TIBCO BusinessWorks™ Container Edition Application Development

Software Release 2.3.1
August 2017

Document Update: September 2017

Two-Second Advantage®

TIBCO®

Important Information

Contents

Figures

TIBCO Documentation

Documentation for this and other TIBCO products is available on the TIBCO Documentation site. This 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, 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.

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

- Concepts
- Installation
- Getting Started
- Application Development
- Bindings and Palettes Reference
- Samples
- Error Codes
- Migration
- Conversion
- REST Implementation
- Application Monitoring and Troubleshooting

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 TIBCO Community

TIBCO Community 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. TIBCO Community offers forums, blogs, and access to a variety of resources including product wikis that provide in-depth information, white papers, and video tutorials. In addition, users can submit and vote on feature requests via the Ideas portal. For a free registration, go to https://community.tibco.com.

Changing Help Preferences

By default, documentation access from TIBCO Business Studio™ Container Edition is online, through the TIBCO Product Documentation site (Doc site) at **https://docs.tibco.com/**which contains the latest version of the documentation. Check the Doc site frequently for updates. To access the product documentation offline, download the documentation to a local directory or an internal web server and then change the help preferences in TIBCO Business Studio™ Container Edition.

Prerequisites

Before changing the help preferences to access documentation locally or from an internal web server, download documentation from https://docs.tibco.com/.

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

To change help preferences on the Preferences dialog to access the documentation from a custom location:

Procedure

1. In TIBCO Business Studio™ Container Edition, click **Window** > **Preferences**. On Mac OS X, click **TIBCO Business Studio BusinessWorks Container Edition** > **Preferences**.
2. In the Preferences dialog, click **BusinessWorks** > **Help**.
3. Click **Custom Location** and then click **Browse** to select the `html` directory in the folder where you unzipped the documentation, or provide the URL to the `html` directory on your internal web server.
4. Click **Apply** and then click **OK**.

Application Design Considerations

Applications help solve integration problems of varying complexity. This section describes some important factors to consider when designing an application.

Choosing Between Integration Styles

The following table provides guidelines to choose a high-level integration style for your applications.

Salient features of integration styles

	Speed of Integration	Data Abstraction	Richness of Orchestration Primitives	Typical Endpoints
Batch-oriented	Non-real time	Record	Low	Databases, files, and so on
Application-oriented	Real-time	Message	Medium	Application APIs, Adapters, and so on
Service-oriented	Real-time	Service, Operation	High	Web services and APIs
Resource-oriented	Real-time	Resource	Medium	Mobile/Web Applications and APIs

In an application-oriented integration style, each operation in a process can be invoked by a call to the process. Invoking multiple operations requires multiple calls to the process, that are then executed sequentially.

A service-oriented style exposes multiple operations available in a process and each of the operations can be called directly. These operations are not related and can be executed independently.

Choose to create (or use) a Java module (or a Java OSGi bundle), if multiple calls from a process to other Java libraries are needed to compute the result. Java modules provide a high degree of customization. To use the enhanced Java development tooling such as source folders, JRE libraries, and so on, select the **Use Java Configuration** check box in TIBCO Business Studio™ Container Edition when creating an application module. Alternatively, create a module that contains existing Java code or custom code.

Differences between Process Modules and Java Modules

	Orchestration Capabilities	Visibility	Granularity	Examples
Process Modules	High	High visibility of process flow logic, services, and bindings.	Better suited for coarse-grained functionality that consists of more discrete functionality and process constructs.	Account opening, mortgage loan, and so on.

	Orchestration Capabilities	Visibility	Granularity	Examples
Java Modules	Low	Low	Better suited for fine-grained functionality that has a very specific function, and often requires very little or no process constructs.	Query flight status, update product description, and so on.

Process Design Considerations

In process-driven design, the business processes or integration flows are first realized and captured. Service contracts might be applicable in a process-centric application, especially for batch or EAI-type automation scenarios. This topic describes some important factors to be considered when using a process-driven approach.

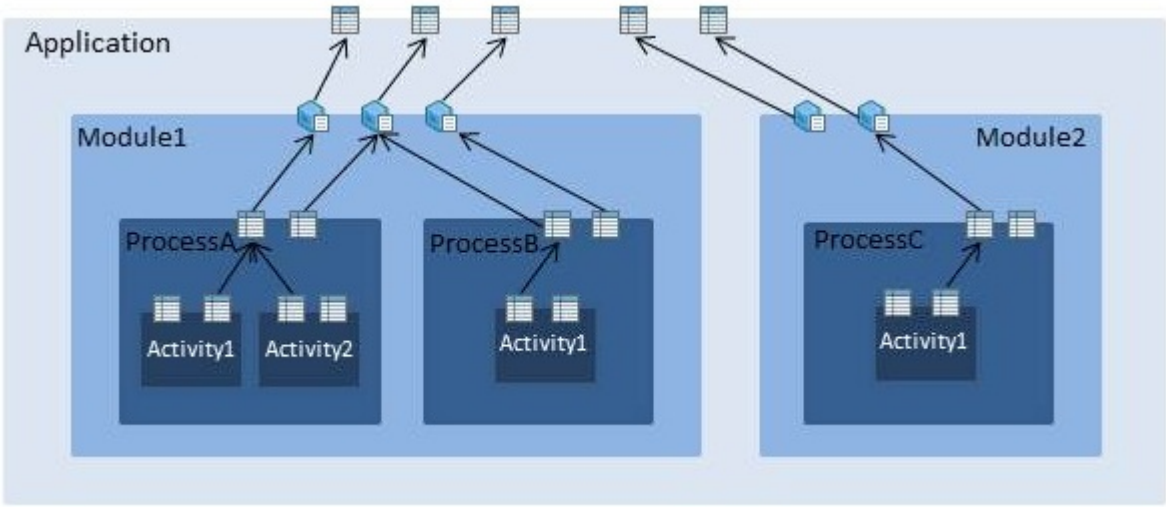
Choosing Between Properties and Variables

