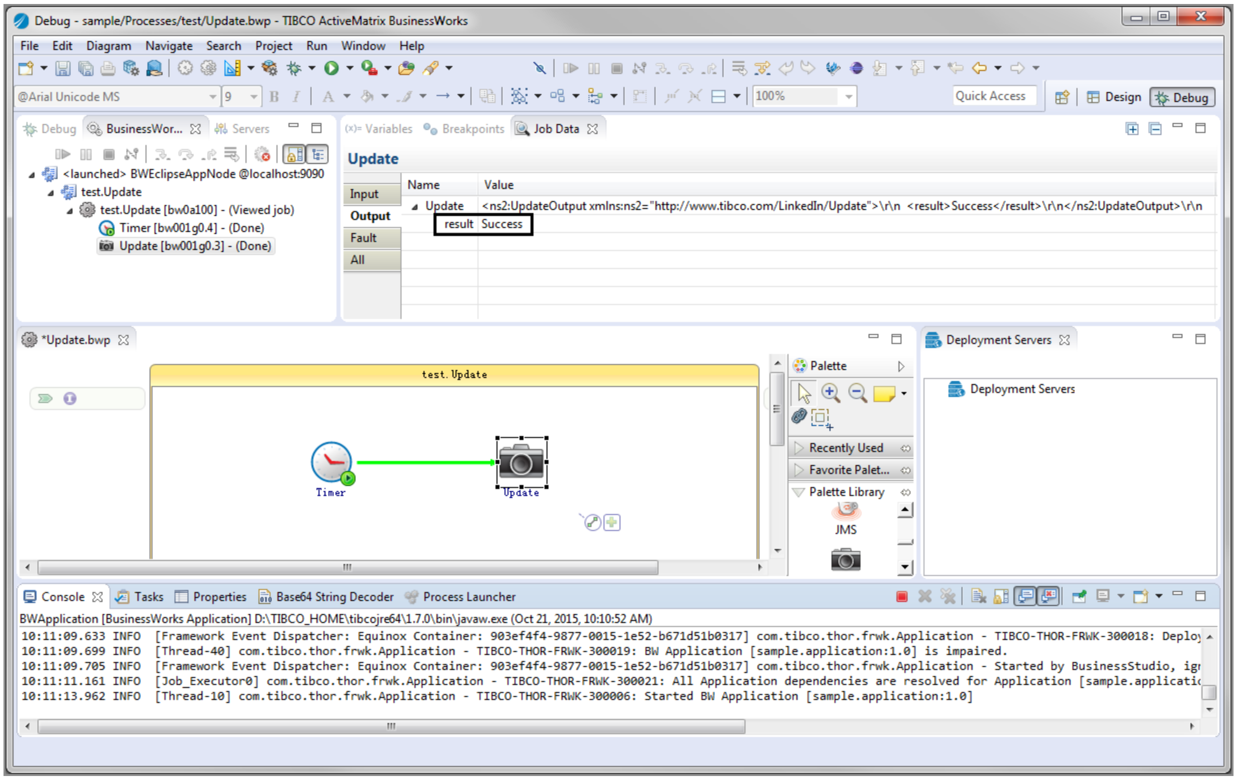
    a. Click the **General** tab, enter values in the **Client ID**, **Client Secret**, **Access Token**, and **Token Secret** fields.

   b. Click the **Input** tab, enter values in the required **text**, **title**, and **URL** fields.

3. On the toolbar, click 💾 to save your configurations.
4. From the menu, click **Run > Debug Configurations**.
5. In the "Create, manage, and run configurations" dialog box, click **BusinessWorks Application > BWApplication** in the left panel, and then select **LinkedIn-Sample.application** in the **Applications** tab in the right panel.
6. Click **Debug** to start the test process.

Result

In the Debug perspective, click the **BusinessWorks Jobs** tab, and then click **Test.Update > Update**. Click the **Output** tab in the Job Data view. The output is displayed in the **Output** tab.



