

TIBCO ActiveMatrix BusinessWorks™ Plug-in Development Kit Developer's Guide

*Software Release 6.1
November 2015*

Important Information

SOME TIBCO SOFTWARE EMBEDS OR BUNDLES OTHER TIBCO SOFTWARE. USE OF SUCH EMBEDDED OR BUNDLED TIBCO SOFTWARE IS SOLELY TO ENABLE THE FUNCTIONALITY (OR PROVIDE LIMITED ADD-ON FUNCTIONALITY) OF THE LICENSED TIBCO SOFTWARE. THE EMBEDDED OR BUNDLED SOFTWARE IS NOT LICENSED TO BE USED OR ACCESSED BY ANY OTHER TIBCO SOFTWARE OR FOR ANY OTHER PURPOSE.

USE OF TIBCO SOFTWARE AND THIS DOCUMENT IS SUBJECT TO THE TERMS AND CONDITIONS OF A LICENSE AGREEMENT FOUND IN EITHER A SEPARATELY EXECUTED SOFTWARE LICENSE AGREEMENT, OR, IF THERE IS NO SUCH SEPARATE AGREEMENT, THE CLICKWRAP END USER LICENSE AGREEMENT WHICH IS DISPLAYED DURING DOWNLOAD OR INSTALLATION OF THE SOFTWARE (AND WHICH IS DUPLICATED IN THE LICENSE FILE) OR IF THERE IS NO SUCH SOFTWARE LICENSE AGREEMENT OR CLICKWRAP END USER LICENSE AGREEMENT, THE LICENSE(S) LOCATED IN THE "LICENSE" FILE(S) OF THE SOFTWARE. USE OF THIS DOCUMENT IS SUBJECT TO THOSE TERMS AND CONDITIONS, AND YOUR USE HEREOF SHALL CONSTITUTE ACCEPTANCE OF AND AN AGREEMENT TO BE BOUND BY THE SAME.

This document contains confidential information that is subject to U.S. and international copyright laws and treaties. No part of this document may be reproduced in any form without the written authorization of TIBCO Software Inc.

TIBCO, Two-Second Advantage, TIBCO ActiveMatrix BusinessWorks, TIBCO Business Studio, TIBCO Enterprise Administrator, and TIBCO ActiveMatrix BusinessWorks Plug-in Development Kit are either registered trademarks or trademarks of TIBCO Software Inc. in the United States and/or other countries.

Enterprise Java Beans (EJB), Java Platform Enterprise Edition (Java EE), Java 2 Platform Enterprise Edition (J2EE), and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle Corporation in the U.S. and other countries.

All other product and company names and marks mentioned in this document are the property of their respective owners and are mentioned for identification purposes only.

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.

Copyright © 2014-2015 TIBCO Software Inc. ALL RIGHTS RESERVED.

TIBCO Software Inc. Confidential Information

Contents

TIBCO Documentation and Support Services	6
Product Overview	7
Installing BusinessWorks Plug-in Development Kit	9
Getting Started	11
Defining a HelloWorld Palette	11
Adding Business Logic	16
Running the HelloWorld Plug-in	18
Packaging the HelloWorld Plug-in	19
Generated Code	21
Plug-in Bundles	21
Design-Time Bundle	21
Model Bundle	23
Runtime Bundle	24
Design-Time Class Specification	26
[PaletteName]	26
[PaletteName]ExceptionsSchema	26
[ActivityName]General/Advanced/customizedSection	27
[ActivityName]ModelHelper	27
[ActivityName]Schema	27
[ActivityName]Signature	28
Runtime Class Specification	28
[ActivityName]EventSource	29
[ActivityName]AsynchronousActivity	30
[ActivityName]SynchronousActivity	32
Target Platform	35
Creating a Plug-in	36
Defining a Palette	36
Adding and Configuring Activities	37
Activity Types	40
Creating Schema with XSD/WSDL	41
Creating Schema with XSD Editor	44
Adding Business Logic	45
Creating Java Global Instance Shared Resource	48
Adding Java Global Instance Shared Resource	48
Adding Business Logic	49
Creating a Process	50
Configuring Java Global Instance Shared Resource	51
Creating Documentation	53

Editing a Plug-in	56
Merging Code	57
Editing an Activity	57
Updating Schema	58
Adding an Activity	60
Deleting an Activity	60
Testing a Plug-in	61
Creating an Installer for a Plug-in	62
Exporting Features	62
Generating an Installer	63
Using a Plug-in	65
Installing and Uninstalling a Created Plug-in	65
Installing a Created Plug-in	66
Uninstalling a Created Plug-in	67
Running the Plug-in	68
Deploying an Application	68
Working with the Sample Projects	70
GSON	70
Importing the GSON Sample Project	70
Importing the JavaToJSON Process	71
Running the JavaToJSON Process	72
LinkedIn	73
Importing the LinkedIn Sample Project	73
Importing the LinkedIn Processes	74
Generating an Access Token and a Token Secret	74
Running the LinkedIn Processes	75
Running the Retrieve Process	75
Running the RetrieveDefaultProfile Process	76
Running the Update Process	77
Managing Logs for a Created Plug-in	79
Log Levels	79
Setting Up a Log Level	79
Exporting Logs to a File	80
Frequently Asked Questions	81
How to Get Input at Run Time	81
How to Create and Update Output at Run Time	81
How to Add Third-Party Libraries	82
How to Add Online Help for a Palette	82
How to Add License	83
How to Add an Activity Icon	83

How to Find BusinessWorks API JavaDoc	84
Troubleshooting	85
General Problems	85
Migration Problems	87

TIBCO Documentation and Support Services

Documentation for this and other TIBCO products is available on the TIBCO Documentation site:

<https://docs.tibco.com>

Documentation on the TIBCO Documentation site is updated more frequently than any documentation that might be included with the product. To ensure that you are accessing the latest available help topics, please visit <https://docs.tibco.com>.

Product-Specific Documentation

Documentation for TIBCO products is not bundled with the software. Instead, it is available on the TIBCO Documentation site. To directly access documentation for this product, double-click the following file:

`TIBCO_HOME/release_notes/TIB_bwpgk_version_docinfo.html`

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

The following documents for this product can be found on the TIBCO Documentation site:

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

How to Contact TIBCO Support

For comments or problems with this manual or the software it addresses, contact TIBCO Support:

- For an overview of TIBCO Support, and information about getting started with TIBCO Support, visit this site:

<http://www.tibco.com/services/support>

- If you already have a valid maintenance or support contract, visit this site:

<https://support.tibco.com>

Entry to this site requires a user name and password. If you do not have a user name, you can request one.

How to Join TIBCOCommunity

TIBCOCommunity is an online destination for TIBCO customers, partners, and resident experts. It is a place to share and access the collective experience of the TIBCO community. TIBCOCommunity offers forums, blogs, and access to a variety of resources. To register, go to the following web address:

<https://www.tibcommunity.com>

Product 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:



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](#).

Installing Plug-in Development Kit




TIBCO ActiveMatrix BusinessWorks Plug-in Development Kit is packaged as a TIBCO Eclipse Plug-in, which is an installation package for the Eclipse provisioning platform.

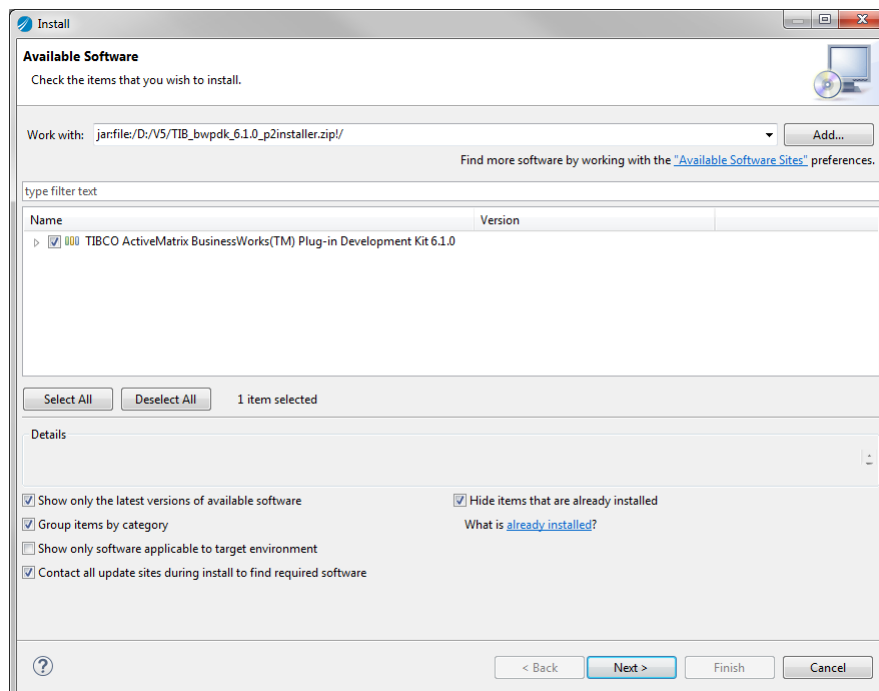
You can use Eclipse Update Manager to install TIBCO ActiveMatrix BusinessWorks Plug-in Development Kit.

Prerequisites

Ensure that you have installed TIBCO ActiveMatrix BusinessWorks.

Procedure

1. Download the installation package of TIBCO ActiveMatrix BusinessWorks Plug-in Development Kit.
2. 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.
3. From the menu, click **Help > Install New Software**.
4. In the Available Software dialog, click **Add**. In the Add Repository dialog, click **Archive** and locate the .zip file downloaded in [Step 1](#). Click **OK**.
5. Select the **TIBCO ActiveMatrix BusinessWorks(TM) Plug-in Development Kit 6.1.0** check box. Click **Next**.



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





Getting Started

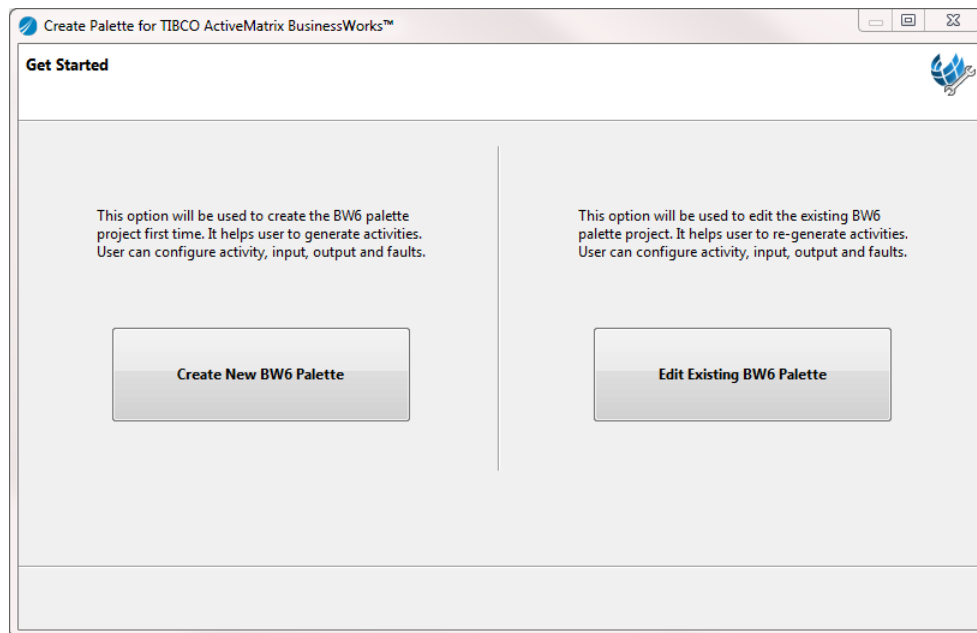
You can begin to use BusinessWorks Plug-in Development Kit by creating a HelloWorld plug-in. For simplicity, the HelloWorld plug-in contains only one activity, which responds with the "Hello World" string.

Defining a HelloWorld Palette


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

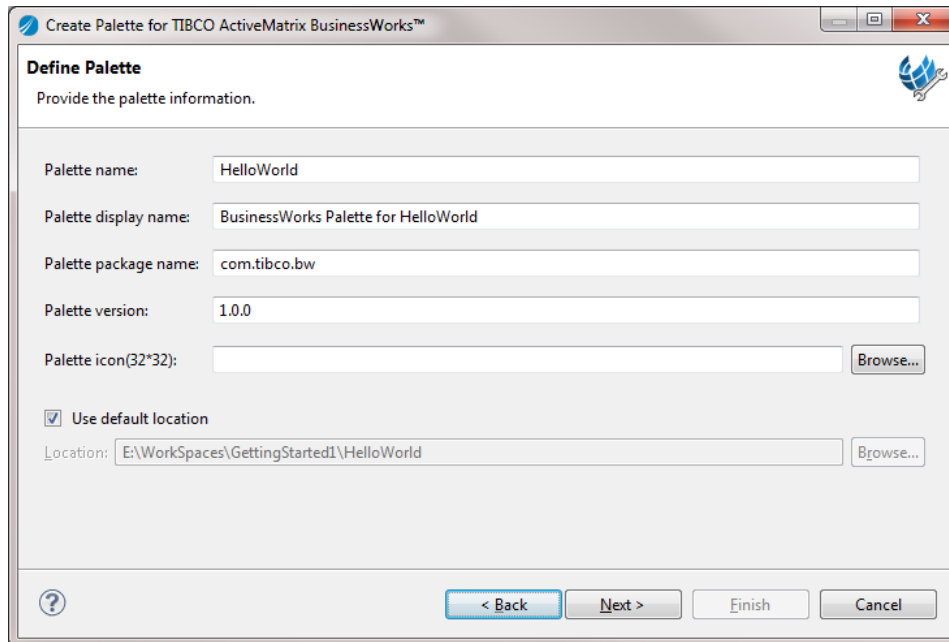
Procedure

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 **Create a new or modify BW Plug-in Project** .
 - Press Alt+T.



3. In the Get Started dialog, 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**.



Create Palette for TIBCO ActiveMatrix BusinessWorks™

Define Palette
Provide the palette information.

Palette name:

Palette display name:


Palette package name:

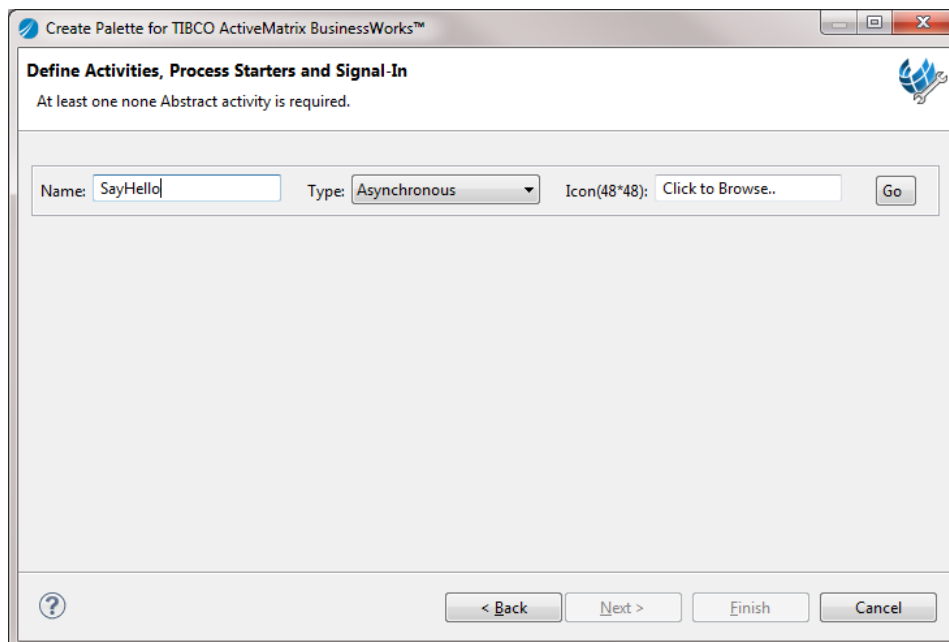
Palette version:

Palette icon(32*32):

☒ Use default location

Location:


- 4. In the Define Activities, Process Starters and Signal-In dialog, 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 use another icon, click **Browse** to locate the icon. Click **Go** to configure the activity.
The size of the icon cannot be less than 48x48.



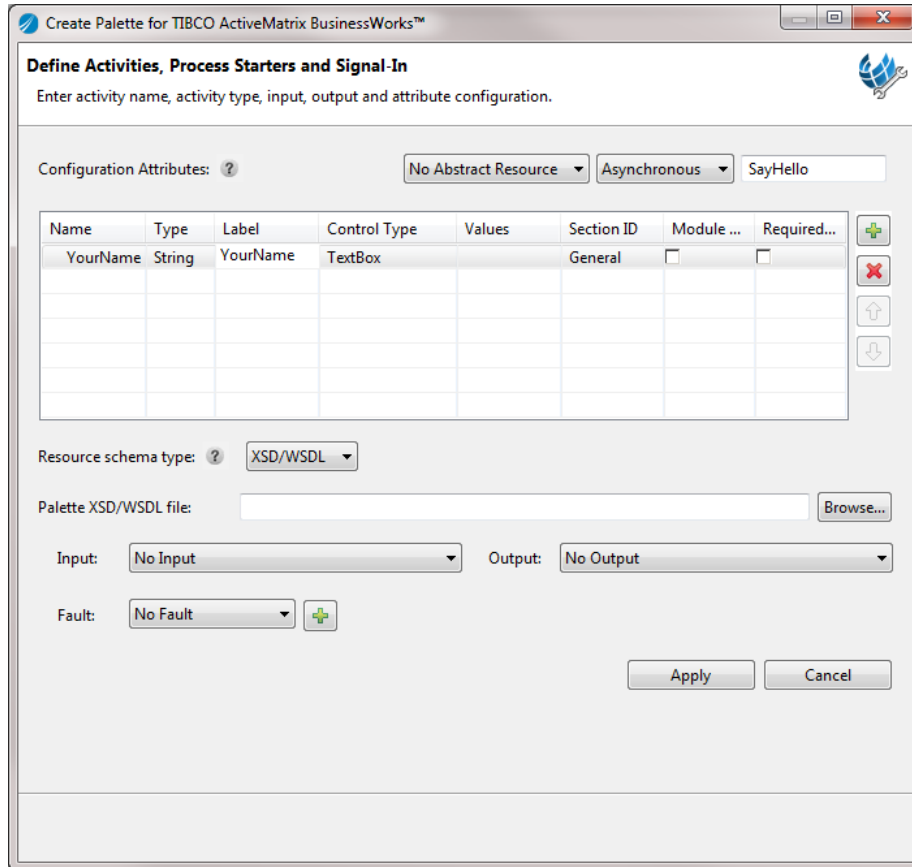
Create Palette for TIBCO ActiveMatrix BusinessWorks™

Define Activities, Process Starters and Signal-In
At least one none Abstract activity is required.