Properties are used to save configuration data at different levels. They can be classified into application properties, module properties, and process properties.

Variables are used to save the state at different levels. They can be classified into process variables, scope variables, and shared variables.

Choosing Between Process Properties, Module Properties, and Application Properties

Properties can be classified into application properties, module properties, and process properties. Properties follow the layered configuration model where configuration is pushed from top to the bottom as seen in the illustration:



Properties defined in the inner layer can reference a property defined at the parent layer. For example, a process property can reference a module property instead of providing a literal value. Public properties are visible to the encapsulating layers.

Choosing the right level ensures an easier to maintain list of properties in your application and keeps the number of properties at the application level to a minimum.

Comparing Process, Module, and Application Properties

	Scope/Visibility	Datatype	Values	Additional Information
Process Properties	Visible within a process.	Literal or shared resource reference.	Literal, shared resource reference, or a module property reference.	Literal values cannot be modified at the module or application level.

	Scope/Visibility	Datatype	Values	Additional Information
Module Properties	• Visible within the module. • Not visible or changeable from Administrator.	Literal or shared resource reference.	• Literal or a shared resource reference. • The DateTime module property must be specified in the format: `yyyy-MM-dd'T'HH:mm:ssXXX`. For example, `2001-07-04T12:08:56-07:00`	• Cannot be assigned to an activity directly. You need to reference a module property from a process property, and then reference the process property from the activity. • Any value for a private module property defined in the profile is ignored.
Application Properties	Displays all the module properties in the application. These properties are visible in Administrator.		• Literal. • Profiles can be used to provide a new set of values for the application.	• Overrides module properties, thus enabling you to use different values for the same module. • Cannot add new properties at application level.

Choosing Between Process Variables, Scope Variables, and Shared Variables

A process variable saves the state at the process level and a scope variable saves the state within the scope.

Variables defined within a scope are visible only within the scope. If the scope variable name is the same as a process variable name, then the scope variable takes precedence over the process variable within the scope.

Shared variables are used to save the state. There are two types of shared variables:

- **Module shared variable** - saves the state at a module level.
- **Job shared variable** - saves the state for the duration of a job.

For more information see *Shared Variables* in the *Concepts* guide

Handling Exceptions

Errors can occur when executing a process. The potential runtime errors in your process can be handled in one of the following ways:

- **Catch Specific**: Used to catch a specific kind of fault at either activity, scope, or process levels.
- **Catch All**: Used to catch all error or faults thrown at the selected level.

You can add an error transition to an activity or a group to specify the transition to take in case of an error.

Service Design Considerations

In service-driven design, the service contract or interface of each functional component is formalized first. The processing logic behind the service simply becomes an implementation detail that is encapsulated. This section describes some important factors to consider when using the service-driven approach.

Choosing Between Abstract Process Starters, Services, and Service Subprocesses

Choose a process starter activity to start a process when an event occurs. There can be only one process starter in a process.

Do not create a process with a technology specific process starter such as an HTTP or JMS process starter.

Choose a service if you want to expose the operations available in a process outside the application module.

Choose a service subprocess to make your business process easier to understand and debug. A subprocess is invoked by a parent process and the output of the subprocess is used in the main process. A parent process calls a subprocess in two ways: in-line and non-in-line. At run time, an in-line subprocess executes as part of the parent process' job, while a non-in-line subprocess spawns a new job.

Choosing between REST and SOAP Bindings

A process service is exposed to external consumers by configuring bindings such as REST or SOAP.

	Data Abstraction	State Information	Overhead of Additional Parameters (Headers or other SOAP elements)
REST Services	Resources	Stateless	Less
SOAP Services	Operations	Stateful	High

TIBCO Business Studio™ Container Edition Essentials

TIBCO Business Studio™ Container Edition is an Eclipse-based integration development environment that is used to design and test BusinessWorks Container Edition applications.

If you are familiar with the TIBCO Business Studio™ Container Edition UI, skip to the section Developing a Basic Process.

Starting TIBCO Business Studio™ Container Edition

To start TIBCO Business Studio Container Edition on Windows, select **Start** > **All Programs** > **TIBCO_HOME** > **TIBCO Business Studio Container Edition 1.0** > **Studio for Designers**. On Linux or Mac OS, select the TIBCO Business Studio™ Container Edition executable located at *<TIBCO_HOME>*`/studio/4.0/eclipse/`.