Update

Name	Value
Update	<ns2:UpdateOutput xmlns:ns2="http://www.tibco.com/LinkedIn/Update">\r\n <result>Success</result>\r\n</ns2:UpdateOutput>\r\n
result	Success

BWApplication [BusinessWorks Application] D:\TIBCO_HOME\tibcojre64\1.7.0\bin\javaw.exe (Oct 21, 2015, 10:10:52 AM)

```
10:11:09.633 INFO  [Framework Event Dispatcher: Equinox Container: 903ef4f4-9877-0015-1e52-b671d51b0317] com.tibco.thor.frwk.Application - TIBCO-THOR-FRWK-300018: Deploy
10:11:09.699 INFO  [Thread-40] com.tibco.thor.frwk.Application - TIBCO-THOR-FRWK-300019: BW Application [sample.application:1.0] is impaired.
10:11:09.705 INFO  [Framework Event Dispatcher: Equinox Container: 903ef4f4-9877-0015-1e52-b671d51b0317] com.tibco.thor.frwk.Application - Started by BusinessStudio, igr
10:11:11.161 INFO  [Job_Executor0] com.tibco.thor.frwk.Application - TIBCO-THOR-FRWK-300021: All Application dependencies are resolved for Application [sample.applicati
10:11:13.962 INFO  [Thread-10] com.tibco.thor.frwk.Application - TIBCO-THOR-FRWK-300006: Started BW Application [sample.application:1.0]
```

Frequently Asked Questions

The questions, such as, how to get the runtime input, how to update the runtime output, and how to add third-party libraries, are listed with the corresponding answers.

How to Get Input at Run Time

BusinessWorks framework passes the input to the `processContext` parameter in the `public N execute(N input, ProcessContext<N> processContext)` method, which can be used to retrieve the actual value of a field.

You can find this method in the *`ActivityName_ActivityType`*`Activity.java` file from the runtime bundle.

In addition to this method, BusinessWorks Plug-in Development Kit provides the following code snippet to get the runtime input:

```
public String getInputParameterStringValueByName(final N inputData,
final ProcessingContext<N> processingContext, final String
parameterName) {
     Model<N> model = processingContext.getMutableContext().getModel();
     N parameter = model.getFirstChildElementByName(inputData, null,
parameterName);
     if (parameter == null) {
          return "";
     }
     return model.getStringValue(parameter);
}
```

where

- `input` is the activity input data.
- `processingContext` is the XML processing context.
- `parameter value` is the parameter name that you want to get the value for.

The following example shows how to get the input:

```
final String ACTIVITY_INPUT_FILE_NAME = "FileName";
String fileName = getInputParameterStringValueByName(input,
processContext.getXMLProcessingContext(), ACTIVITY_INPUT_FILE_NAME);
```

How to Create and Update Output at Run Time

BusinessWorks Plug-in Development Kit provides a `protected <A> N evalOutput(N inputData, ProcessingContext<N> processingcontext, Object data)` method to create and update runtime output.

You can find this method in the `ActivityName_ActivityTypeActivity.java` file from the runtime bundle. If you have selected using XSD to create the activity output, then the XSD elements are created inside this method.

```
protected <A> N evalOutput(N inputData, ProcessingContext<N>
processingContext, Object data) throws Exception {

       SayHelloOutput sayHelloOutput = new SayHelloOutput();
       sayHelloOutput.setOutput("Hello World");
       N output = PaletteUtil.parseObjtoN(SayHelloOutput.class,
sayHelloOutput, processingContext, activityContext.getActivityOutputType
().getTargetNamespace(), "SayHelloOutput");
       // begin-custom-code
       // add your own business code here
       // end-custom-code
       return output;
}
```

How to Add Third-Party Libraries

The added third-party libraries are accessible in bundles.