Name: Type: Icon(48*48):

d) Click  to add an attribute for the SayHello activity and configure the attribute as follows:

- Enter `YourName` as the value of the **Name** attribute.
- Select `String` as the value of the **Type** attribute.
- Enter `YourName` 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.



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	<input type="checkbox"/>	<input type="checkbox"/>


Resource schema type: ? XSD/WSDL

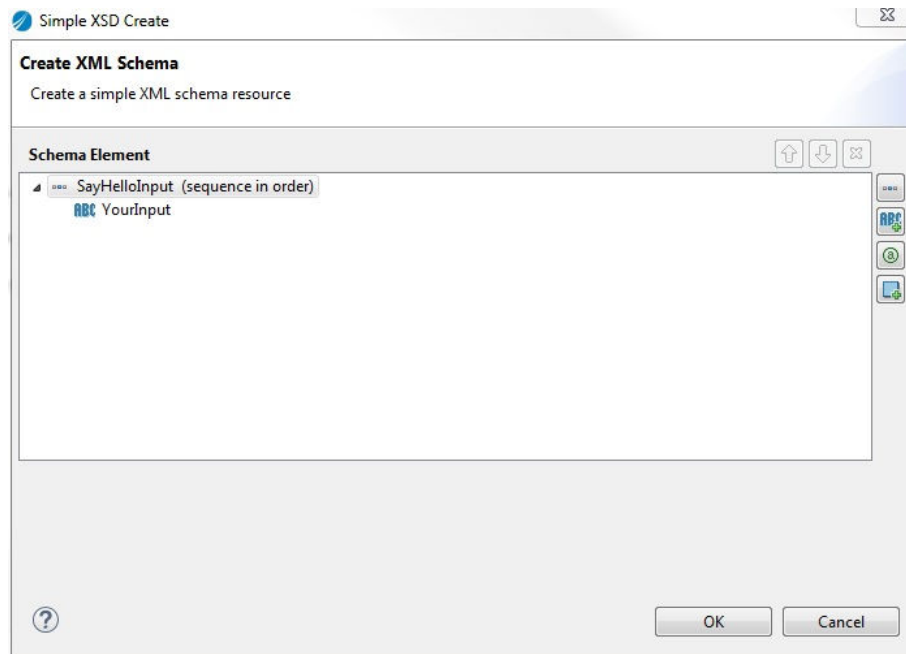
Palette XSD/WSDL file: Browse...


Input: No Input Output: No Output

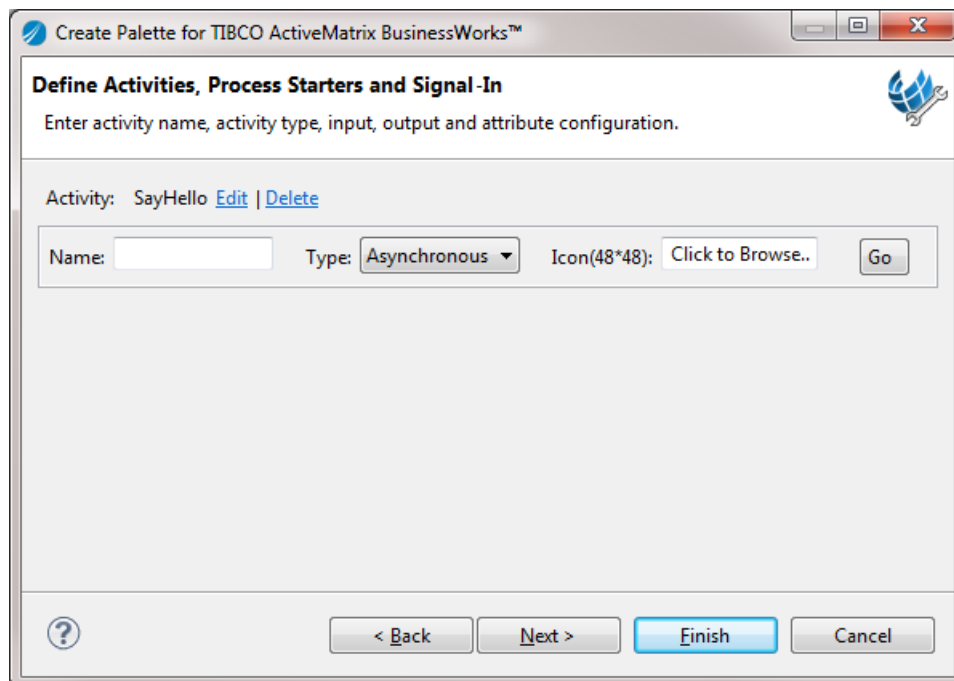
Fault: No Fault 

Apply Cancel

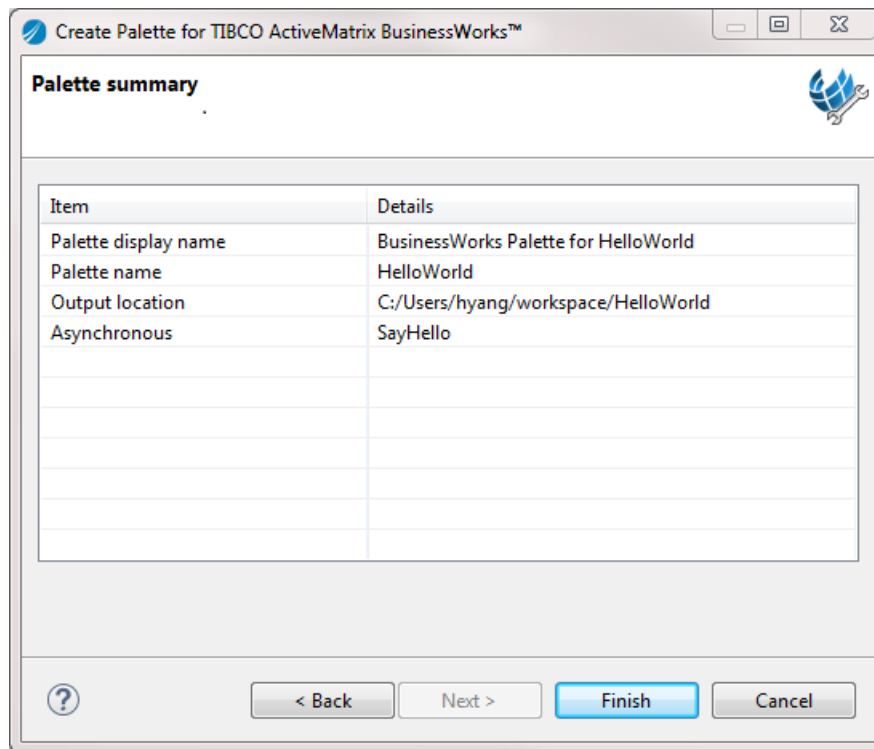
- e) From the **Resource schema type** list, select **XSD Editor**.
- f) Click **Input** to create an input schema for the activity.
- g) In the Create XML Schema dialog, click  to add a primitive element, and then set the element name to `YourInput`. Click **OK**.



- h) Click **Output** to create an output schema for the activity. In the Create XML Schema dialog, click  to add a primitive element, and then set the element name to Output. 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. The configured SayHello activity is displayed.



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



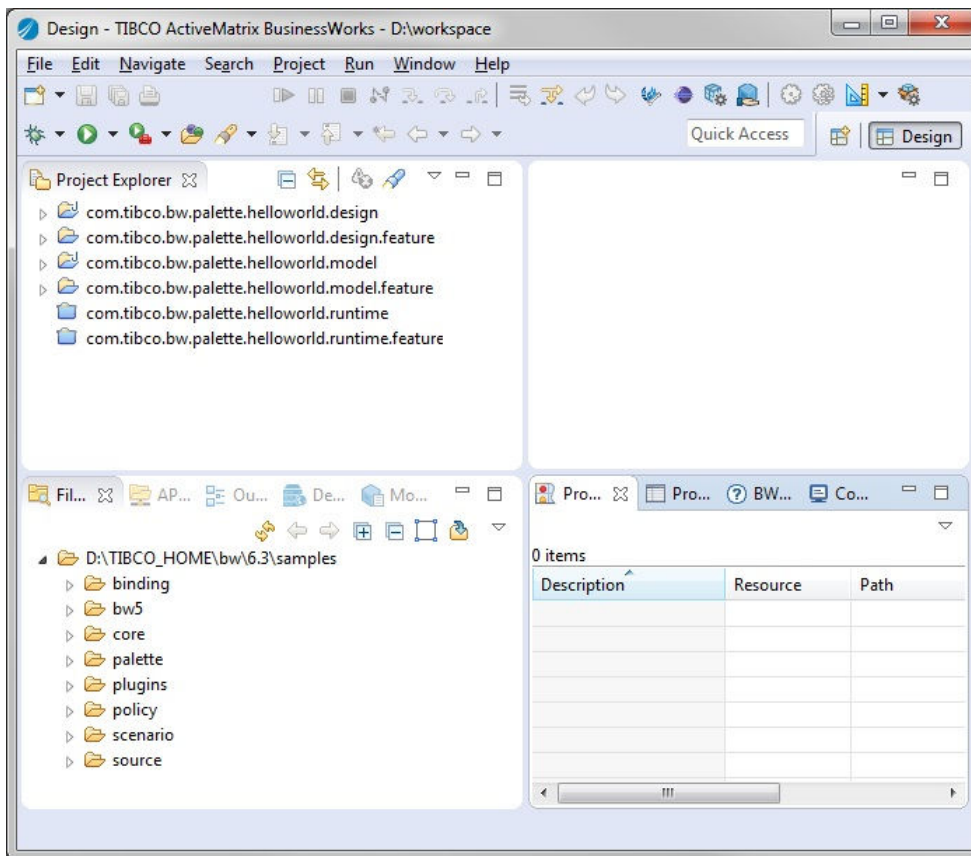
8. Change the target platform to running platform for the design-time module.
 - a) Click **Window > Preferences**, and then click **Plug-in Development > Target Platform**.
 - b) Click **Add** to add a running platform for the design-time module.
 - c) In the Target Definition dialog, click **Default** to choose the running platform. Click **Next**.
 - d) In the Target Content dialog, click **Finish**.
 - e) You are back to the Target Platform dialog, 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.

9. Close the runtime bundle and feature, and open the design bundle and feature.

Result

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



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.

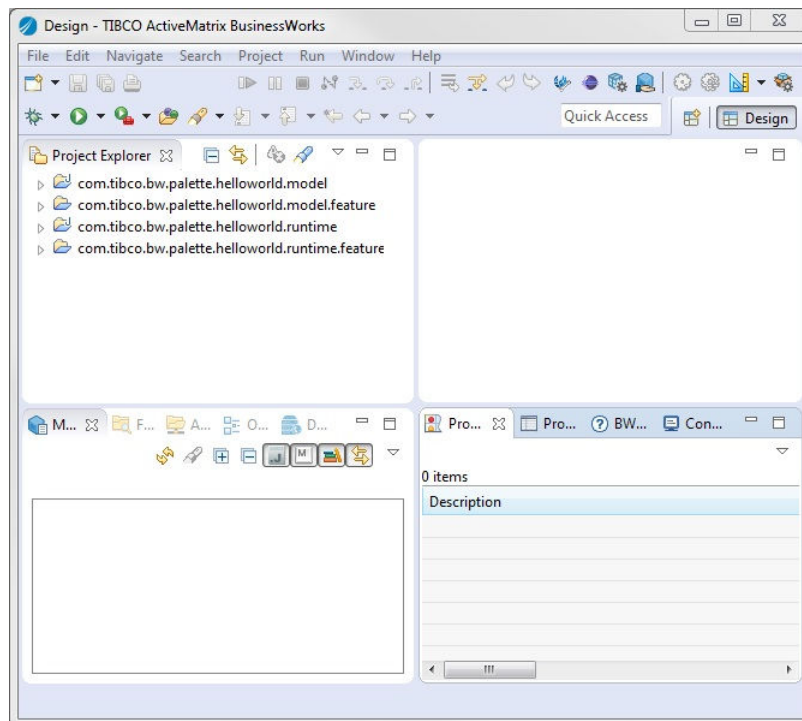
Prerequisites

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, double-click **Eclipse Application** in the left panel, and then click **New_configuration**.
 - b) Click the (x)= **Arguments** tab, and then enter the following parameters in the **VM arguments** field. Click **Apply**.
 -  Microsoft Windows: -Dorg.osgi.framework.bootdelegation=javax.xml.* -XX:MaxPermSize=256m -XX:+UnlockDiagnosticVMOptions -XX:+UnsyncloadClass -XX:+UseParNewGC -Xms512m -Xmx768m

-  Linux: `-Dorg.osgi.framework.bootdelegation=javax.xml.* -XX:MaxPermSize=256m -XX:+UnlockDiagnosticVMOptions -XX:+UnsyncloadClass -XX:+UseParNewGC -Xms512m -Xmx768m`
 -  Mac OS: `-XstartOnFirstThread -Dorg.osgi.framework.bootdelegation=javax.xml.* -XX:MaxPermSize=512m -XX:+UnlockDiagnosticVMOptions -XX:+UnsyncloadClass -XX:+UseParNewGC -Xms512m -Xmx768m`
- c) 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, expand the **General** folder and select **Existing Projects into Workspace**. Click **Next**.
 - b) In the Import Projects dialog, 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.