On the Workspace Launcher dialog, accept the default workspace or browse to create a new workspace, and then click **OK**. TIBCO Business Studio™ Container Edition starts and the default development environment, called a *workbench*, displays. A welcome screen is displayed in the window when a workspace is opened for the first time.

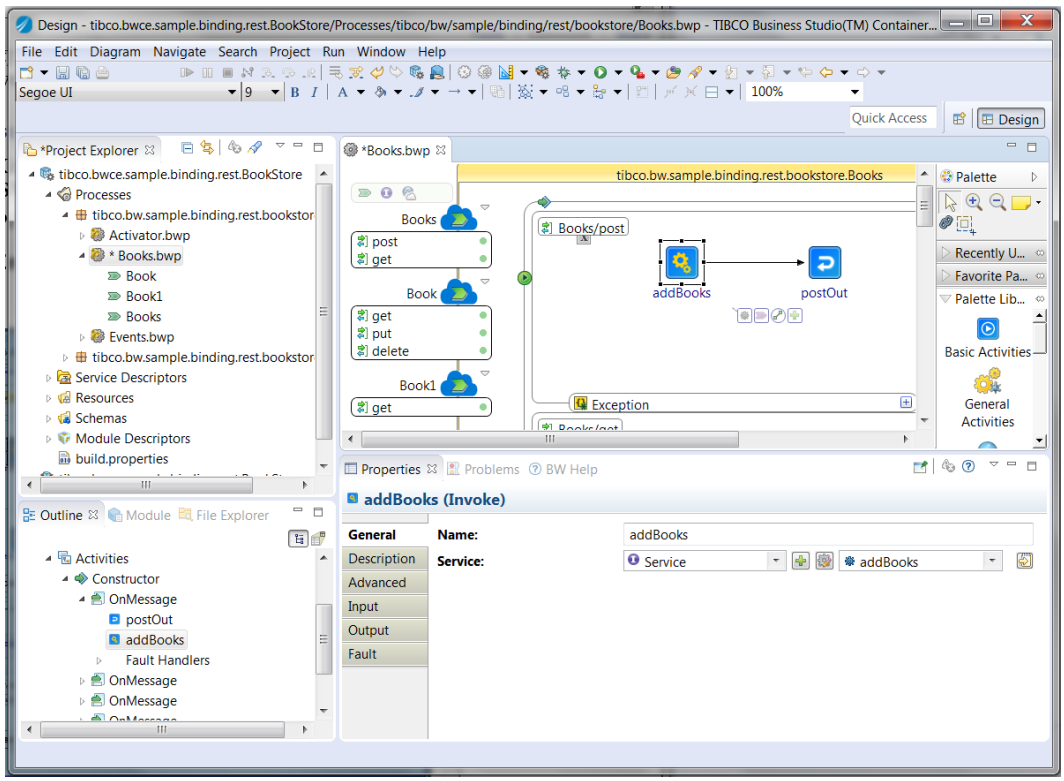
On Mac OS, TIBCO Business Studio™ Container Edition displays the Subversion Native Library Not Available dialog box if the SVN interface is set to JavaHL (default) and the JavaHL libraries are not available. To ensure that the dialog box is not displayed each time you start TIBCO Business Studio™ Container Edition, perform one of the following:

- Install the JavaHL libraries. See http://subclipse.tigris.org/wiki/JavaHL for instructions.
- Update the SVN interface to use SVNKit instead of JavaHL. Select **Window** > **Preferences** and in the Preferences dialog box, select **Team** > **SVN**. For the **`SVN interface Client`** field, select `SVNKit (Pure Java) interface` from the drop-down list.

TIBCO Business Studio™ Container Edition Development Environment

TIBCO Business Studio™ Container Edition provides a workbench that is used to create, manage, and navigate resources in your Eclipse workspace. A *workspace* is the location on your machine where the artifacts related to your TIBCO BusinessWorks™ Container Edition projects are stored.

TIBCO Business Studio™ Container Edition Workbench



The Studio workbench has features such as:

- **Menu:** Contains menu items such as File, Edit, Navigate, Search, Project, Run, Window, and Help.
- **Toolbar:** Contains buttons for the frequently used commands such as:
  - New
  - Save
  - Enable/Disable Business Studio™ Container Edition Capabilities
  - Create a new BusinessWorks Application Module
  - Create a new BusinessWorks Shared Module
  - Debug
  - Run Run
- **Perspectives**: Contain an initial set and layout of views that are needed to perform a certain task. TIBCO Business Studio™ Container Edition launches the Design perspective by default. Use the Design perspective when designing a process and the Debug perspective when testing and debugging a process. To change the perspective, select **Window** > **Open Perspective** > ***perspective_name*** from the main menu. Or, you can click the icon at the top right-hand side of the workbench and select the perspective to open.
- **Views**: Display resources and allow for navigation in the workbench. For example, the Project Explorer view displays the TIBCO BusinessWorks™ Container Edition applications, modules, and

other resources in your workspace, and the Properties view displays the properties for the selected resource. To open a view, select **Window** > **Show View** > *view_name* from the main menu.