Create a Bundle Project for Third-Party JAR Libraries

You can create a project that contains the third-party JAR libraries, and then import the project to a bundle:

1. In TIBCO Business Studio, click **File > New > Project**.
2. In the "Select a wizard" dialog box, click the **Plug-in Development** folder, and then click **Plug-in from Existing JAR Archives**. Click **Next**.
3. In the "JAR selection" dialog box, click **Add** to add the third-party JAR files.
4. In the Project Explorer view, click the `MANIFEST.MF` file of the bundle where you want to add the third-party JAR files.
5. Click the **Dependencies** tab, and click **Add** in the **Imported Packages** section to add the packages required from the third-party JAR.

Add Third-Party Libraries from a Library Folder

You can add the third-party libraries from a library folder:

1. Create a **lib** folder in the runtime bundle.
2. Copy the third-party library that you want to add to the **lib** folder.
3. Open the `MANIFEST.MF` file in the runtime bundle.
4. Click the **Runtime** tab.
5. In the **Classpath** area, click **Add** and select the third-party library from the **lib** folder.

> **Important:** Manual changes to `MANIFEST.MF` will be lost if a plug-in code is re-generated.

How to Add Online Help for a Palette

You can enable the online help function for a created palette.

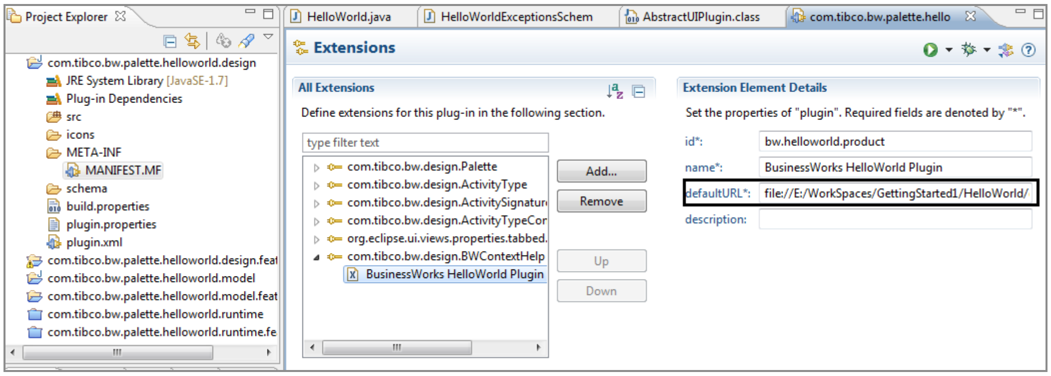
BusinessWorks Plug-in Development Kit generates a documentation template in the doc folder that is located in the palette project folder. You can tailor the user's guide to add information specific to your plug-in. The user's guide later can be made available online for a palette by using one of the BusinessWorks extensions.

See Creating Documentation for more details about how to update the document template.

Procedure

1. Open the `MANIFEST.MF` file in the design-time bundle.
2. Click the **Extensions** tab and click the **com.*company_name*.bw.design.BWContextHelp** extension.
3. Update the default URL for your plug-in. Ensure that you have placed the document that is used as the online help to an available online site.

> **Note:** Do not change **id** field.


Project Explorer
com.tibco.bw.palette.helloworld.design
JRE System Library [JavaSE-1.7]
Plug-in Dependencies
src
icons
META-INF
MANIFEST.MF
schema
build.properties
plugin.properties
plugin.xml
com.tibco.bw.palette.helloworld.design.feat
com.tibco.bw.palette.helloworld.model
com.tibco.bw.palette.helloworld.model.feat
com.tibco.bw.palette.helloworld.runtime
com.tibco.bw.palette.helloworld.runtime.fe
HelloWorld.java
HelloWorldExceptionsSchem
AbstractUIPlugin.class
com.tibco.bw.palette.hello
Extensions
All Extensions
type filter text
com.tibco.bw.design.Palette
com.tibco.bw.design.ActivityType
com.tibco.bw.design.ActivitySignatur
com.tibco.bw.design.ActivityTypeCor
org.eclipse.ui.views.properties.tabbed.
com.tibco.bw.design.BWContextHelp
BusinessWorks HelloWorld Plugin
Add...
Remove
Up
Down
Extension Element Details

Define extensions for this plug-in in the following section.