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 SayHelloAsynchronousActivity.java file, find the evalOutput() method.
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("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;
}
```

6. Save the SayHelloAsynchronousActivity.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.

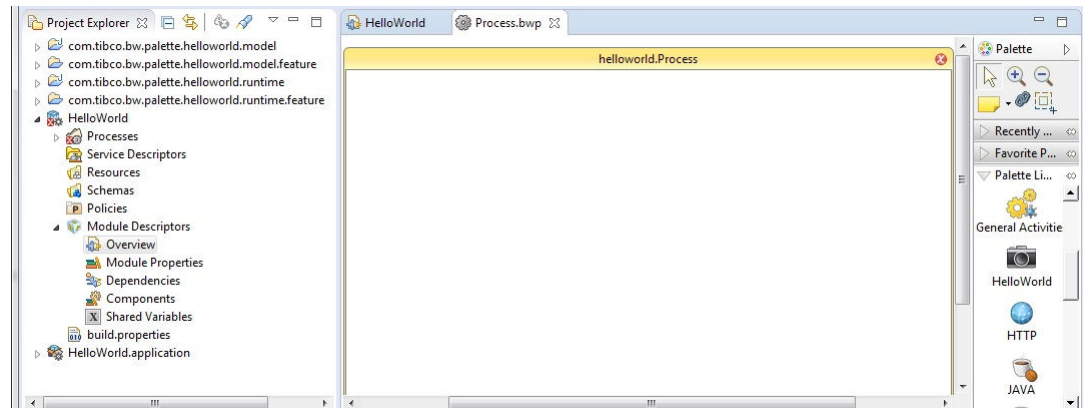
Prerequisites


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, click **BusinessWorks Application Module**. Click **Next**.
3. In the Project dialog, 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.



The  icon is displayed only when you select a Timer activity.

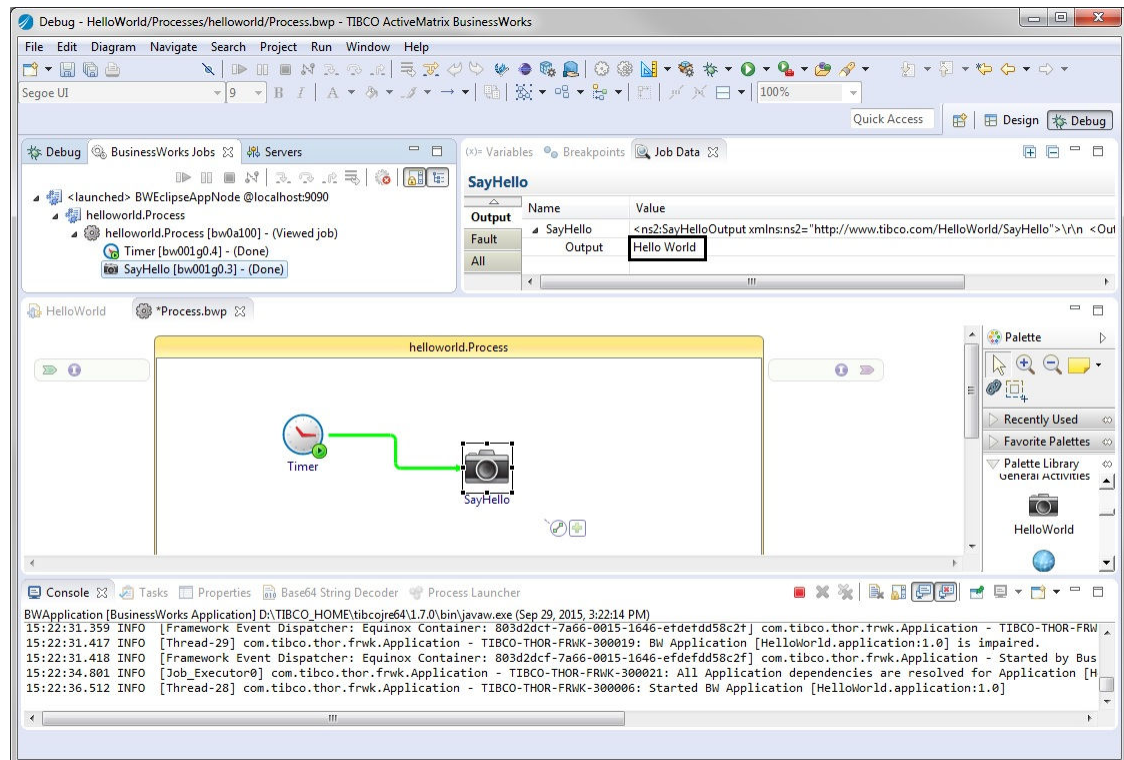
6. In the process editor, double-click the SayHello activity:
 - a) In the **General** tab, enter AAA in the **YourName** field.
 - b) In the **Input** tab, enter AAA as the value of the YourInput element.
7. On the toolbar, click  to save your configurations.

8. From the menu, click **Run > Debug Configurations**. In the "Create, manage, and run configurations" dialog, 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.



What to do next

Packaging the HelloWorld Plug-in

Packaging the HelloWorld Plug-in

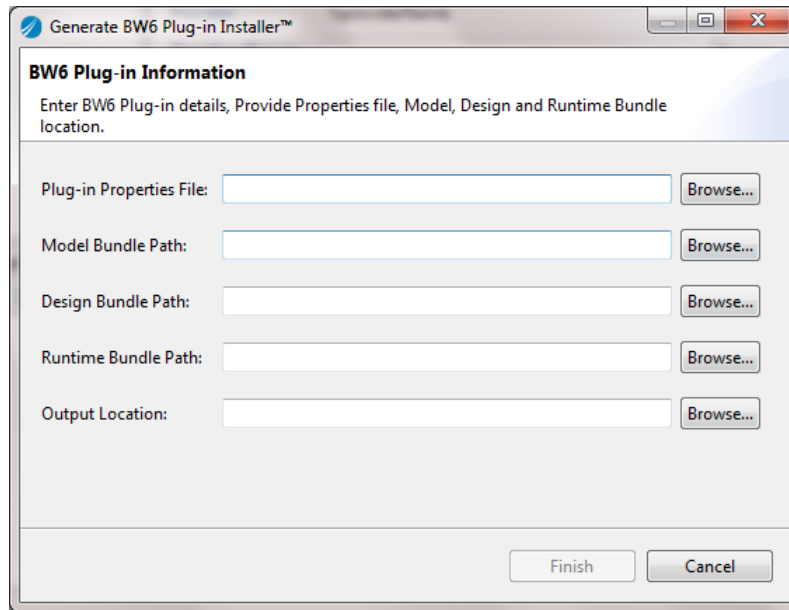
You can generate a provisioning platform (p2) installer for the created HelloWorld plug-in.

Prerequisites

Ensure that you have exported the design, model, and runtime features of the plug-in to three folders one by one. See [Exporting Features](#) for more details.

Procedure

1. In the parent TIBCO Business Studio with the running platform selected, click **Help > Create BusinessWorks Plug-in Installer**.
2. In the BW6 Plug-in Information dialog, provide the following information:



- **Plug-in Properties File**
Click **Browse** to locate the `bw6_devkit_configuration.properties` file that is located in the HelloWorld project folder.
- **Model Bundle Path**
Click **Browse** to locate the folder that contains the model feature exported in the [Exporting Features](#) section.
- **Design Bundle Path**
Click **Browse** to locate the folder that contains the design-time feature exported in the [Exporting Features](#) section.
- **Runtime Bundle Path**
Click **Browse** to locate the folder that contains the runtime feature exported in the [Exporting Features](#) section.
- **Output Location**
Click **Browse** to select the folder where the generated plug-in installer is saved.

What to do next

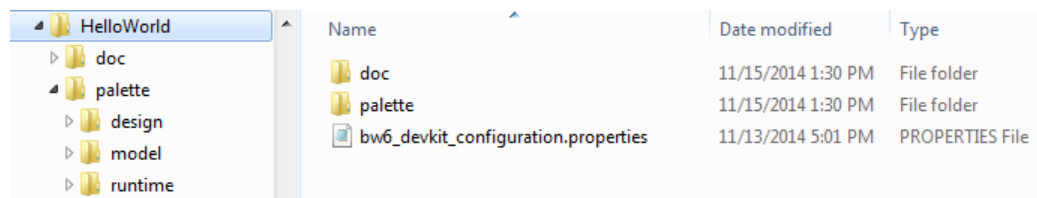
[Using a Plug-in](#)

Generated Code

TIBCO ActiveMatrix BusinessWorks Plug-in Development Kit is a tool for generating a code template of a BusinessWorks plug-in.

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 `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.



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.

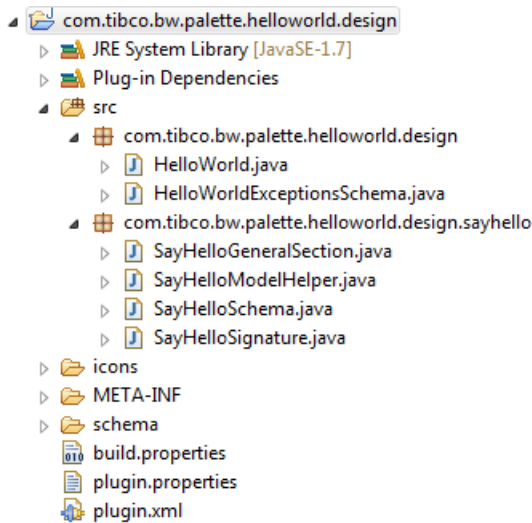
After [adding and configuring activities](#) for your plug-in, you have to change the target platform to design time. When the code generation is completed, the following bundles are generated and displayed in the Project Explorer view:



- [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:



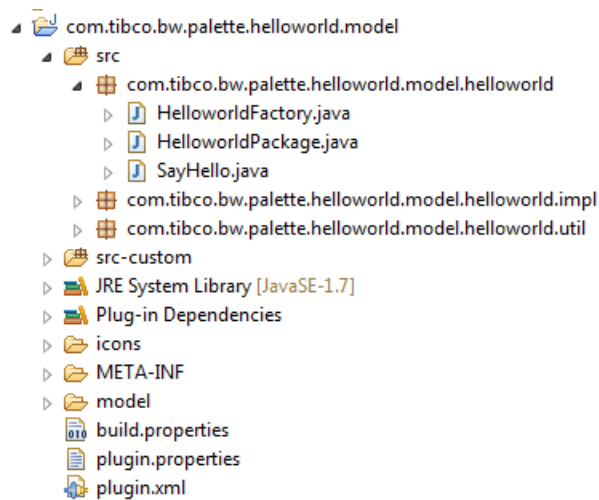
Folders and Files	Description
src	<p>Contains the source code of the design-time configurations.</p> <p>A palette package is created, which contains the following Java files:</p> <ul style="list-style-type: none"> The <i>Palette_Name.java</i> file contains the methods to access your plug-in project. The <i>Palette_NameExceptionsSchema.java</i> file contains the fault schema related methods. <div style="display: flex; align-items: center;">  <div style="margin-left: 10px;"> <p>This file is generated only when you do <i>not</i> configure any fault schema.</p> </div> </div> <p>Besides, a separate package is created corresponding to each activity. Each activity package contains the following Java files:</p> <ul style="list-style-type: none"> The <i>ActivityNameGeneral/Advanced/customizedSection.java</i> 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 <i>ActivityNameModelHelper.java</i> file contains the source code to initiate a model. The <i>ActivityNameSchema.java</i> file contains the input and output schema. <div style="display: flex; align-items: center;">  <div style="margin-left: 10px;"> <p>This file is generated when using an XSD file or a WSDL file or using XSD Editor to generate the activity input and output.</p> </div> </div> <ul style="list-style-type: none"> The <i>ActivityNameSignature.java</i> 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.

Folders and Files	Description
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:

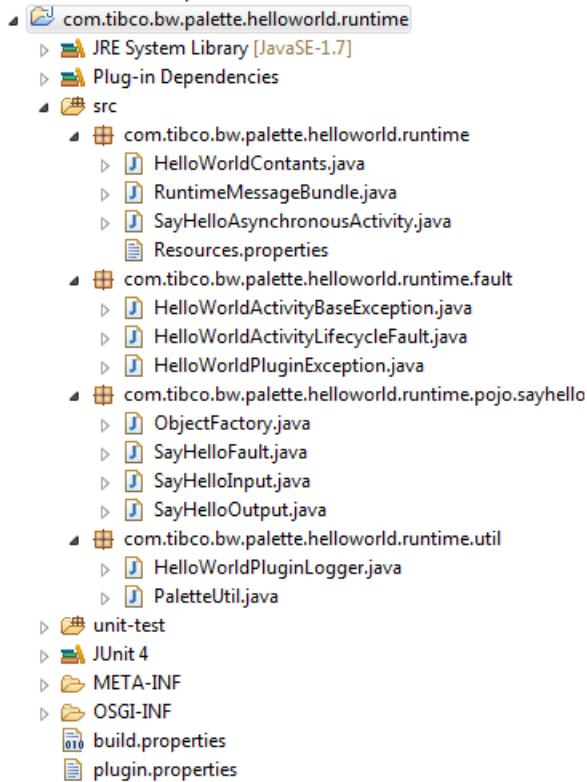



Folders and Files	Description
src	Contains the source code of the data model. Use the interface and corresponding implementation files of the activity to create a data model.
src-custom	Contains the implementation code of the activity validator.
icons	Contains the activity icon to be displayed in the 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.
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:



Folders and Files	Description
src	<p>Contains the source code to implement the business logic of each activity.</p> <ul style="list-style-type: none"> The runtime package contains the following files: <ul style="list-style-type: none"> The <i>PaletteNameConstants.java</i> file contains the defined constants used in the code. The <i>RuntimeMessageBundle.java</i> file contains the bundle message definition. The <i>ActivityNameActivityType.java</i> 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 <i>Resources.properties</i> file contains the implementation of bundle messages and error codes. The fault package contains the following files: <ul style="list-style-type: none"> The <i>PaletteNameActivityBaseException.java</i> file contains the errors specified for the synchronous and asynchronous activities. The <i>PaletteNameActivityLifecycleFault.java</i> file contains initialization errors. The <i>PaletteNamePluginException.java</i> file contains the error defined by BusinessWorks Plug-in Development Kit by default. The <i>FaultNameFault.java</i> file contains the generated fault schema according to the design-time configurations. Use the private <code><N, A> N constructErrData ()</code> method and the public <code>QName getFaultElementQName()</code> method to add other faults at run time. The private <code><N, A> N constructErrData ()</code> 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. <div data-bbox="564 1318 608 1367">  </div> <div data-bbox="678 1318 1390 1381"> <p>The <i>PaletteNameActivityBaseException.java</i> file is not available for the signal-in and process starter activities.</p> </div> <ul style="list-style-type: none"> The pojo package contains Java Architecture for XML Binding (JAXB) generated common Java classes according to the output configurations. You can assign values to these POJO classes. The util package contains the following files: <ul style="list-style-type: none"> The <i>PaletteNamePluginLogger.java</i> file contains the generated logging utility. The <i>PaletteUtil.java</i> file contains the generated output utility.
unit-test	Contains the source code for unit tests.
META-INF	Contains a <i>MANIFEST.MF</i> file that provides information regarding the plug-in bundle and package.

Folders and Files	Description
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
<code>public void start()</code>	This method is called when the design-time bundle is initialized by Eclipse. This is a BusinessWorks 6 life-cycle method.
<code>public void stop()</code>	This method is called when the design-time bundle is stopped by Eclipse. This is a BusinessWorks 6 life-cycle method.
<code>public static <i>palette_name</i> getDefault()</code>	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 getSchemaAsStream()	Use this method to get the input stream of the fault schema. This is a BusinessWorks 6 life-cycle method.
public static List<XSDElementDeclaration> getplug-in_nameFaultTypes()	Use this method to get the fault type.
private static List<XSDElementDeclaration> getFaultElements()	Use this method to get the fault elements.

[ActivityName]General/Advanced/customizedSection