- **Editors**: Provide a canvas to configure, edit, or browse a resource. Double-click a resource in a view to open the appropriate editor for the selected resource. For example, double-click on an TIBCO BusinessWorks™ Container Edition process (`MortgageAppConsumer.bwp`) in the Project Explorer view to open the process in the editor.

Designing a Process

Design a process in TIBCO Business Studio™ Container Edition to implement the business logic. See Developing a Basic Process.

Testing and Debugging an Application

Using TIBCO Business Studio™ Container Edition you can test and debug your application from the design-time.

To run the selected application, select **Run** > **Run** from the main menu, or click  on the toolbar.

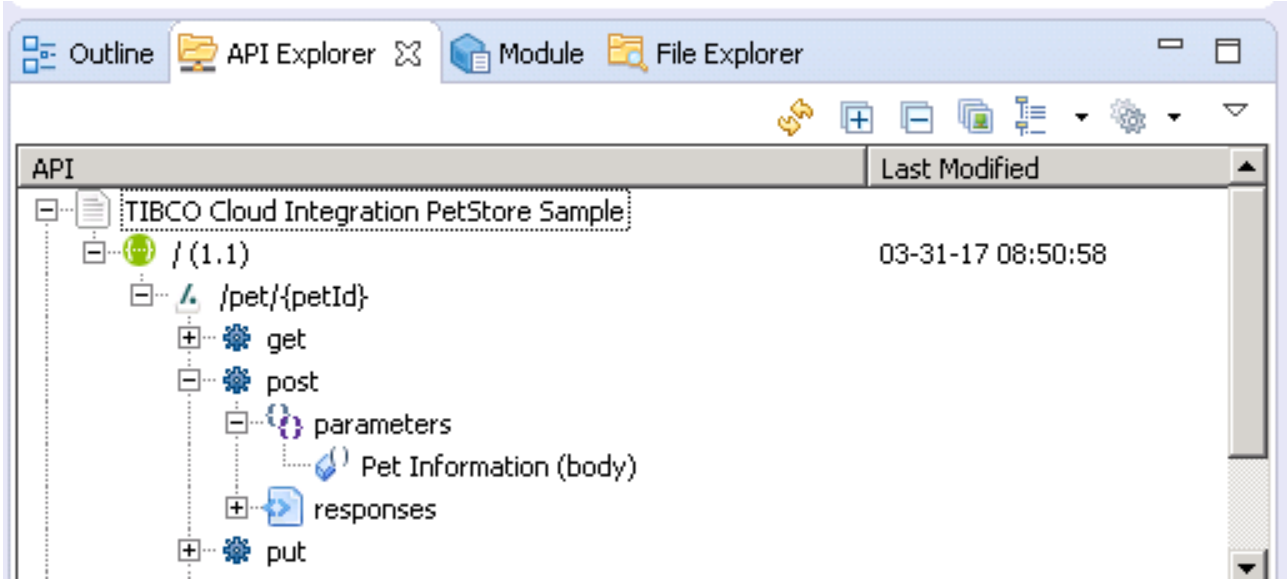
To execute and debug the application, select **Run** > **Debug** from the main menu, or click  on the toolbar.

By default, the project displayed in the Process Editor launches. You can run or debug an application using a specific configuration. Create one or more configurations for your application by selecting **Run** > **Run Configurations** from the main menu and specifying the following:

- Bundles to be executed.
- Arguments such as the target operating system, target architecture, target web services, engine properties, and so on.
- Settings that define the Java Runtime Environment including the Java executable, runtime JRE, configuration area, and so on.
- Tracing criteria for the OSGi JAR file, if needed.
- Common options such as saving the results either as local files or as shared files, displaying them in the menus (Debug and/or Run), and defining encoding for the result files.