Set the properties of "plugin". Required fields are denoted by "*".

id*: bw.helloworld.product

name*: BusinessWorks HelloWorld Plugin

defaultURL*: file://E:/WorkSpaces/GettingStarted1/HelloWorld/

description:

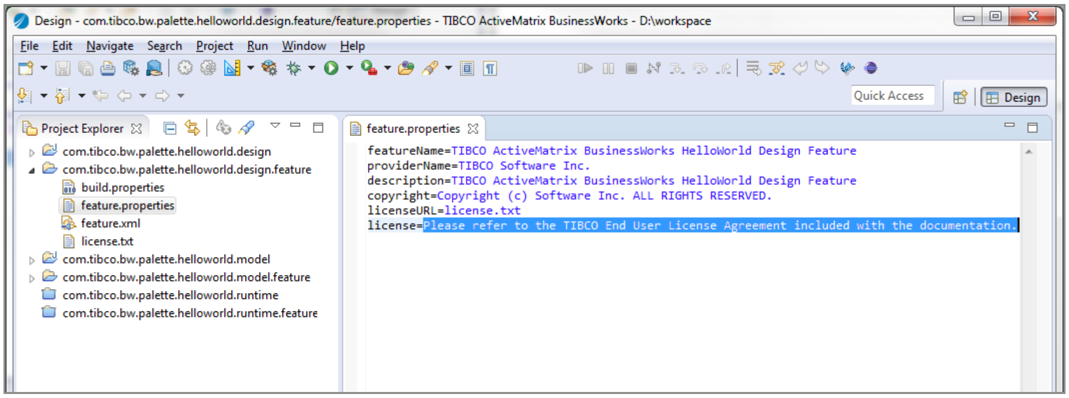
> **Important:** Manual changes to `MANIFEST.MF` will be lost if a plug-in code is re-generated.

How to Add License

You can add your license by changing the `feature.properties` file.

Procedure

1. In the Project Explorer view, open the design-time feature.
2. Open the `feature.properties` file and enter your license content as the value of the `license` parameter.


Design - com.tibco.bw.palette.helloworld.design.feature/feature.properties - TIBCO ActiveMatrix BusinessWorks - D:\workspace
File Edit Navigate Search Project Run Window Help
Quick Access
Design
Project Explorer
com.tibco.bw.palette.helloworld.design
com.tibco.bw.palette.helloworld.design.feature
build.properties
feature.properties
feature.xml
license.txt
com.tibco.bw.palette.helloworld.model
com.tibco.bw.palette.helloworld.model.feature
com.tibco.bw.palette.helloworld.runtime
com.tibco.bw.palette.helloworld.runtime.feature
feature.properties

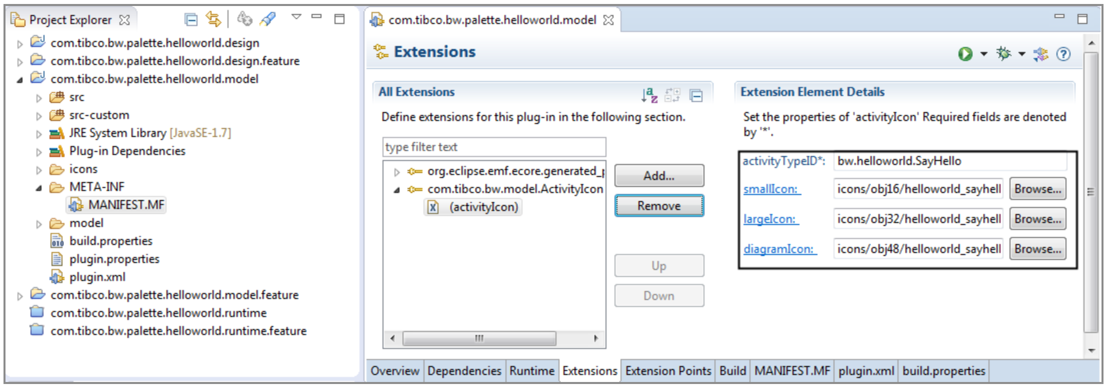
```
featureName=TIBCO ActiveMatrix BusinessWorks HelloWorld Design Feature
providerName=TIBCO Software Inc.
description=TIBCO ActiveMatrix BusinessWorks HelloWorld Design Feature
copyright=Copyright (c) Software Inc. ALL RIGHTS RESERVED.
licenseURL=license.txt
license=Please refer to the TIBCO End User License Agreement included with the documentation.
```