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

The [ActivityName]General/Advanced/customizedSection 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 <i>customized</i> tabs. This is a BusinessWorks 6 life-cycle method.



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.

Methods	Description
<pre>public static List<XSDElementDeclaration> getFaultElements()</pre>	<p>Use this method to get the fault element.</p> <p> If you do not specify any fault schema at design time, this method is not generated.</p>
<pre>protected InputStream getSchemaAsStream()</pre>	<p>Use this method to get the input stream.</p> <p>This is a BusinessWorks 6 life-cycle method.</p>

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

Runtime Class Specification

Use runtime classes to add business logic for each activity.

The following runtime classes are generated:

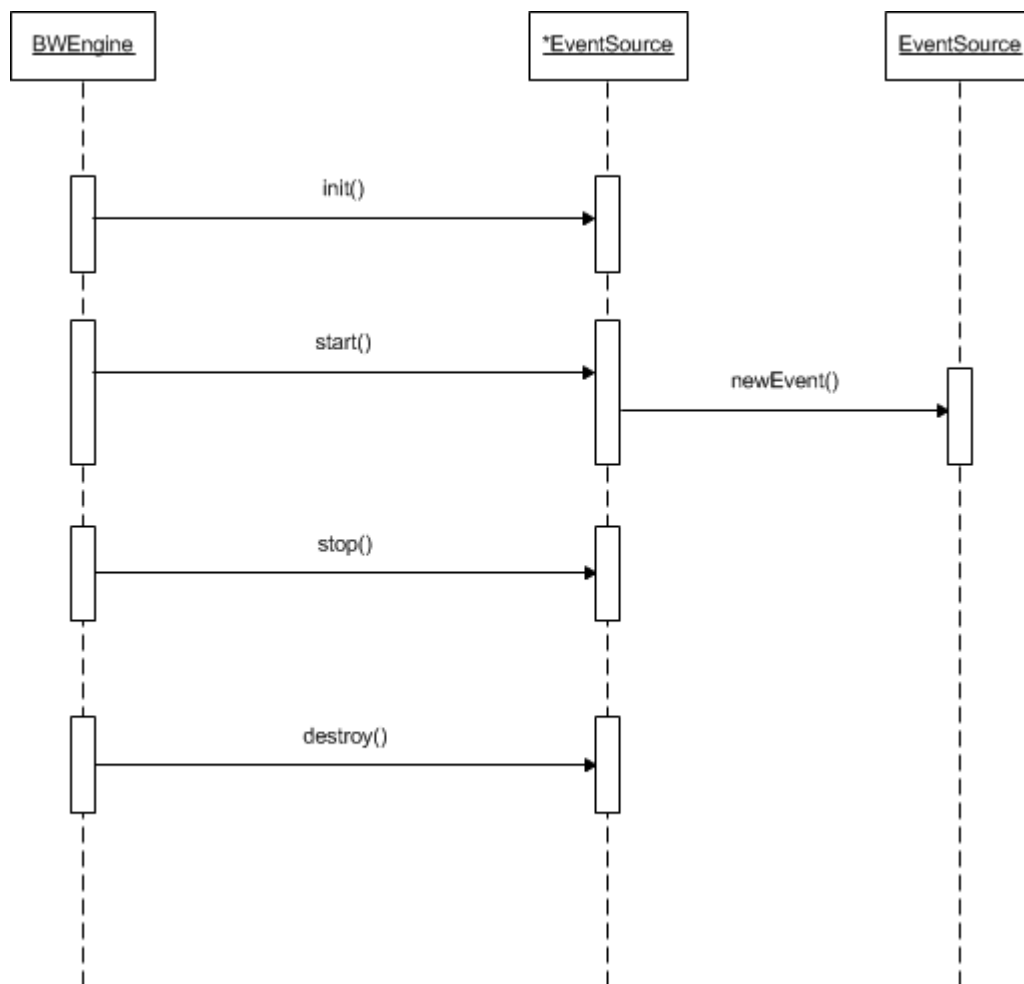
- [\[ActivityName\]EventSource](#)
- [\[ActivityName\]AsynchronousActivity](#)
- [\[ActivityName\]SynchronousActivity](#)

[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:

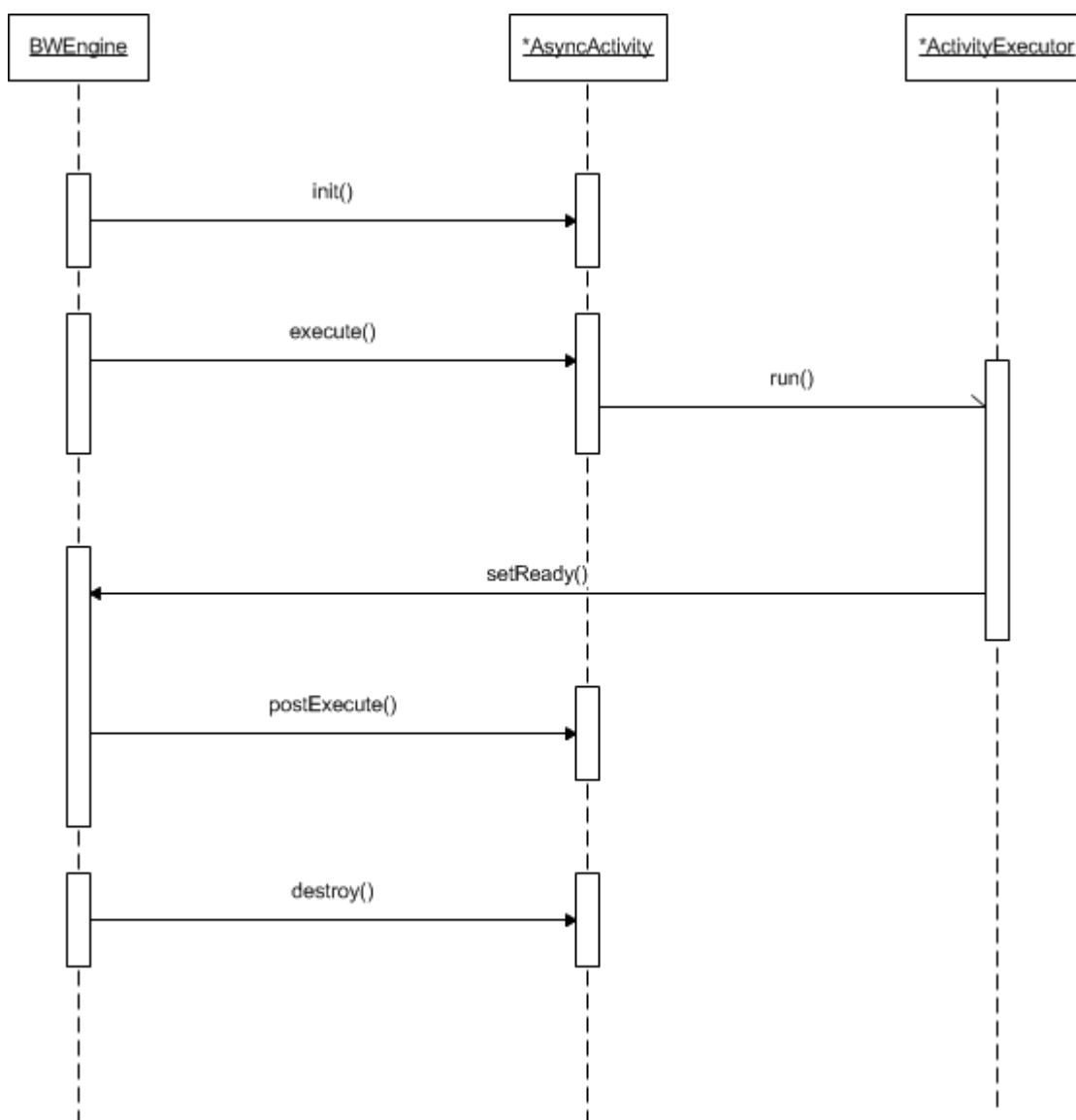
Methods	Description
public synchronized void destroy()	Use this method to release or clean resources held by a source, when an event source is destroyed. This is a BusinessWorks 6 life-cycle method.
public synchronized boolean isStarted()	Returns a boolean value indicating the status of an event source. This is a BusinessWorks 6 life-cycle method.

Methods	Description
public synchronized void start()	<p>Use this method to start an event source. You cannot start a new event until this method is called.</p> <p>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.</p> <p>This is a BusinessWorks 6 life-cycle method.</p>
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()	<p>Use this method to stop the event source from processing new events.</p> <p>When this method is called, the event source cannot use the {@link EventSourceContext} interface to notify the BusinessWorks engine of a new event.</p> <div>  <p>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 EventSource#stop()} method.</p> </div> <p>This is a BusinessWorks 6 life-cycle method.</p>
public void init()	<p>Use this method to perform any required initialization.</p> <p>The <code><code>eventSourceKind</code></code> argument of this method indicates that the event source is being initialized by a process starter activity or a signal-in activity.</p> <div>  <p>Do not use this method to start an event source. You can start the event source until the {@link EventSource#start()} method is called.</p> </div> <p>This is a BusinessWorks 6 life-cycle method.</p>

[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:

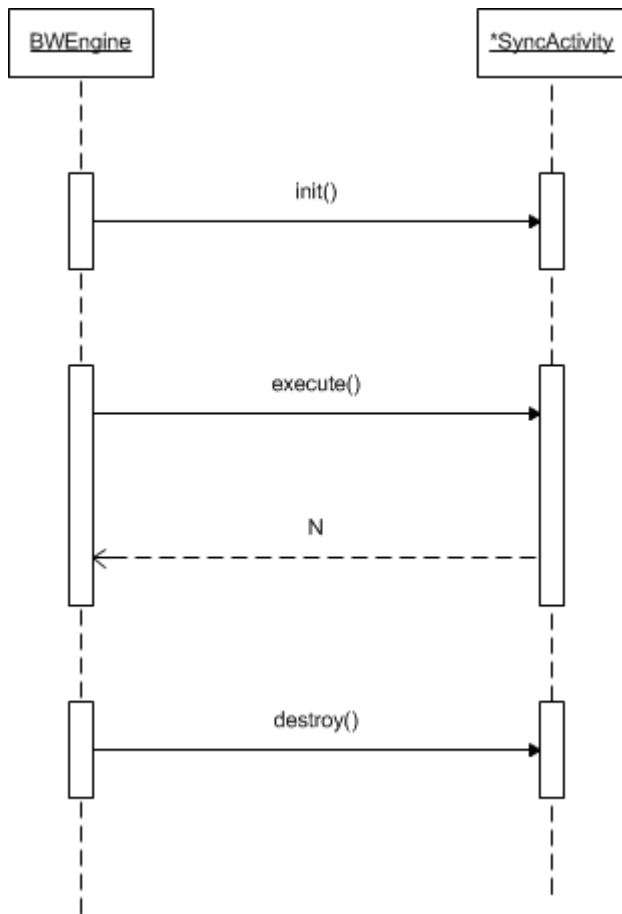
Methods	Description
public void init()	Use this method to initialize the activity. This is a BusinessWorks 6 life-cycle method.
public void destroy()	Use this method to release or clean resources held by a source, when an event source is destroyed. This is a BusinessWorks 6 life-cycle method.

Methods	Description
public void cancel()	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.
public void execute()	Use this method to execute an asynchronous activity.  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.
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.  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]AsyncActivityExecutor<A> internal class contains the following method:	
public void run()	Use this method to add business logic before calling the <code>evalOutput()</code> 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. 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.

Methods	Description
<pre>public String getInputParameterStringValueByName()</pre>	<p>Use this method to get the value of a string parameter according to the parameter name from the input.</p> <div>  <p>This method cannot retrieve the value of a sub node.</p> </div>
<pre>public String getInputAttributeStringValueByName()</pre>	<p>Use this method to get the value of an attribute according to the parameter name from the input.</p>
<pre>public boolean getInputParameterBooleanValueByName()</pre>	<p>Use this method to get the value of a boolean parameter according to the parameter name from the input.</p>

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 with the plug-ins defined by your target platform.

When you launch TIBCO Business Studio, the bw-runtime platform is used by default. You have to change the platform to the running (design-time) platform when using BusinessWorks Plug-in Development Kit to generate the palette code.

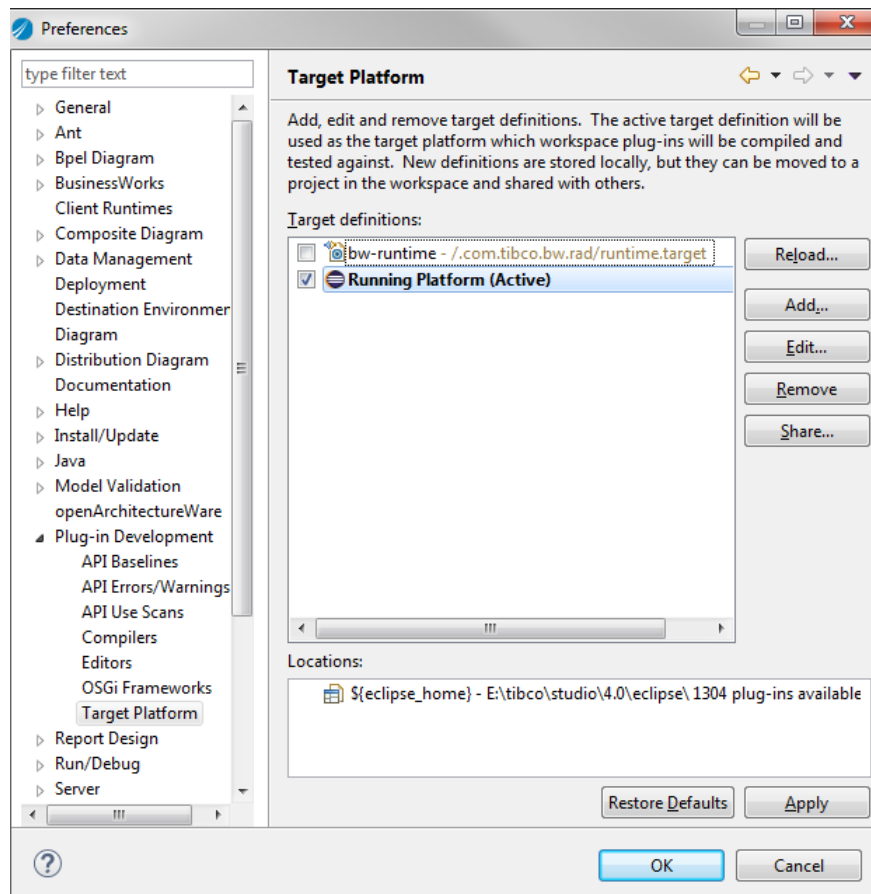
- Running Platform

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 Running Platform.

- bw-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 bw-runtime.

Click **Window > Preferences**, and then click **Plug-in Development > Target Platform**. The target platform in use is selected.



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, configure a palette for the plug-in:

- 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

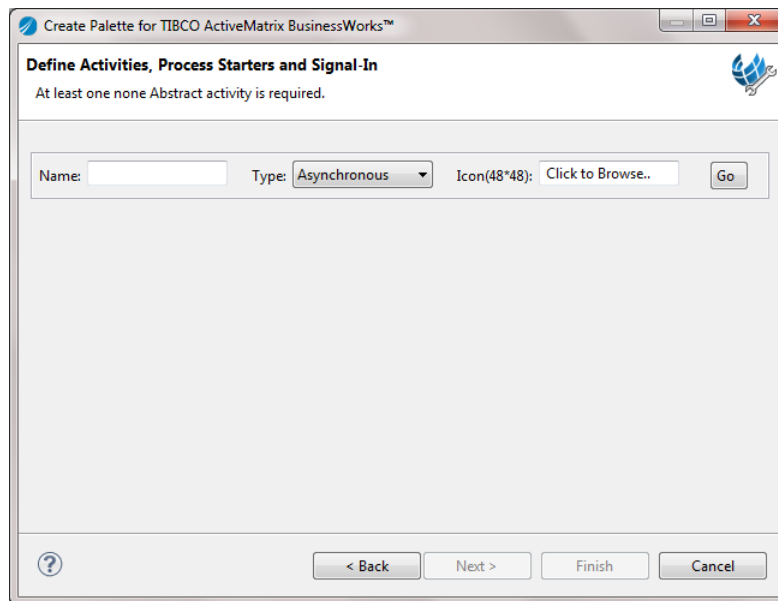
An activity is an individual unit of work in a process. Each activity is represented by an icon in TIBCO Business Studio and consists of a set of configurations.



Prerequisites

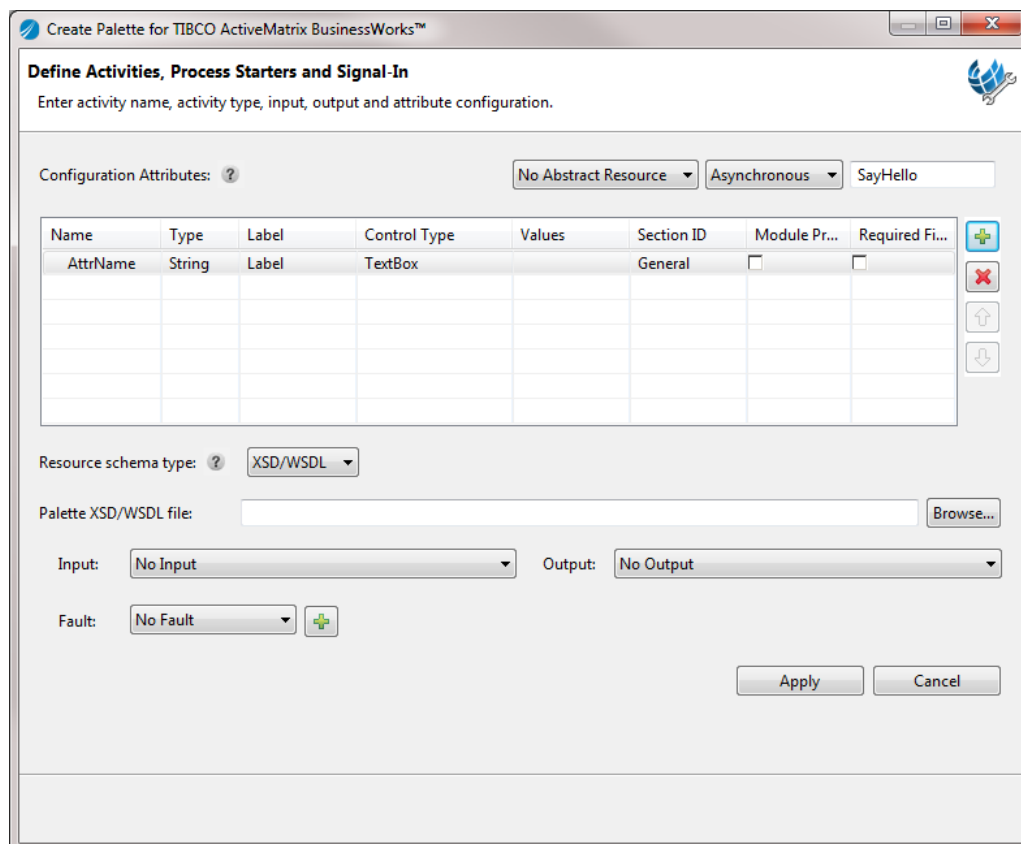
Ensure that you have configured a palette, as described in [Defining a Palette](#).



Procedure

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



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



Properties	Description
Name	Name of the attribute.
Type	<p>Type of the attribute.</p> <p>The following attribute types are supported: Int, Short, String, Float, Double, Long, Boolean, and Date.</p> <div>  <p>The attribute of the Double, Long, Float, or Short type is not displayed in TIBCO Business Studio.</p> </div>
Label	Display name of the attribute. The display name is the field name displayed in TIBCO Business Studio.
Control Type	<p>Control type of the attribute.</p> <p>The following types are supported: Hide, TextBox, FilePickerField, ComboViewer, Button, PasswordField, DirectoryPickerField, ActivityReferenceField, PropertyField.</p> <p>BusinessWorks Plug-in Development Kit automatically selects a control type according to the selected attribute type.</p> <div>  <p>If you set the control type of an attribute to Hide, the attribute of the activity is not displayed on UI.</p> <p>If you want to add a Java global instance shared resource, select PropertyField from this list.</p> </div>
Values	<p>Values to be displayed as default options.</p> <p>Use comma (,) to separate values when the control type is ComboViewer.</p>
Section ID	<p>Name of the section that the attribute belongs to. The following three options are supported:</p> <ul style="list-style-type: none"> • 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.

- 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.

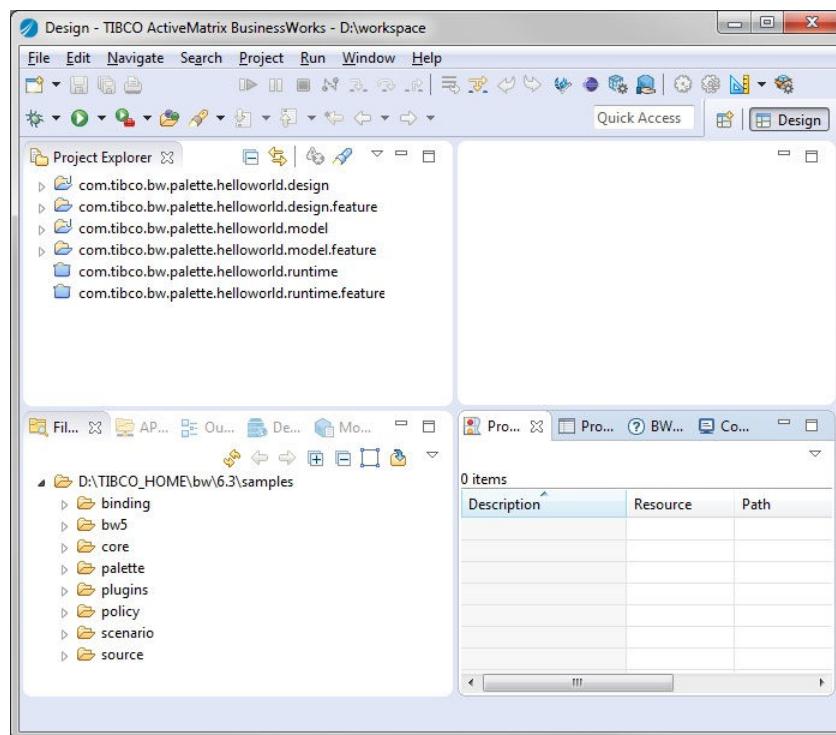
- If you select **None**, no input and output are generated. BusinessWorks Plug-in Development Kit generates default schema for fault.
6. Click **Apply** to save your configurations.
You are now directed to the Define Activities, Process Starters and Signal-In dialog. Go back to [Step 1](#) to add more activities.
 7. Click **Finish** to generate code for the defined palette and activities.
 8. 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, click **Default** to choose the running platform. Click **Next**.
 - d) In the Target Content dialog, click **Finish**.
 - e) You are back to the Target Platform dialog, 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.

9. Close the runtime bundle and feature, and open the design bundle and feature.

Result

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



What to do next

[Adding Business Logic](#)

Activity Types

Activities are the individual units of work in a process.

Activities generally interact with an external system and perform a task. Activity that are used to connect the same external applications 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, 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 `elementFormatDefault` attribute is unqualified.