API Explorer

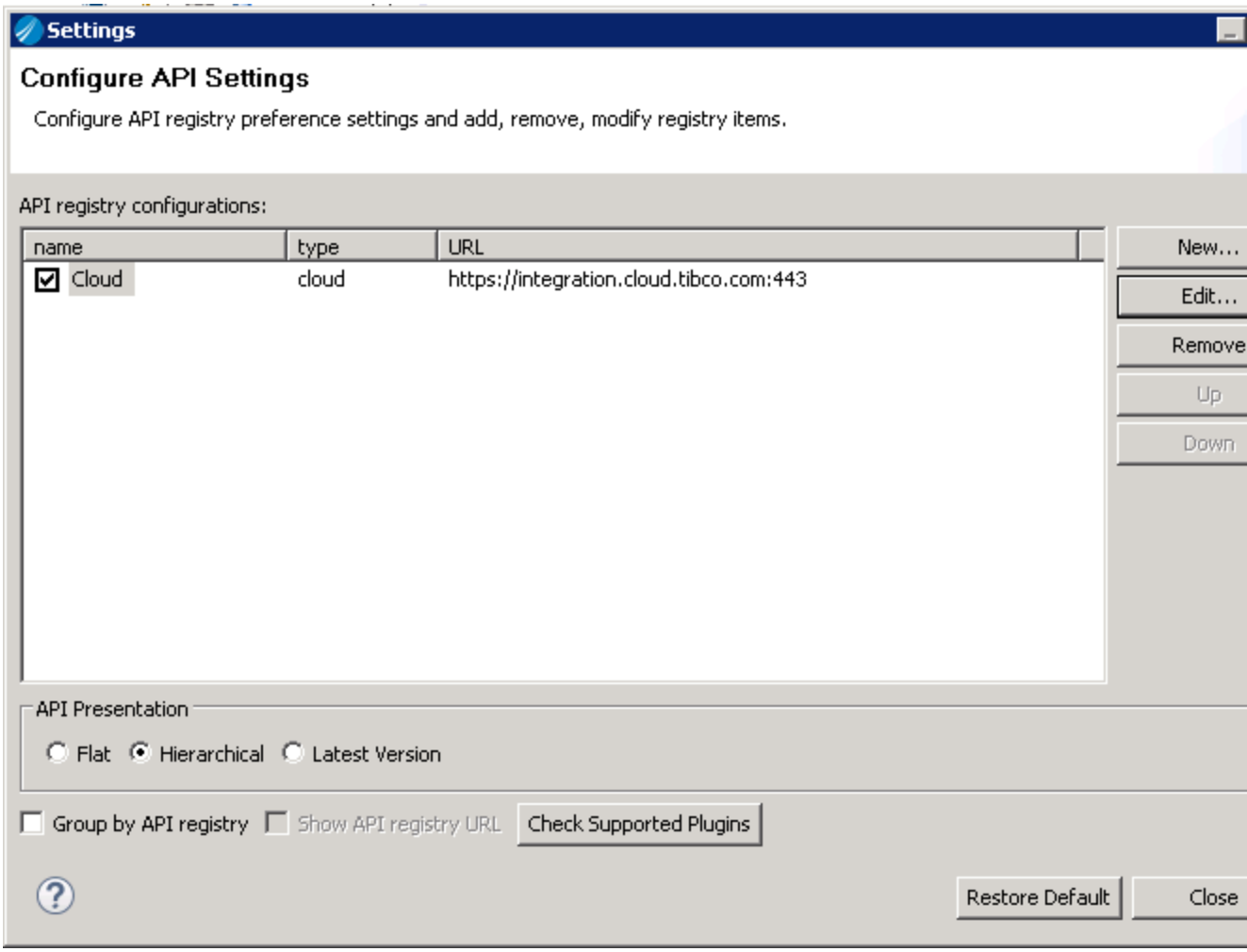
Displays a connected view of the TIBCO BusinessWorks Container Edition API Modeler residing in the cloud. This view shows abstract APIs that were created in API Modeler. You can also view the APIs residing on your local machine from the API Explorer.



When you open the TIBCO Business Studio Container Edition for the very first time, you will need to enter your credentials for the registry site by opening the Settings dialog and double-clicking on the registry name and entering your username and password for the site in the resulting dialog. To open the Settings dialog click the ( ) button on the upper right corner of the API Explorer view and click **Settings**. This will populate the API Explorer with the APIs that are available in the registry.

Adding a new registry to the API Explorer view

Use the Settings dialog in the **API Explorer** to add a new registry (location) from where you want to view the APIs. To open the Settings dialog click the ( ) button on the upper right corner of the API Explorer view and click **Settings**.



By default, the Settings dialog is configured with a Cloud registry which is set to the URL for the API Modeler.

To create a new registry:

1. Click the **New** button.
2. Enter a name for the registry **Name** field.
3. Select whether the registry will be pointing to a local folder on your machine (**Local Folder**) or to a URL in the cloud (**Cloud**).
4. Provide the location of the registry in the **URL** field. If the registry points to a location on the cloud, you will need to provide the authentication details for it in the **Username** and **Password** text boxes.
5. Click **Finish**.

To edit an existing registry entry:

1. Click the name of the registry and click **Edit**.
2. Make your edits to the entry. You can change the name of the registry, delete the registry configuration by clicking **Remove**, or changing the order in which the registries show up in the API Explorer by using the **Up** and **Down** button.
3. Click **Finish** when you are done with your edits.

To select specific registries to display in the API Explorer view, select the checkbox next to the entry you want to display. You can also do so at a later time. See Filtering the APIs in the API Explorer View for details on how to do this.

Setting the presentation of the APIs in the API Explorer view:

In this dialog, you can specify how the discovered APIs will appear in the API Explorer:

- **API Presentation** - specifies how the APIs will appear in the API Explorer

  Flat - displays the APIs as a flat list with each API's version number displayed next to its name in parenthesis. If there are multiple versions of the same API, each version will be shown as a separate API, hence multiple APIs with the same name but different version numbers.

  Hierarchical - displays every API as a hierarchy of API name lable with version number folder under it and the actual API under the version folder. If there are multiple versions for an API, each version will be listed in its own separate folder under the API name label.

  Latest Version - displays only the latest version of the API, even though there might be multiple versions available.
- **Group by API registry** - groups the APIs according to the registry from which they were discovered. You also have the option to display the URL of the APIs next to the registry name by selecting the **Show API Registry URL** check box.