How to Add an Activity Icon

You can add an icon for an activity.

Procedure

1. Open the `MANIFEST.MF` file in the model bundle.
2. Click the **Extensions** tab, and then click **com.*company_name*.bw.model.ActivityIcon > activityIcon**.
3. Update the ID of the activity type, and click **Browse** to change the icons of small, large, and diagram icons.


Project Explorer
com.tibco.bw.palette.helloworld.design
com.tibco.bw.palette.helloworld.design.feature
com.tibco.bw.palette.helloworld.model
src
src-custom
JRE System Library [JavaSE-1.7]
Plug-in Dependencies
icons
META-INF
MANIFEST.MF
model
build.properties
plugin.properties
plugin.xml
com.tibco.bw.palette.helloworld.model.feature
com.tibco.bw.palette.helloworld.runtime
com.tibco.bw.palette.helloworld.runtime.feature
com.tibco.bw.palette.helloworld.model
Extensions
All Extensions
Define extensions for this plug-in in the following section.
type filter text
org.eclipse.emf.ecore.generated_
com.tibco.bw.model.ActivityIcon
(activityIcon)
Add...
Remove
Up
Down
Extension Element Details
Set the properties of 'activityIcon' Required fields are denoted by '*'.
Overview
Dependencies
Runtime
Extensions
Extension Points
Build
MANIFEST.MF
plugin.xml
build.properties

activityTypeID*: bw.helloworld.SayHello

smallIcon: icons/obj16/helloworld_sayhell Browse...

largeIcon: icons/obj32/helloworld_sayhell Browse...

diagramIcon: icons/obj48/helloworld_sayhell Browse...

How to Find BusinessWorks API JavaDoc

You can find BusinessWorks API and GenXSD JavaDoc to know details about classes and methods.

Procedure

1. After installing ActiveMatrix BusinessWorks, you can find BusinessWorks API and GenXSD JavaDoc in the `*TIBCO_HOME*/bw/*version_number*/doc/html/javadoc` directory.

Troubleshooting

You might encounter compiling errors when creating a plug-in, or you might encounter issues when you migrate BusinessWork Plug-in Development Kit earlier versions to 6.3.1.

For troubleshooting, see General Problems

General Problems

You might encounter compiling errors when creating a plug-in, you can go over the listed scenarios for troubleshooting.

Scenarios	Reason/Workaround
A refresh error occasionally occurs when generating the plug-in bundles because the project is not open yet.	Reopen the project and refresh the project, and then close the project.
A configuration error occurs after loading the property file of the plug-in to be edited.	Check the ecore file according to error information.
After generating the palette code, an error dialog box is displayed when importing projects.	Close the error dialog box.
Validation errors occur when running activities that contain the `any` element.	Add code manually.
After installing a generated plug-in, "TIBCO ActiveMatrix" is added at the beginning of the feature	Edit the `featureName` property to change the feature name, the `providerName` property to change the provider name, and the `description` property to change the