```
<?xml version='1.0' encoding='UTF-8' ?>
<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"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

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>
```

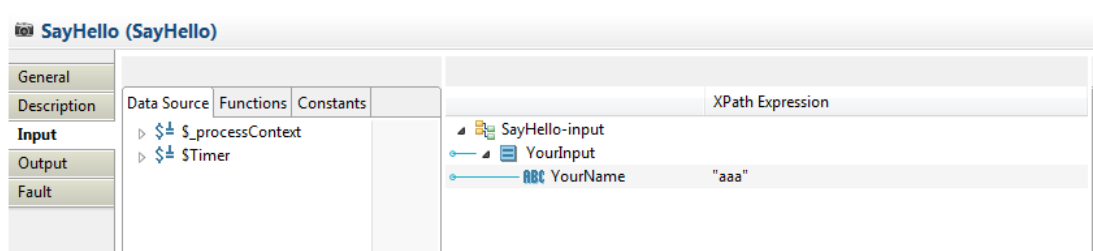
When [adding and configuring activities](#) in the Define Activities, Process Starters and Signal-In dialog, use the example XSD file to configure the activity schema:

- Select the `YourInput` element as the activity input.
- Select the `YourOutput` element as the output.

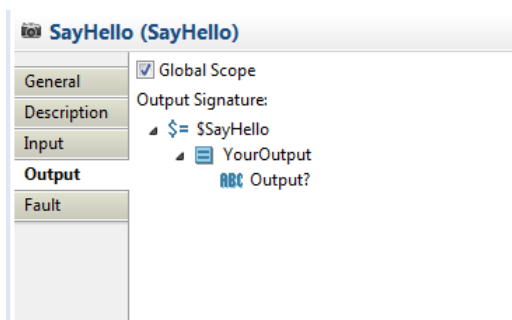
- Select the `InputFileException` element as one fault.
- Select the `PluginException` element as another fault.

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

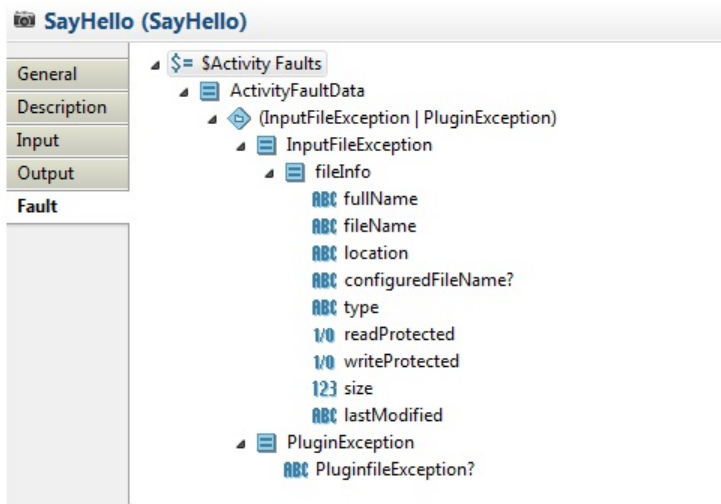
The following example is the generated input schema based on the selected XSD file:



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



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.





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

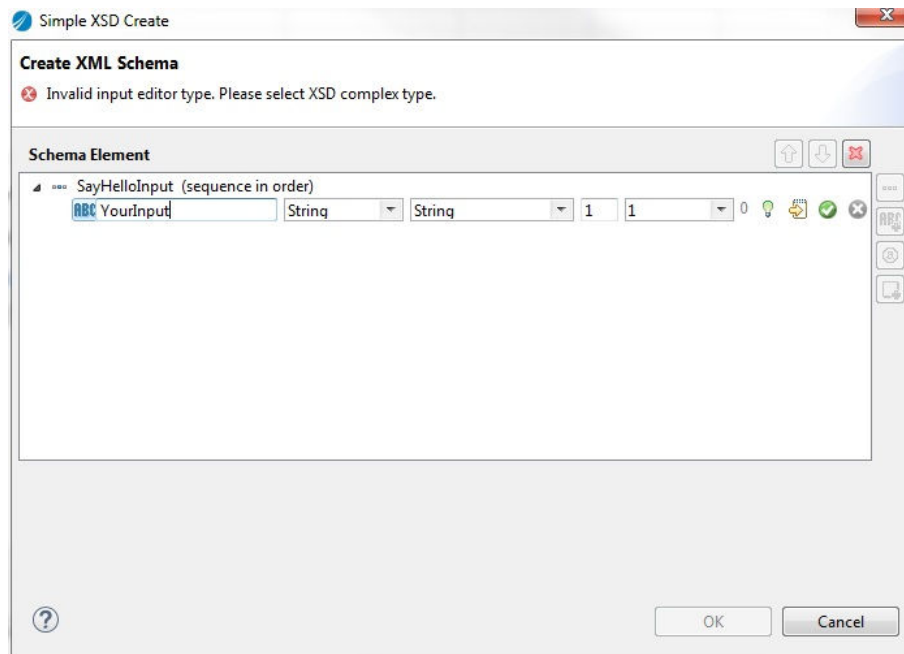


The Choose Element Type function and the number of reference function are not supported when the element type is anyType.

Take the input of the SayHello activity as an example.

Procedure

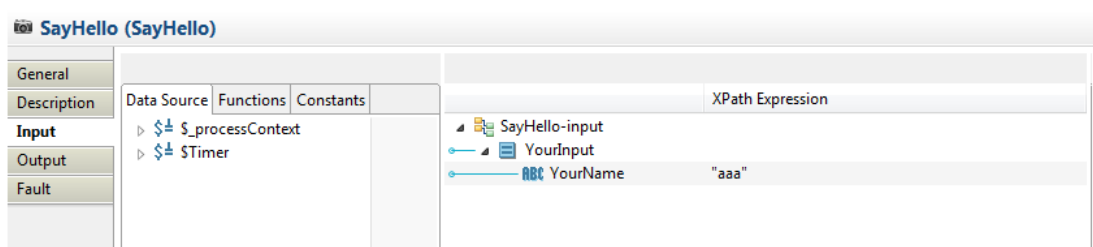
1. In the Simple XSD Create dialog, click **SayHelloInput**.
2. Click  to add a primitive element.
3. Click the added primitive element and enter `YourInput` as the element name. Click  to save your configurations.



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.

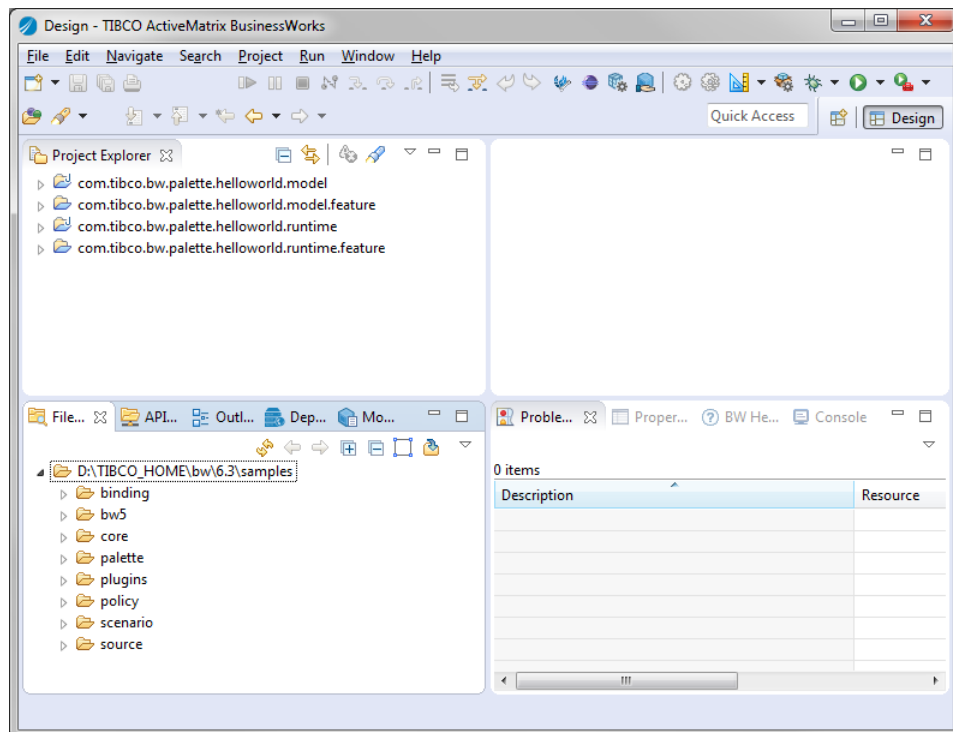
- For a synchronous activity, add your business logic in the [\[ActivityName\]SynchronousActivity](#) class.
- For an asynchronous activity, add your business logic in the [\[ActivityName\]AsynchronousActivity](#) class.
- For a process starter activity or a signal-in activity, add your business logic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

Prerequisites

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, double-click **Eclipse Application** in the left panel to create a new Eclipse application.
You can also use an existing Eclipse application.
3. Click the **(x)= Arguments** tab, and then enter the following parameters in the **VM arguments** field. Click **Apply**.
 -  Microsoft Windows: `-Dorg.osgi.framework.bootdelegation=javax.xml.* -XX:MaxPermSize=256m -XX:+UnlockDiagnosticVMOptions -XX:+UnsyncloadClass -XX:+UseParNewGC -Xms512m -Xmx768m`
 -  Linux: `-Dorg.osgi.framework.bootdelegation=javax.xml.* -XX:MaxPermSize=256m -XX:+UnlockDiagnosticVMOptions -XX:+UnsyncloadClass -XX:+UseParNewGC -Xms512m -Xmx768m`
 -  Mac OS: `-XstartOnFirstThread -Dorg.osgi.framework.bootdelegation=javax.xml.* -XX:MaxPermSize=512m -XX:+UnlockDiagnosticVMOptions -XX:+UnsyncloadClass -XX:+UseParNewGC -Xms512m -Xmx768m`
4. Click **Run** to launch a child TIBCO Business Studio configured with the bw-runtime platform.
5. In the child TIBCO Business Studio, click **File > Import** to import the model and runtime bundles and features:
 - a) In the Select dialog, expand the **General** folder and select **Existing Projects into Workspace**. Click **Next**.
 - b) In the Import Projects dialog, 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.



6. 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.

7. 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 Java Global Instance Shared Resource

You can use Java Global Instance shared configuration resource to specify a Java object to be shared across all process instances in a Java Virtual Machine (JVM).

When the engine is started, an instance of the specified Java class is constructed. When the process engine shuts down, if specified, a cleanup method is invoked on the object. The object is released before the engine shuts down. If multiple process instances access the shared Java global instance, you might want to ensure that only one process instance can access the object at a time. You can accomplish this by either declaring the methods of the configured class as synchronous or by using a critical section group.

Prerequisites

Ensure that you have created a palette, as described in [Defining a Palette](#).

Procedure

1. [Adding Java Global Instance Shared Resource](#)
2. [Adding Business Logic](#)
3. [Creating a Process](#)
4. [Configuring Java Global Instance Shared Resource](#)


Adding Java Global Instance Shared Resource

You can add Java Global Instance shared resource for an activity.

Prerequisites

Ensure that you have created a palette, as described in [Defining a Palette](#).

Procedure

1. In the Define Activities, Process Starters and Signal-In dialog, click  to add an attribute for the activity that you created in [Adding and Configuring Activities](#).
2. Configure the attribute as follows:
 - Enter the name of the attribute in the **Name** field. For example, JavaGlobal.
 - Select **String** from the **Type** list.
 - Enter the displayed name of the attribute in the **Label** field. For example, Java Global Instance.
 - Select **PropertyField** from the **Control Type** list.
 - Select **General** or **Advanced** from the **Section ID** list, or enter any customized section name.
 - Select or clear the **Required Field** check box to set whether the attribute is required or not.
3. From the **Resource schema type** list, select a way to generate the input, output, and fault schema:
 - If you select **XSD/WSDL**, see [Creating Schema with XSD/WSDL](#).
 - If you select **Java Code**, BusinessWorks Plug-in Development Kit automatically generates a sample code of input and output.
 - If you select **XSD Editor**, see [Creating Schema with XSD Editor](#).

- If you select **None**, no input and output are generated. BusinessWorks Plug-in Development Kit generates a fault schema.
4. Click **Apply** to save your configurations.
You are now directed to the Define Activities, Process Starters and Signal-In dialog.
 5. Click **Finish** to generate the activity with Java global instance shared resource in the palette.
 6. 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, click **Default** to choose the running platform. Click **Next**.
 - d) In the Target Content dialog, click **Finish**.
 - e) You are back to the Target Platform dialog, 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.
 7. Close the runtime bundle and feature, and open the design bundle and feature.

Result

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

What to do next

[Adding Business Logic](#)

Adding Business Logic

You can add your own logic for each activity.

Prerequisites

Ensure that you have added Java global instance shared resource, as described in [Adding Java Global Instance Shared Resource](#).

Procedure

- Complete the steps in [Adding Business Logic](#).

Java global instance shared resource is added to the runtime classes of your created activity.

```
/**
 *
 *
 *
 * @generated
 */
@property(name = "JavaGlobalInstanceName")
public JavaGlobalInstanceResource javaGlobalInstanceResource;
```

You can also invoke methods to get the classes information that you configured.

```

public N execute(N input, ProcessContext<N> processContext) throws ActivityFault {
    if (getActivityLogger().isDebugEnabled()) {
        activityLogger.debug(RuntimeMessageBundle.DEBUG_PLUGIN_ACTIVITY_METHOD_CALLED
            , new Object[] { "execute()"
                , activityContext.getActivityName()
                , activityContext.getProcessName()
                , activityContext.getDeploymentUnitName()
                , activityContext.getDeploymentUnitVersion() });
        String serializedInputNode = XMLUtils.serializeNode(input, processContext.getXMLProcessingContext());
        activityLogger.debug(RuntimeMessageBundle.DEBUG_PLUGIN_ACTIVITY_INPUT, new Object[] { activityContext.getActivityName(), serializedInputNode });
    }
    N result = null;
    try {
        // begin-custom-code
        // add your own business code here
        javaGlobalInstanceResource.
        // end-custom-code

        // create output data according to input
        result = evalOutput(input,
    } catch (Exception e) {
        throw new ActivityFault(activityContext.getActivityName(), e);
    }
    if (getActivityLogger().isDebugEnabled()) {
        String serializedOutputNode = XMLUtils.serializeNode(result, processContext.getXMLProcessingContext());
        activityLogger.debug(RuntimeMessageBundle.DEBUG_PLUGIN_ACTIVITY_OUTPUT, new Object[] { activityContext.getActivityName(), serializedOutputNode });
    }
    return result;
}
/**
 * <!-- begin-custom-doc -->
 *
 *
 */

```

What to do next

Creating a Process


Creating a Process

You can create a process to include the activity that contains the created Java global instance shared resource.

Prerequisites


Ensure that you have created a business logic, as described in [Adding Business Logic](#).

Procedure

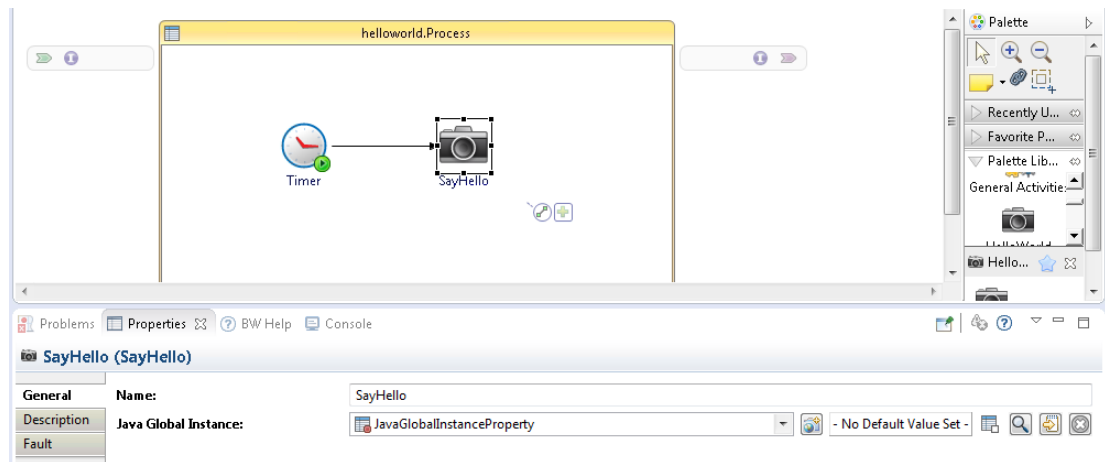
1. In the child TIBCO Business Studio, click **File > New > BusinessWorks Resources** from the menu.
2. In the "Select a wizard" dialog, click **BusinessWorks Application Module**. Click **Next**.
3. In the Project dialog, enter a value in the **Project name** field, and select the **Use Java configuration** check box. Click **Finish** to create a project.
4. In the created process editor, find the General Activities palette, select and drop a Timer activity to the process editor.
5. From the palette that you created in [Defining a Palette](#), select and drop the activity that you created in [Adding and Configuring Activities](#) to the process editor.
6. Drag the  icon to create a transition between the Timer activity and your created activity.

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



The  icon is displayed only when you select a Timer activity.

7. In the process editor, double-click your created activity. Click the **General** tab, you can see the Java global instance shared resource you just added in [Adding Java Global Instance Shared Resource](#).



What to do next

Configuring Java Global Instance Shared Resource


Configuring Java Global Instance Shared Resource

You can configure Java global instance shared resource for an activity according to your own environment.

Prerequisites

Ensure that you have created a process to include the activity that contains the created Java global instance shared resource, as described in [Creating a Process](#).

Procedure

1. In the **General** tab of your created activity, click  to create or choose a default resource.
 - a) In the Select JavaGlobalInstanceResource Resource Template dialog, click **Create Shared Resource**.
 - b) In the Java Global Instance Resource dialog, review the detailed information, and click **Finish**.
2. In the Project Explorer view, expand *project name*, right-click the **src** folder, and select **New > Class**.



You have to select the **Use Java configuration** check box when you create the project, and then the **src** folder can be created.

3. In the Java Class dialog, specify values in the **Package** and **Name** fields. Click **Finish**.
4. In the new .java file, add your own code logic, and save the file.

This is an example:

```
package com.test;

public class Person {
    private String name;
    private String age;

    public Person(String name, String age) {
        this.name = name;
        this.age = age;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getName() {
```

```

        return name;
    }

    public void setAge(String age) {
        this.age = age;
    }

    public String getAge() {
        return age;
    }
}

```

5. In the Project Explorer view, expand *project name* > **Resources** > *project name* > **JavaGlobalInstanceResource.javaGlobalInstanceResource**.
6. In the JavaGlobalInstanceResource.javaGlobalInstanceResource file, find the **Java Global Instance** tab.
 - a) Click **Browse** to find the .java file that created in [Step 4](#). In the Class Selection dialog, enter the .java file name to find the created file. Select the file in the **Matching items** field, and click **OK**.
 - b) From the **Method** list, select the constructor of the class.
 - c) In the **Parameter Input** area, specify a value in the **Value** field for the parameter. For example, default.
7. In the **Advanced Configuration** tab, select the **Invoke Cleanup Method** check box, and select clean() from the **Cleanup Method** list. Save your configurations.
8. In the Project Explorer view, expand **com.company_name.bw.palette.palette_name.runtime** > **src** > **com.company_name.bw.palette.palette_name.runtime** > *activity_nameactivity_type.java*.
9. Double-click the .java file, you can see the code changes of the created Java global instance shared resource.