You should now see the APIs displayed in the API Explorer in the format that you specified in the **Settings** dialog. Expanding an API will show you its version, the resource path, and the operations you can perform on that resource.

The API Explorer view has the following quick-access buttons that you can use to format the way the APIs are listed:

- Refresh
- Expand All
- Collapse All
- Group by API Registry
- API Presentation
- API Registries. Selecting a registry from this drop-down list toggles between displaying and hiding the registry in the API Explorer.

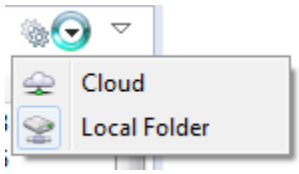
Searching for APIs in API Explorer:

Use the search filter that appears at the bottom of the API Explorer view to search for API names that match the string that you enter in the **Filter** text box. You can search by typing in the version number, the full API name, or a full word within an API name. Wildcards is not supported. The search is case insensitive.

Filtering the APIs in the API Explorer view

If your APIs reside in multiple locations and you have set up the API registries in the Settings dialog of the API Modeler view, you can filter the APIs in API Modeler such that it shows you only the APIs available in a certain registry.

To do so, click the ( ) button on the upper right corner of the API Modeler view and select the registry whose APIs you want to view.



Settings

The Settings dialog is configured by default to access the TIBCO BusinessWorks Container Edition site. You can configure additional registries that you might need to access using this dialog.

API Registry Configurations

Configure a new API registry by clicking the **New** button. Enter the following information:

name - A unique name for the registry

type - Registry type. If the registry is in the cloud, you must select **Cloud** and enter its URL in **URL** text box. If the API resides on your local file system, select **Local Folder** and browse to the folder using the **Browse** button.

URL - If you selected Cloud as your registry type, you must enter the site's URL in this text box.

Authentication - When creating a new API registry, you will be prompted to enter your username and password for the registry that exists on the Cloud.

API Presentation

Configure how you want your API to appear in this view. The three types of presentations available are:

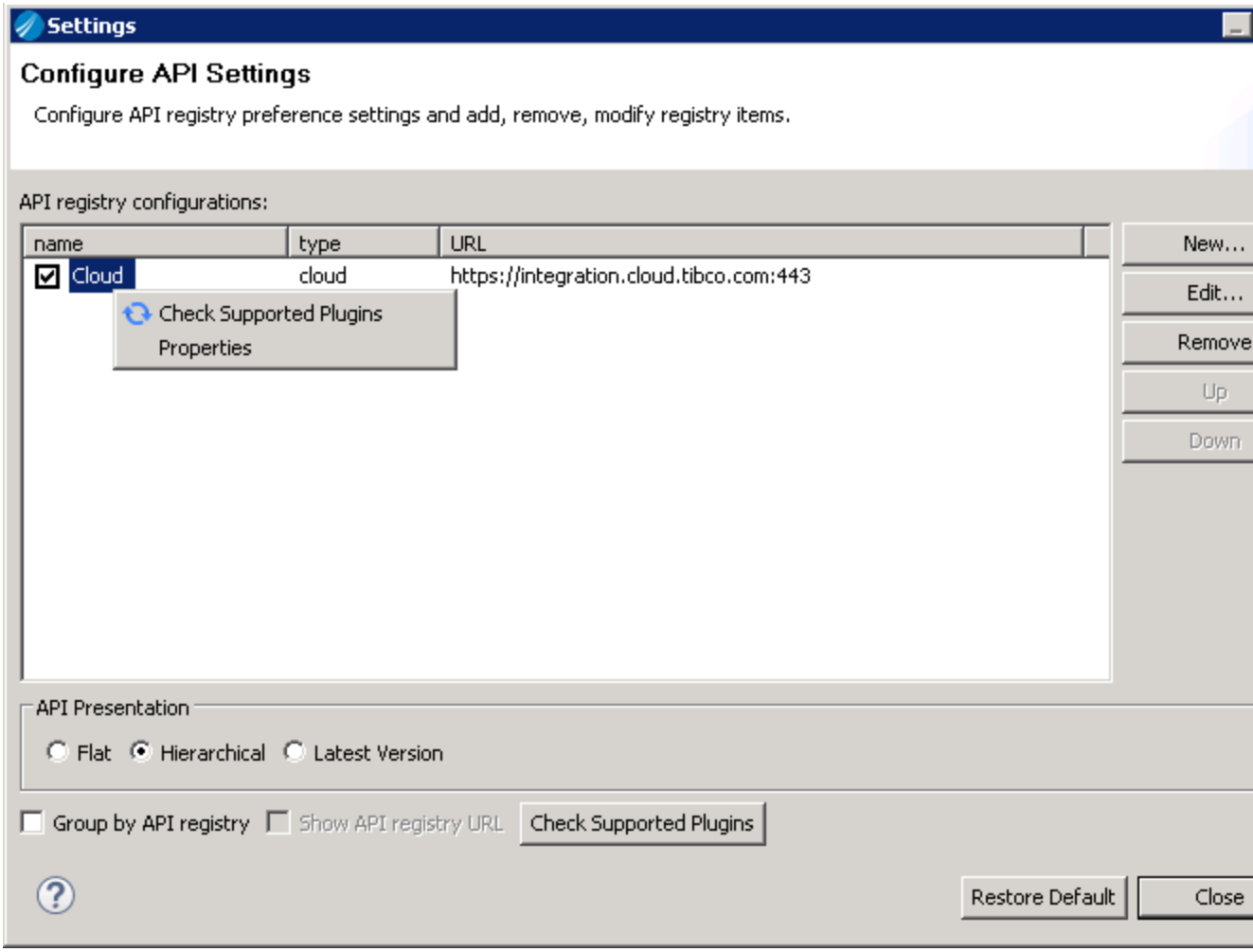
- **Flat** - Displays the APIs as a flat list with each API's version number displayed next to its name in parenthesis. If there are multiple versions of the same API, each version will be shown as a separate API, hence multiple APIs with the same name but different version numbers.
- **Hierarchical** - Displays every API as a hierarchy of API name lable with version number folder under it and the actual API under the version folder. If there are multiple versions for an API, each version will be listed in its own separate folder under the API name label.
- **Latest Version** - If one or more APIs in your registry has multiple versions, selecting this option will show only the latest version of the API and hide the older versions.