Scenarios	Reason/Workaround

name.	details in the `feature.properties` files in the following folders:
	1. Design feature:
	`com.`*`companyname`*`.bw.palette.`*`palette_name`*`.design.feature`
	2. Model feature:
	`com.`*`companyname`*`.bw.palette.`*`palette_name`*`.model.feature`
	3. Runtime feature:
	`com.`*`companyname`*`.bw.palette.`*`palette_name`*`.runtime.feature`

Child Studio fails to start with an error. You see this message: java.lang.IllegalStateException: Unable to acquire application service.	Design is not the current target platform. Switch to the Design platform using the main window. Navigate to **Target Platform > Design**.
Starting of BWApplication fails. You see this message: Missing Constraint: Require-Capability...	BWApplication does not contain palette bundles. Navigate to **Run > Debug Configurations**. In the Create, manage, and run configurations page, expand **BusinessWorks Application** and click **BWApplication**. Select all the palette bundles in the **Bundles** tab.
Child Studio fails to start with an error "java.lang.IllegalStateException: Unable to acquire application service."	"Design" is not the current target platform. Switch to the Design target platform from main window; **Target Platform > Design**.

Migration Problems

You might encounter errors when you migrate ActiveMatrix BusinessWorks Plug-in Development Kit 6.2.1 or older version to 6.3.1, you can go over the listed scenarios for troubleshooting.

Scenarios	Reason/Workaround
The Runtime plug-in displays the following compilation errors: JAXBContext cannot be resolved JAXBElement cannot be resolved JAXBException cannot be resolved import javax.xml.bind cannot be resolved	The `javax.xml.bind` package has been removed from Java 11, so add the following import package statement in your Runtime plug-in's `MANIFEST.MF`. `javax.xml.bind;version="[2.0.0,3.0.0)", javax.xml.bind.annotation;version="[2.0.0,3.0.0)",`
The plug-in project displays the following compilation errors: Syntax error, annotations are only available if source level is 1.5 or greater '<>' operator is not allowed for source level below 1.7	1. Right click the project and select **Properties**. 2. In the left pane of the Properties window, click **Java Compiler**. 3. Select the **Enable Project Specific Settings** check box. 4. From the **Compiler compliance level:** drop down list, select **1.8**. 5. Click **Apply and Close**.

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 TIBCO Product Documentation website, mainly in HTML and PDF formats.

The TIBCO 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™ Plug-in Development Kit Product Documentation page:

- *TIBCO ActiveMatrix BusinessWorks™ Plug-in Development Kit Release Notes*
- *TIBCO ActiveMatrix BusinessWorks™ Plug-in Development Kit Installation*
- *TIBCO ActiveMatrix BusinessWorks™ Plug-in Development Kit Developer's Guide*

To directly access documentation for this product, double-click the following file:

`TIBCO_HOME/release_notes/TIB_bwpdk_6.3.1_docinfo.html`

where *TIBCO_HOME* is the top-level directory in which TIBCO products are installed. On Windows, the default *TIBCO_HOME* is `C:\tibco`. On UNIX systems, the default *TIBCO_HOME* is `/opt/tibco`.

How to Contact TIBCO Support

Get an overview of TIBCO Support. You can contact TIBCO Support in the following ways:

- For accessing the Support Knowledge Base and getting personalized content about products you are interested in, visit the TIBCO Support website.
- For creating a Support case, you must have a valid maintenance or support contract

with TIBCO. You also need a user name and password to log in to TIBCO Support website. If you do not have a user name, 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

This 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 this software version 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. TIBCO SOFTWARE INC. MAY MAKE IMPROVEMENTS AND/OR CHANGES IN THE PRODUCT(S) AND/OR THE PROGRAM(S) DESCRIBED IN THIS DOCUMENT AT ANY TIME.

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 "READ ME" FILES.

This and other products of TIBCO Software Inc. may be covered by registered patents. Please refer to TIBCO's Virtual Patent Marking document (https://www.tibco.com/patents) for details.