```

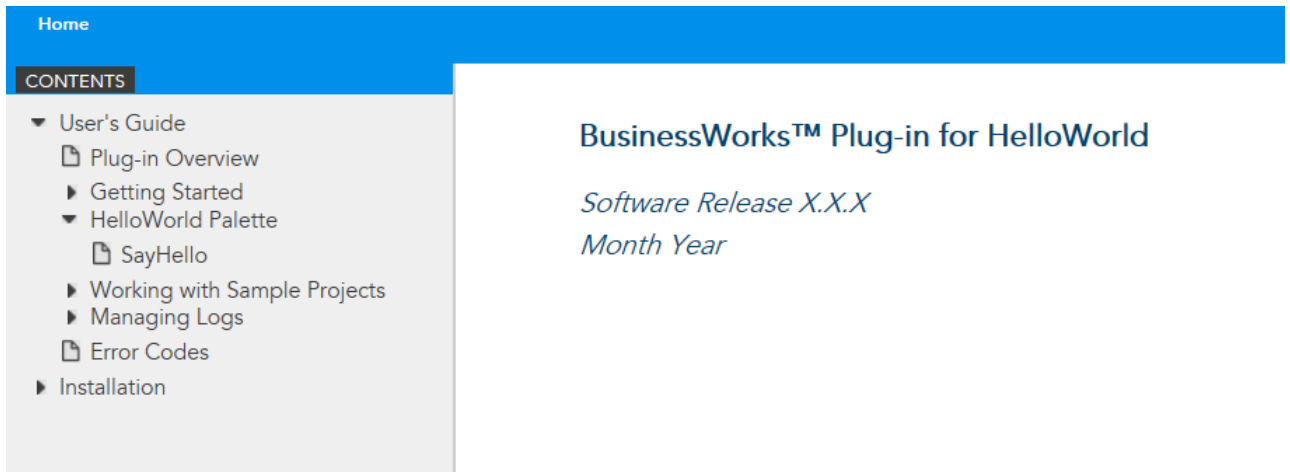
/**
 *
 *
 * @generated
 */
@property(name = "JavaGlobalInstanceName")
public JavaGlobalInstanceResource javaGlobalInstanceResource;

```

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.



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.

```

8      "static/print.css?xKpAPhd3" media="print" rel="stylesheet" />
9
10     <script src="static/head.js?qUF721F0"></script>
11     <!-- saved from url=(0014)about:internet -->
12     <script src="/javascripts/implementations.js"></script>
13     </head>
14     <body id="GUID-7eb4f705-3f4a-4934-a2ca-ff25633d6f13" data-approot=""><header><a href=
15     "index.html" id="logo-link">
16     </a>
17     </header>
18     <section id="center"><article><a name="GUID-7eb4f705-3f4a-4934-a2ca-ff25633d6f13"></a>
19     <h1 class="topicTitle1">SayHello</h1>
20     <div><p> The SayHello Activity always publishes a HelloWorld string no matter what you provide
21     as the input.
22     </p>
23     <div class="section" id=
24     "GUID-7eb4f705-3f4a-4934-a2ca-ff25633d6f13__GUID-836BCE98-5FB6-49B8-9867-962967B1659F"><a
25     name="GUID-7eb4f705-3f4a-4934-a2ca-ff25633d6f13__GUID-836BCE98-5FB6-49B8-9867-962967B1659F"
26     ></a>
27     </div>
28     <div class="section" id=
29     "GUID-7eb4f705-3f4a-4934-a2ca-ff25633d6f13__GUID-46D94B38-C150-4D1A-BEB3-EA4BA52710D7"><a
30     name="GUID-7eb4f705-3f4a-4934-a2ca-ff25633d6f13__GUID-46D94B38-C150-4D1A-BEB3-EA4BA52710D7"
31     ></a><h2 class="sectionTitle">General</h2>
32     <p>The
33     <span class="uiControl">General</span> tab contains the following fields.
34     </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.

The screenshot displays the TIBCO ActiveMatrix BusinessWorks IDE interface. The top pane shows a process diagram titled "test.Process" with a "Timer" activity connected to a "SayHello" activity. The bottom pane is divided into two sections: a "CONTENTS" sidebar and a main documentation area for the "SayHello" activity.

CONTENTS

- ▼ User's Guide
 - Plug-in Overview
 - ▶ Getting Started
 - ▼ HelloWorld Palette
 - SayHello**
 - ▶ Working with Sample Projects
 - ▶ Managing Logs
 - ▶ Error Codes
 - ▶ Installation

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.

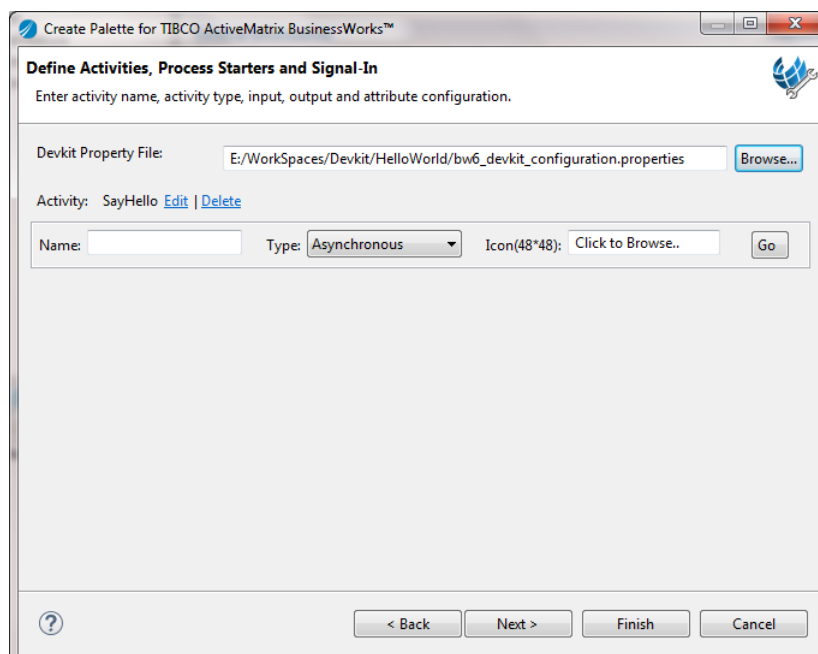
Before editing a plug-in, see [Merging Code](#) for more details about how BusinessWorks Plug-in Development Kit merges the code.

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 Existing BW6 Palette**.
4. In the Define Activities, Process Starters and Signal-In dialog, click **Browse** next to the **Devkit Property 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.



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.



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.

It is not 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	<input type="checkbox"/>	<input type="checkbox"/>

Resource schema type: ? XSD Editor

Create input, output or fault schema using following options.

[SayHelloInput](#) ✖ [SayHelloOutput](#) ✖ [Fault](#)

[Apply](#) [Cancel](#)

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:

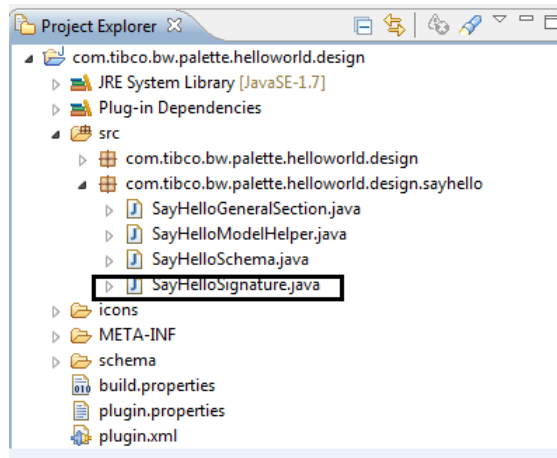
- If you use an XSD file or a WSDL file to define the input, output, and fault schema, you can update the XSD file or the WSDL file accordingly and regenerates the palette.
- If you use the default input, output, and fault schema generated by BusinessWorks Plug-in Development Kit, you can use the `[ActivityName]Signature`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

In all, no matter you use which method to define the schema, you can use the `[ActivityName]Signature` class that is located in the `[ActivityName]Signature.java` file to update the activity schema.

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.






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);

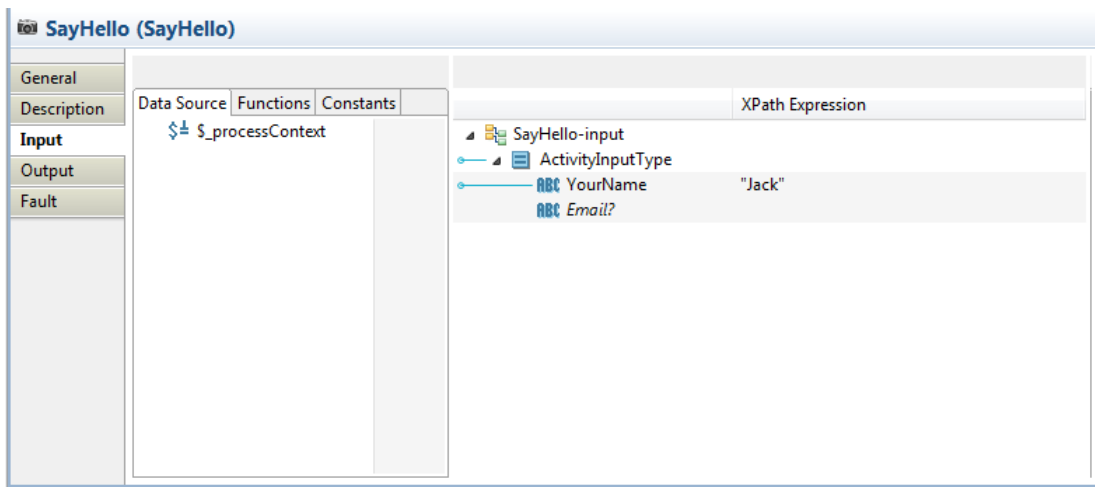
    XSDModelGroup rootInput = XSDUtility.addComplexTypeElement(inputSchema,
"ActivityInputType", "ActivityInputType", XSDCompositor.SEQUENCE_LITERAL);
    XSDUtility.addSimpleTypeElement(rootInput, "YourName", "string", 1, 1);
    XSDUtility.addSimpleTypeElement(rootInput, "Email", "string", 0, 1);

    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 the (x)= **Arguments** tab, and then enter the following parameters in the **VM arguments** area. Click **Apply**.
 -  Microsoft Windows: `-Dorg.osgi.framework.bootdelegation=javax.xml.* -XX:MaxPermSize=256m -XX:+UnlockDiagnosticVMOptions -XX:+UnsyncloadClass -XX:+UseParNewGC -Xms512m -Xmx768m`
 -  Linux: `-Dorg.osgi.framework.bootdelegation=javax.xml.* -XX:MaxPermSize=256m -XX:+UnlockDiagnosticVMOptions -XX:+UnsyncloadClass -XX:+UseParNewGC -Xms512m -Xmx768m`
 -  Mac OS: `-XstartOnFirstThread -Dorg.osgi.framework.bootdelegation=javax.xml.* -XX:MaxPermSize=512m -XX:+UnlockDiagnosticVMOptions -XX:+UnsyncloadClass -XX:+UseParNewGC -Xms512m -Xmx768m`
6. Click **Run** 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, 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 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, if 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.

Exporting Features

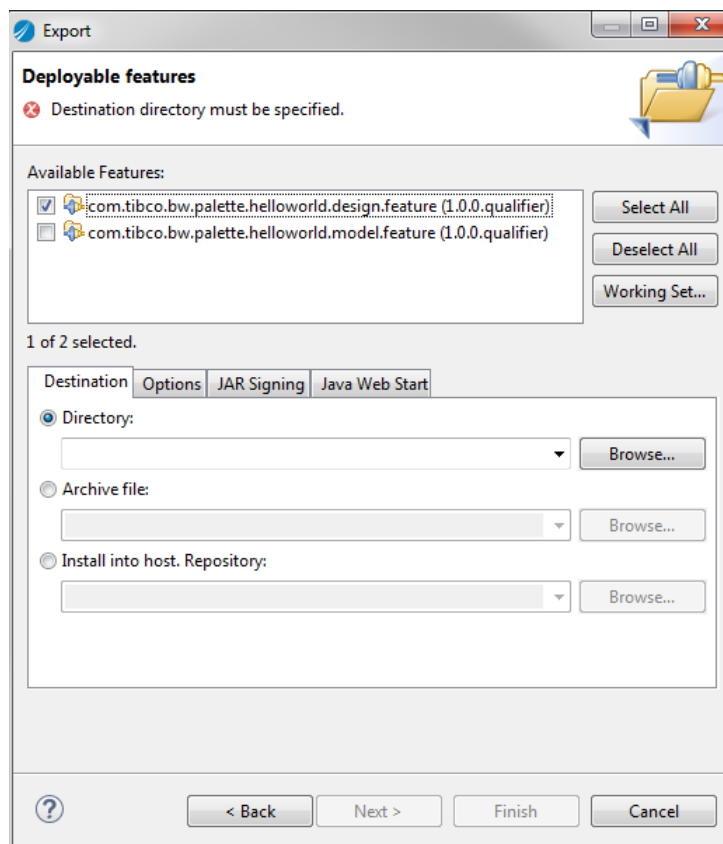
Before packaging a plug-in, you have to export the design, runtime, and model features one by one.

Prerequisites

Ensure that you have created a plug-in as described in [Creating a Plug-in](#).

Procedure

1. In TIBCO Business Studio with the running platform selected, click **File > Export** from the menu.
2. In the Select dialog, click the **Plug-in Development** folder and select **Deployable features**. Click **Next**.
3. In the "Deployable features" dialog, select the design-time feature in the **Available Features** area.



4. In the **Destination** tab, click **Browse** next to the **Directory** field and locate the folder where the design-time feature is exported. Click **Finish**.
An `artifacts.jar` file, a `content.jar` file, a `features` folder, and a `plugins` folder are exported to the specified location.
5. Repeat [Step 1](#) to [Step 4](#) to export the model and runtime features one by one.



Ensure that you have exported the design, model, and runtime features of the plug-in to three different folders one by one.

Ensure that the target platform is bw-runtime when exporting the runtime feature. See [Target Platform](#) for more details.


Generating an Installer

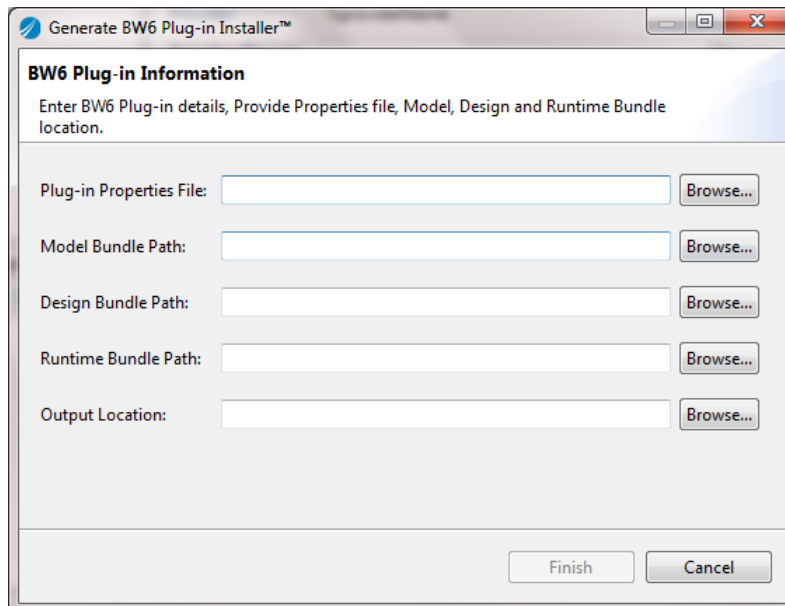
BusinessWorks Plug-in Development Kit can package a created plug-in as a TIBCO Eclipse Plug-in.

Prerequisites

Ensure that you have exported required bundles as described in [Exporting Features](#).