Other Configurations

Group by API registry - Groups the APIs according to the registry from which they were discovered.

Show API Registry URL - Displays the URL of the APIs next to the registry name.

Check Supported Plugins - This button refreshes the supported list of plug-ins from TIBCO BusinessWorks Container Edition . When you import an existing project that uses plug-ins, you can validate that the plug-ins used in the project are supported in TIBCO BusinessWorks Container Edition by clicking this button. You will see a message saying that the supported plug-ins are synchronized. The read-only list of supported plug-ins shows up in the **Supported Plugins** tab of the **Properties** dialog that you can access from the right-click menu as shown below. You can verify that you have the latest list from the synchronization timestamp at the bottom of the **Properties** dialog. You can also access the **Check Supported Plugins** option by right-clicking in the API registry that you want to connect to and selecting it from the resulting context menu.



This list represents the plug-ins that are available to your projects in TIBCO BusinessWorks Container Edition during runtime. In order to use a plug-in during design-time, you must have the plug-in installed locally on your machine. If your project uses a plug-in that is not supported in TIBCO BusinessWorks Container Edition , you will see an error saying so when pushing the project to the cloud.

Properties - The **Properties** context menu item opens a dialog which provides information about the registry from which you selected **Properties** in its **General** tab. The **Supported Plugins** tab provides a read-only list of plug-ins that are supported in TIBCO BusinessWorks Container Edition .

Entity Naming Conventions

Most of the TIBCO BusinessWorks™ Container Edition named entities are modeled as NCNames (or as a subset of an NCNames). These include activity names, service names, reference names, binding names, and component names.

Process names and shared resource names are represented as a subset of an NCName as they do not allow the use of a dot (.) character in their names. A small set of named entities are modeled as OSGi symbolic names. This set includes application names, module names, process package names, and shared resource package names.

NCName stands for XML "non-colonized" name. See http://www.w3.org/TR/xmlschema-2/#NCName for the W3C definition of NCName. NCName represents the set of characters that conforms to the following restrictions:

- Can include letters or numbers A-Z, a-z, 0-9, -, _
- Cannot include the following characters: @, :, $, %, &, /, +, ,, ;,), and white space characters.
- Cannot begin with a number, dot (.), or minus (-) character. However, these characters can appear later in an NCName.

The OSGi symbolic name is defined as part of the OSGi Specification, which is available at http://www.osgi.org/download/r5/osgi.core-5.0.0.pdf. OSGi symbolic names are represented using the following syntax:

```
symbolic-name ::= token('.'token)*
token ::= ( alphanum | '_' | '-' )+
alphanum ::= alpha | digit
digit ::= [0..9]
alpha ::= [a..zA..Z]
```

Developing a Basic Process

Using processes you can implement business logic that obtains and manages the flow of information in an enterprise between a source and different destinations.

TIBCO Business Studio™ Container Edition Workbench provides a design environment to develop and test a process. Developing a simple process consists of the following phases:

1. Creating an Application Module to contain the processes and shared resources.
2. Creating a Shared Module (optional).
3. Creating a Process that implements the business logic.
4. Configuring a Process to define the runtime behavior of the process.
5. Adding Activities to the process that describe the tasks in the business logic.
6. Connecting Activities with Transitions to describe the business process flow between activities in a process.
7. Configuring the input and output data for the activities. See Working with Standard Activity Features for details.

At run time, the process engine executes the process definition and creates an instance of the process definition called a *job*. A job automates your business process by executing what is described in the process definition.

Conceptual information about processes and their use is provided in the *Concepts* guide.

Creating an Application Module

Application modules are packages containing one or more processes, shared resources, and metadata such as name, version, dependencies, and so on.
The New BusinessWorks Application Module wizard helps create an application module. There are multiple ways to launch the wizard:

- From the main menu, select **File** > **New** > **BusinessWorks Resources** and then select  **BusinessWorks Application Module**.
- Right-click in the Project Explorer view and choose **New** > **BusinessWorks Application Module**.