Procedure

1. Choose one of the following ways to open the installer generation wizard:
 - From the menu, click **Help > Create BusinessWorks Plug-in Installer**.
 - On the toolbar, click **Create BusinessWorks Plug-in Installer** .
 - Press Ctrl+6.
2. In the BW6 Plug-in Information dialog, provide the following information:



- **Plug-in Properties File**

Click **Browse** to select the `bw6-devkit_configuration.properties` file that contains the project information of the plug-in.

The configuration file is generated when generating your plug-in code for the first time. You can find this configuration file in your project folder that you specified in BusinessWorks Plug-in Development Kit wizard.

- **Model Bundle Path**

Click **Browse** to select the folder that contains the model feature exported in the [Exporting Features](#) section.

- **Design Bundle Path**

Click **Browse** to select the folder that contains the design-time feature exported in the [Exporting Features](#) section.

- **Runtime Bundle Path**

Click **Browse** to select the folder that contains the runtime feature exported in the [Exporting Features](#) section.

- **Output Location**

Click **Browse** to select the folder where the generated plug-in installer is saved.

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

Result

The following two folders are generated in the output folder:

- The `PaletteName-P2Installer` folder contains the JAR files and plug-in features.
You can use this installer to install the created palette in TIBCO Business Studio by using Eclipse Update Manager.
- The `PaletteName-RuntimeInstaller` folder contains the `devkitpackager.jar` file.
You can use this `devkitpackager.jar` file to install only the runtime component by using the 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

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.

After generating the p2 installer for a created plug-in as described in [Creating an Installer for a Plug-in](#), the following folders are created in the installer folder:

- The `PaletteName-P2Installer` folder contains the JAR files and plug-in features.



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

See [Installing a Created Plug-in](#) for more details about how to install the created plug-in, and see [Uninstalling a Created Plug-in](#) for more details about how to uninstall a plug-in.


- The `PaletteName-RuntimeInstaller` folder contains the `devkitpackager.jar` file.

You can use this `devkitpackager.jar` file to install only the runtime component by using the command line.

To install or uninstall the runtime component of a plug-in:

-  Microsoft Windows and  Mac OS:
 1. Open the command line and switch to the `PaletteName-RuntimeInstaller` directory where the `devkitpackager.jar` file is located.
 2. Enter the following command to install the runtime component:


```
java -jar devkitpackager.jar -ri install -th tibcohome -i installerfolder
```
 3. Enter the following command to uninstall the runtime component:


```
java -jar devkitpackager.jar -ri uninstall -th tibcohome -p palettename -v version_number
```
-  Linux:
 1. Open the command line and switch to the `TIBCO_HOME/tibcojre64/1.7.0/bin` directory where TIBCO Java is installed.
 2. Enter the following command to install the runtime component:


```
./java -jar runtimeinstallerfolder/devkitpackager.jar -ri install -th tibcohome -i installerfolder
```
 3. Enter the following command to uninstall the runtime component:


```
./java -jar runtimeinstallerfolder/devkitpackager.jar -ri uninstall -th tibcohome -p palettename -v version_number
```

where

- `-th tibcohome` is the directory where TIBCO ActiveMatrix BusinessWorks is installed. For example, `c:\tibco_bw6`.
- `-i installerfolder` is the directory that contains the generated p2 installer.
- `-p palettename` is the palette that you want to uninstall.
- `-v version_number` is the version of the plug-in that you want to uninstall.
- `runtimeinstallerfolder` is the directory that contains the `devkitpackager.jar` file.




Installing a Created Plug-in

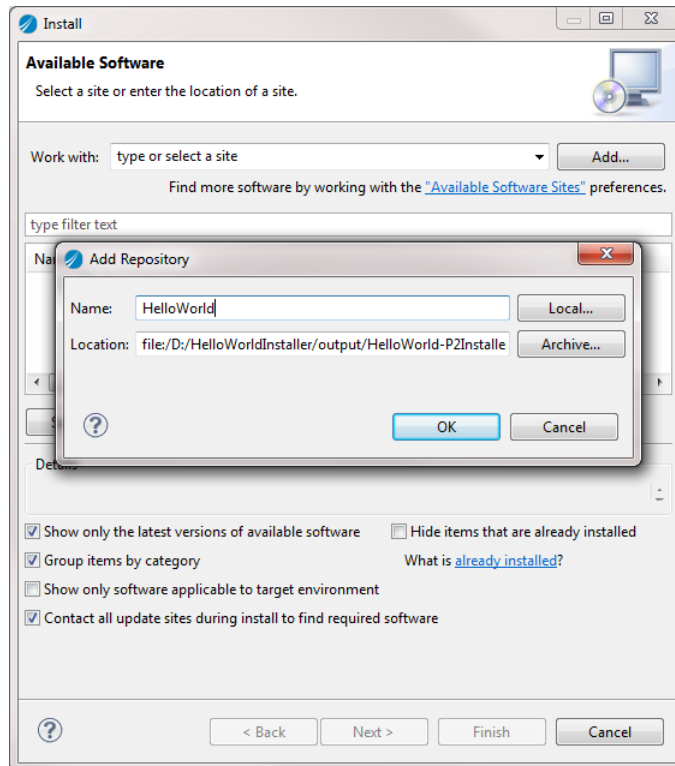
You can use Eclipse Update Manager to install a created BusinessWorks 6 plug-in in TIBCO Business Studio.

Prerequisites

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, click **Add**.
4. In the Add Repository dialog, click **Local** and select the p2 installer folder that you have generated. Click **OK**.






5. Select the created plug-in that you want to install. Click **Next**.
6. In the Install Details dialog, review the components to be installed. Click **Next**.
7. In the Review Licenses dialog, click **I accept the terms of the license agreement**. Click **Finish** to install the plug-in.
8. During the installation, a security warning dialog 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 > Install New Software** to open the Eclipse Update Manager.
3. In the Available Software dialog, click the **already installed** link.
4. In the **Installed Software** tab, click the plug-in that you want to uninstall, and then click **Uninstall**.

5. In the Uninstall Details dialog, review the product to be uninstalled. Click **Finish** to uninstall the selected plug-in.
6. Click **Yes** when you are prompted to restart TIBCO Business Studio.

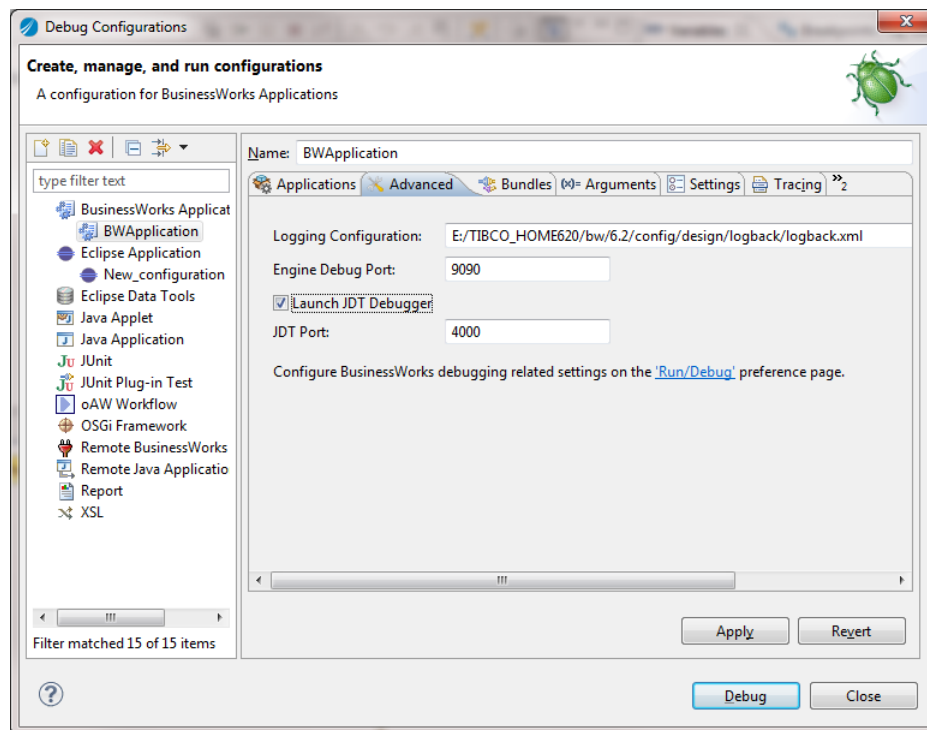
Running the Plug-in

After installing the plug-in, you can add your plug-in activities to a BusinessWorks process to complete a task.

A process captures and describes the flow of business information in an enterprise between different data sources and destinations. TIBCO ActiveMatrix BusinessWorks provides several methods to create processes.

To debug the runtime code when running your created plug-in, you have to select the **Launch JDT Debugger** check box in the **Advanced** tab under **BusinessWorks Application > BWApplication** in the "Create, manage, and run configurations" dialog.

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



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, 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.
 - 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, click **Default** to choose the running platform. Click **Next**.
 - d) In the Target Content dialog, click **Finish**.
 - e) You are back to the Target Platform dialog, 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, double-click **Eclipse Application** in the left panel to create a new Eclipse application.
7. Click the **(x)= Arguments** tab, and then enter the following parameters in the **VM arguments** field. Click **Apply**.
 -  Microsoft Windows: `-Dorg.osgi.framework.bootdelegation=javax.xml.* -XX:MaxPermSize=256m -XX:+UnlockDiagnosticVMOptions -XX:+UnsyncloadClass -XX:+UseParNewGC -Xms512m -Xmx768m`
 -  Linux: `-Dorg.osgi.framework.bootdelegation=javax.xml.* -XX:MaxPermSize=256m -XX:+UnlockDiagnosticVMOptions -XX:+UnsyncloadClass -XX:+UseParNewGC -Xms512m -Xmx768m`
 -  Mac OS: `-XstartOnFirstThread -Dorg.osgi.framework.bootdelegation=javax.xml.* -XX:MaxPermSize=512m -XX:+UnlockDiagnosticVMOptions -XX:+UnsyncloadClass -XX:+UseParNewGC -Xms512m -Xmx768m`
8. Click **Run** to launch a child TIBCO Business Studio configured with the bw-runtime platform.
9. In the child TIBCO Business Studio, click **File > Import** to import the model and runtime bundles and features:
 - a) In the Select dialog, expand the **General** folder and select **Existing Projects into Workspace**. Click **Next**.
 - b) In the Import Projects dialog, 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.

Prerequisites

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, 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, 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.

Prerequisites

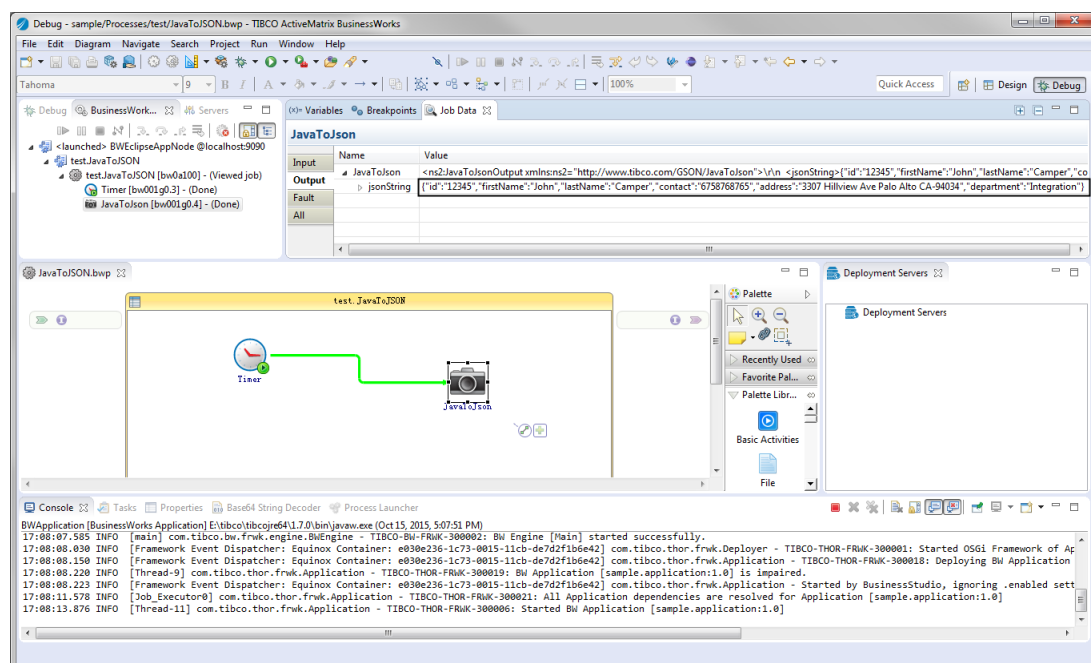
Ensure that you have imported the process, as described in [Importing the JavaToJSON Process](#).

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. See [Configuring Java Global Instance Shared Resource](#) for more details.
 - 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, 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.



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, 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, click **Default** to choose the running platform. Click **Next**.
 - d) In the Target Content dialog, click **Finish**.
 - e) You are back to the Target Platform dialog, 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, double-click **Eclipse Application** in the left panel to create a new Eclipse application.
7. Click the **(x)= Arguments** tab, and then enter the following parameters in the **VM arguments** field. Click **Apply**.
 -  Microsoft Windows: `-Dorg.osgi.framework.bootdelegation=javax.xml.* -XX:MaxPermSize=256m -XX:+UnlockDiagnosticVMOptions -XX:+UnsyncloadClass -XX:+UseParNewGC -Xms512m -Xmx768m`

-  Linux: `-Dorg.osgi.framework.bootdelegation=javax.xml.* -XX:MaxPermSize=256m -XX:+UnlockDiagnosticVMOptions -XX:+UnsyncloadClass -XX:+UseParNewGC -Xms512m -Xmx768m`
 -  Mac OS: `-XstartOnFirstThread -Dorg.osgi.framework.bootdelegation=javax.xml.* -XX:MaxPermSize=512m -XX:+UnlockDiagnosticVMOptions -XX:+UnsyncloadClass -XX:+UseParNewGC -Xms512m -Xmx768m`
8. Click **Run** to launch a child TIBCO Business Studio configured with the bw-runtime platform.
 9. In the child TIBCO Business Studio, click **File > Import** to import the model and runtime bundles and features:
 - a) In the Select dialog, expand the **General** folder and select **Existing Projects into Workspace**. Click **Next**.
 - b) In the Import Projects dialog, 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.

Prerequisites

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, 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, 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.

Prerequisites

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.

Prerequisites

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

- 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.

Prerequisites

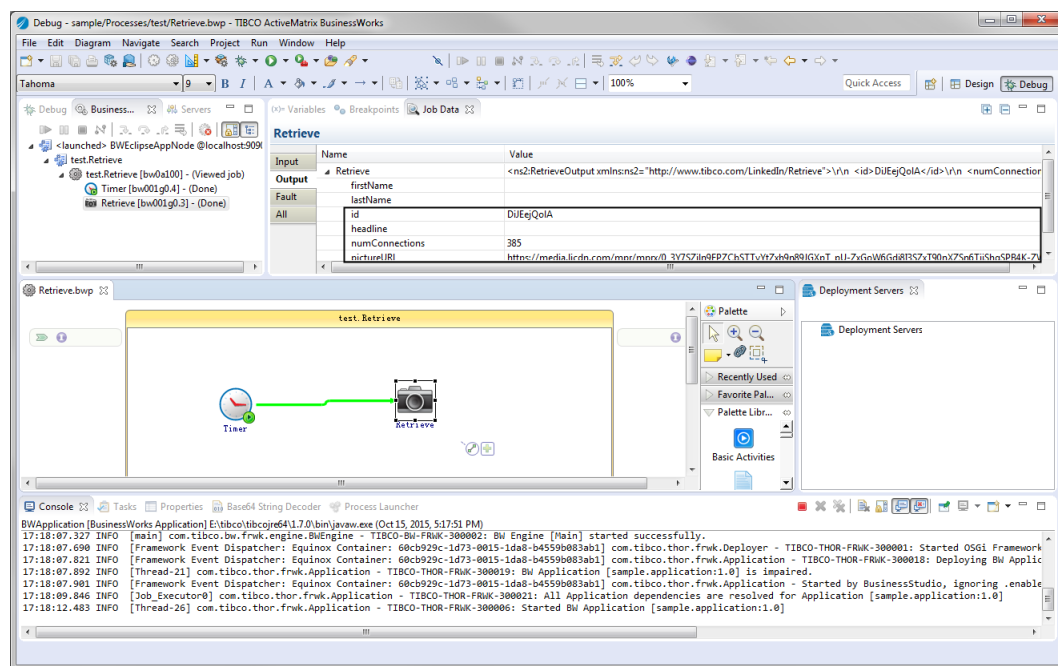
Ensure that you have imported the process, as described in [Importing the LinkedIn Processes](#).

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, 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.




Running the RetrieveDefaultProfile Process

Run the RetrieveDefaultProfile process to get the default content.

Prerequisites

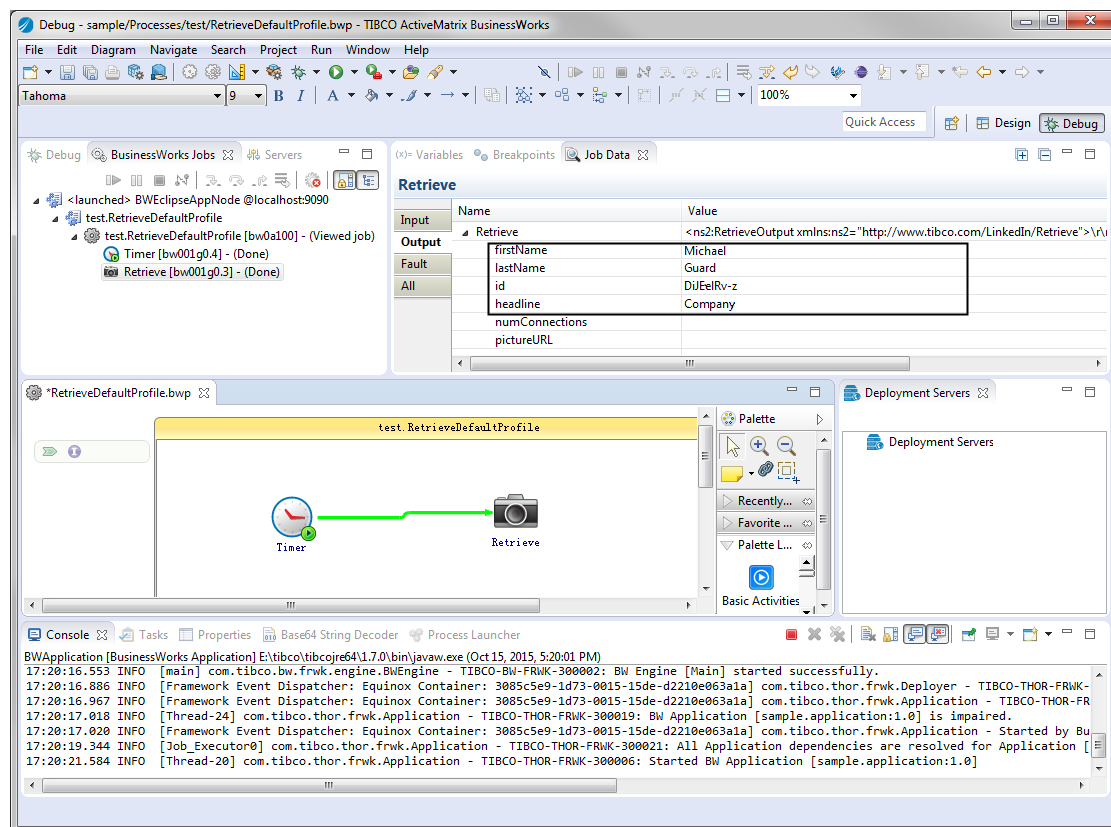
Ensure that you have imported the process, as described in [Importing the LinkedIn Processes](#).

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, 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.




Running the Update Process

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

Prerequisites

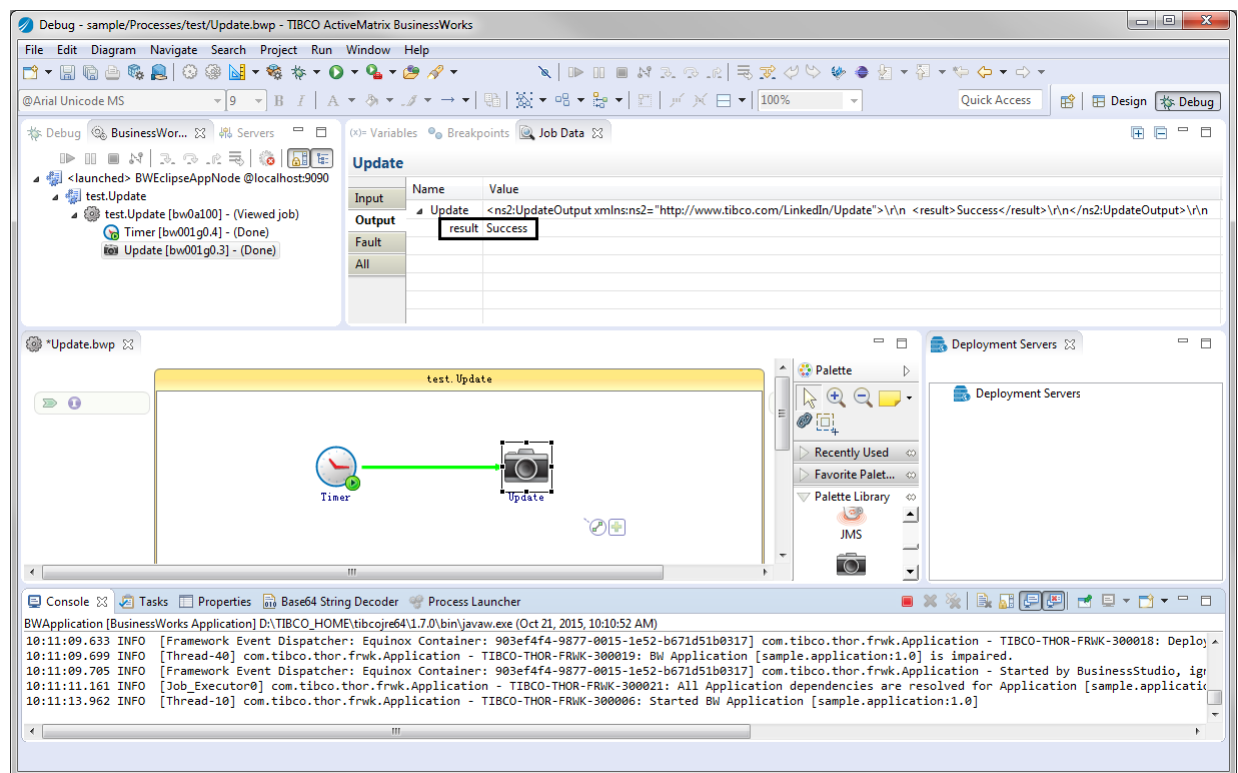
Ensure that you have imported the process, as described in [Importing the LinkedIn Processes](#).

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, 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.



Managing Logs for a Created Plug-in

TIBCO ActiveMatrix BusinessWorks provides the functionality to set up a log level to trace and troubleshoot plug-in exceptions.

A `logback.xml` file is located in the `TIBCO_HOME/bw/version_number/config/design/logback` directory. Update this file to [set up a log level](#) and [export logs to a file](#).

Log Levels

The plug-in captures logs at different levels.

Log Level	Description
Trace	Includes all information regarding the running process.
Debug	Includes all information regarding the running process.
Info	Indicates normal plug-in operations. No action is required. A tracing message tagged with Info indicates that a significant processing step is reached, and logged for tracking or auditing purposes. Only information messages preceding a tracking identifier are considered as significant steps.
Warn	Indicates that an abnormal condition occurred. Processing continues, but special attention from an administrator is required.
Error	Indicates that an unrecoverable error occurred. Depending on the severity of the error, the plug-in might continue with the next operation or might stop.

Setting Up a Log Level

By default, the log level is Error. You can change the log level to trace different messages.



If neither the plug-in log nor the BusinessWorks log is configured in the `logback.xml` file, the error logs of the plug-in are displayed in the Console view by default.

If the plug-in log is not configured but the BusinessWorks log is configured in the `logback.xml` file, the configuration for the BusinessWorks log is implemented by the plug-in.

Procedure

1. Navigate to the `TIBCO_HOME/bw/version_number/config/design/logback` directory and open the `logback.xml` file.

2. Add the following node in the User loggers area to specify the log level for a created plug-in:

```
<logger name="com.tibco.bw.palette.palette_name.runtime">
  <level value="DEBUG"/>
</logger>
```



When the `level` is set to Debug, the input and output for the plug-in activities are also displayed in the Console view. See [Log Levels](#) for more details regarding each log level.

3. Optional: Add the following node in the User loggers area to specify the log level for an activity:

```
<logger
name="com.tibco.bw.palette.palette_name.runtime.ActivityName_ActivityType_Activity">
  <level value="DEBUG"/>
</logger>
```



If the activity that you want to add a log level for is an event source activity, add the following node:

```
<logger
name="com.tibco.bw.palette.palette_name.runtime.ActivityName_ActivityType">
  <level value="DEBUG"/>
</logger>
```

For example, if you want to set the log level of the SayHello activity to Debug, add the following node:

```
<logger
name="com.tibco.bw.palette.HelloWorld.runtime.SayHelloAsynchronousActivity">
  <level value="DEBUG"/>
</logger>
```



If you do not configure a specific log level for an activity, the activity applies the log levels that you configured for the plug-in.

4. Save the file.

Exporting Logs to a File

Modify the `logback.xml` file to export plug-in logs to a file.

Procedure

1. Navigate to the `TIBCO_HOME/bw/version_number/config/design/logback` directory and open the `logback.xml` file.



After deploying an application in TIBCO Enterprise Administrator, navigate to the `TIBCO_HOME/bw/version_number/domains/domain_name/appnodes/space_name/node_name` directory to find the `logback.xml` file.

2. Add the following node to specify the file location.

```
<appender name="FILE" class="ch.qos.logback.core.FileAppender">
  <file>c:/bw6-helloworld.log</file>
  <encoder>
    <pattern>%d{HH:mm:ss.SSS} [%thread] %-5level %logger{36}-%msg%n</pattern>
  </encoder>
</appender>
```

The `file` tag defines the location to which the log is to be exported, and the value is the absolute path of the file.



You also have to add the file name in the file path.

3. Add the following node to the root node at the bottom of the `logback.xml` file to export the logs to a file.

```
<root level="DEBUG">
  <appender-ref ref="STDOUT" />
  <appender-ref ref="FILE" />
</root>
```

4. Save the file.

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_ActivityTypeActivity.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, click the **Plug-in Development** folder, and then click **Plug-in from Existing JAR Archives**. Click **Next**.
3. In the "JAR selection" dialog, 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.

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.

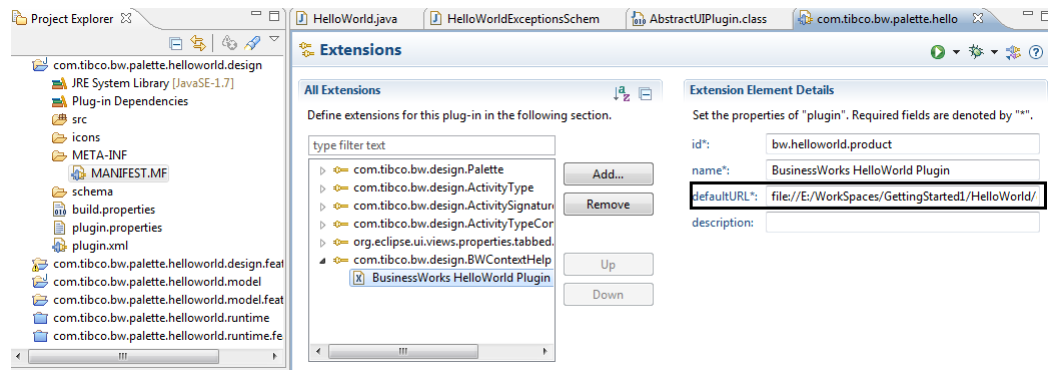
See [Creating Documentation](#) for more details about how to update the document template.

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.



Do not change **id** field.

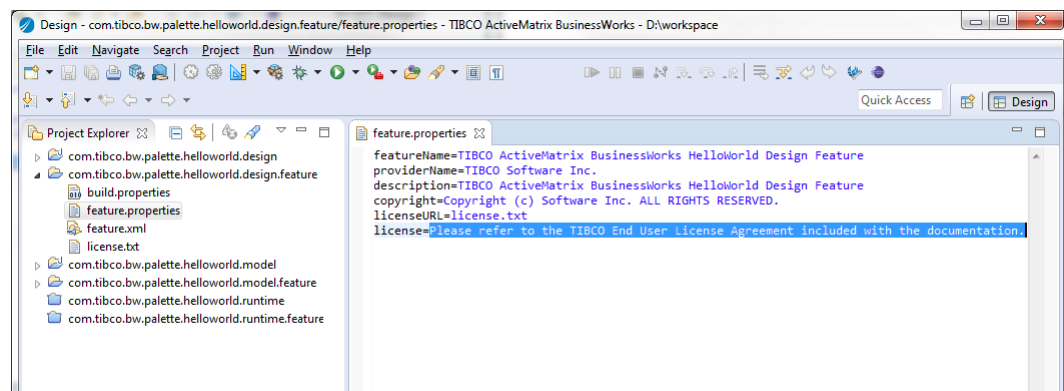


How to Add License

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.

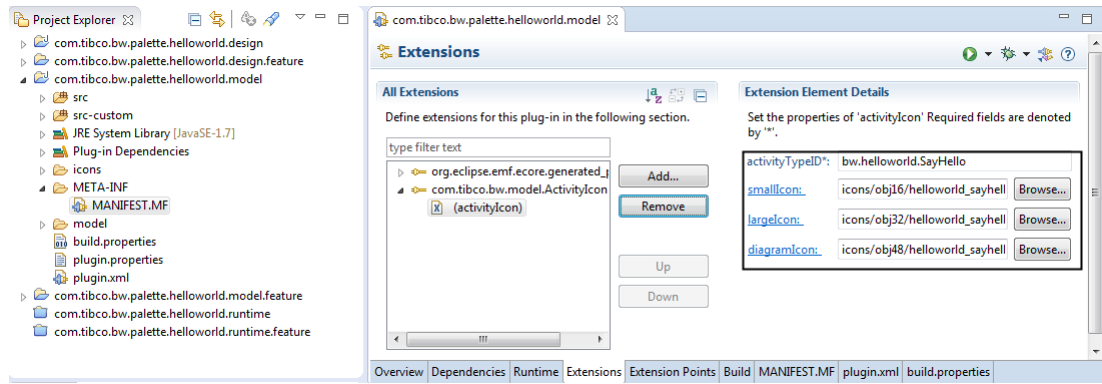


How to Add an Activity Icon

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.



How to Find BusinessWorks API JavaDoc

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

Procedure

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

Troubleshooting

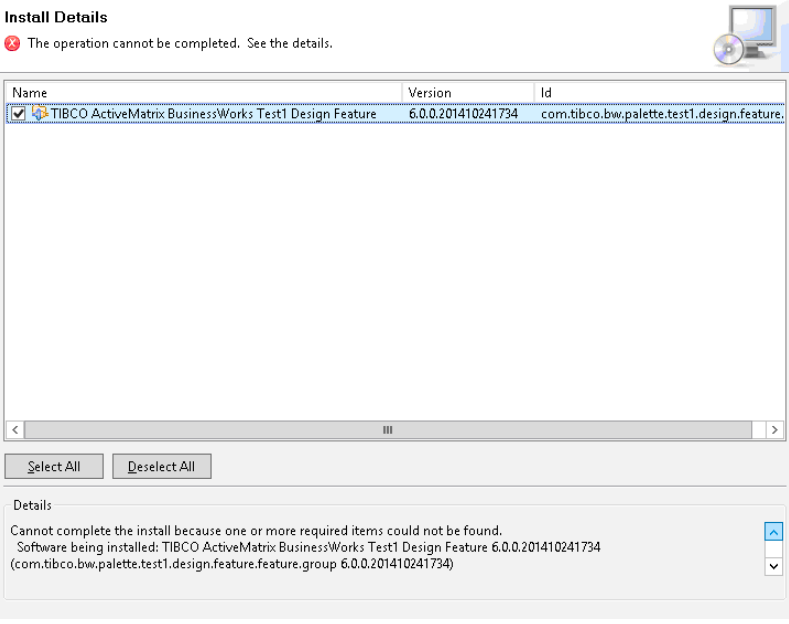
You might encounter compiling errors when creating a plug-in, or you might encounter problems when you migrate BusinessWork Plug-in Development Kit 6.0 to 6.1. You can go over the listed scenarios for troubleshooting.

- [General Problems](#)
- [Migration 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
You might encounter a file not found, or cannot find ecore file error when creating a plug-in.	Open a new workspace and create the plug-in again.
A refresh error occasionally occurs when generating the plug-in bundles because the project is not opened 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 is displayed when importing projects.	Close the error dialog.

Scenarios	Reason/Workaround
<p>When you do a repackage for a plug-in that you have created by using TIBCO ActiveMatrix BusinessWorks Plug-in Development Kit, if you import the design, runtime, and model features to the same folder that contains the features imported in the first package, and you also use the same output folder for the generated package.</p> <p>An error occurs when you install the plug-in with the second package in TIBCO Business Studio.</p>	<ol style="list-style-type: none"> 1. Delete the exported features. 2. Export the runtime, design-time, and model features to three folders one by one, and then regenerate the installer.
	
<p>Validation errors occur when running activities that contain the any element.</p>	<p>Add code manually.</p>

Scenarios	Reason/Workaround
After installing a generated plug-in, "TIBCO ActiveMatrix" is added at the beginning of the feature name.	<p>Edit the <code>featureName</code> property to change the feature name, the <code>providerName</code> property to change the provider name, and the <code>description</code> property to change the details in the <code>feature.properties</code> files in the following folders:</p> <ol style="list-style-type: none"> 1. Design feature: <pre>com.companyname.bw.palett e.palette_name.design.fea ture</pre> 2. Model feature: <pre>com.companyname.bw.palett e.palette_name.model.fea ture</pre> 3. Runtime feature: <pre>com.companyname.bw.palett e.palette_name.runtime.fe ature</pre>

Migration Problems

You might encounter problems when you migrate BusinessWork Plug-in Development Kit 6.0 based on TIBCO ActiveMatrix BusinessWorks 6.2.X to version 6.1 based on TIBCO ActiveMatrix BusinessWorks 6.3. You can go over the listed scenarios for troubleshooting.

Scenarios	Reason/Workaround
PDK version errors occur.	Change a new workspace, or delete the <code>.JETEmitters</code> folders in the old workspace.
The installer that was generated with BusinessWork Plug-in Development Kit 6.0 fails to be installed on TIBCO ActiveMatrix BusinessWorks 6.3.	Update the plug-in project dependency based on TIBCO ActiveMatrix BusinessWorks 6.3, and generate a new installer with BusinessWork Plug-in Development Kit 6.1.
When editing processes with the process starter and signal-in activities, code errors occur.	Comment out the <code>return isStarted</code> code.
The processes fail to run when the output and fault schema contain any element.	Add code manually.

Scenarios	Reason/Workaround
The design-time <code>com.companyname.amf.sca.model.validation</code> bundle is missing.	Add the <code>com.companyname.amf.sca.model.validati</code> on bundle in the MANIFEST.MF file, or editing the project with ActiveMatrix BusinessWorks Plug-in Development Kit 6.1.0.