Specify the values for the following fields in the wizard:

1. **Project name**: Name of the application module.
2. **Use default location**: Specifies the location on disk to store the application module's data files. By default, this value is set to the workspace. To change, clear the check box and browse to select the location to be used.
3. **Version**: Version of the application module.
4. **Create empty process**: Selected by default to create an empty process with the specified name (default: Process). Clear the check box if you do not want to create an empty process.
5. **Create Application**: Selected by default to create an application with the specified name. Clear the check box if you do not want to create an application.
6. **Use Java Configuration**: Select to provide the Java tooling capabilities in your module. Selecting this option creates a Java module.
7. Click **Finish**.

Package names must be unique within an application. If there are two packages with the same name in an application, then you must either rename one of the packages or remove one of the packages from the application.

Result

An application module with the specified name is created and opened in the workbench. If the options to create an empty process and an application were selected, the process and application with the specified names are also created.

Creating a Shared Module

Shared modules are the smallest unit of resources that are named, versioned, and packaged as part of an application and can be used by other modules that are part of the same application.

The New BusinessWorks Shared Module wizard helps create a shared module. There are multiple ways to launch the wizard:

- From the main menu, select **File** > **New** > **BusinessWorks Resources** and then select  **BusinessWorks Shared Module**.
- Right-click in the Project Explorer view and select **New** > **BusinessWorks Shared Module**.

Specify the values for the following fields in the wizard:

1. **Project name**: Name of the shared module.
2. **Use default location**: Specifies the location on disk to store the shared module's data files. By default, this value is set to the workspace. To change, clear the check box and browse to select the location to be used.
3. **Version**: Version of the shared module.
4. **Use Java Configuration**: Select to provide the Java tooling capabilities in your module. Selecting this option creates a Java module.
5. Click **Finish**.

Result

A shared module with the specified name is created and opened in the workbench.

Exporting a Shared Module as a Binary Shared Module

You can create a binary shared module from a shared module. However, you cannot convert a binary shared module to a regular shared module.

To export a shared module as a binary shared module, begin by implementing the process you want to share. The process must have a descriptive name and a description. Next, test the process by calling it from a test application. Once satisfied, you create a zip archive file for the project which contains the process and distribute that zip using a mechanism such as email, FTP, or a web page, that is external to TIBCO Business Studio Container Edition .

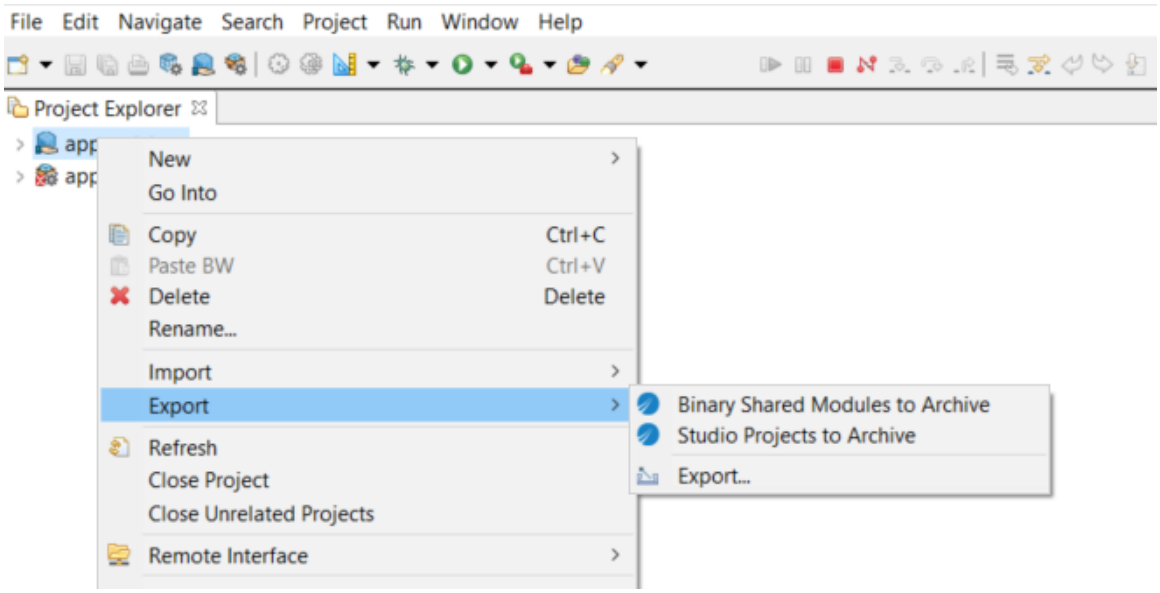
Back up the shared module by exporting the project as an archive file. To do this, select **Export** > **Studio Projects to Archives**.

TIBCO Business Studio Container Edition

To export a shared module as a binary shared module from TIBCO Business Studio Container Edition , follow these steps.

Procedure

1. In Project Explorer, right-click on the shared module folder, and choose one of the following options to begin exporting the shared module as a binary shared module:
   - Select **Export** > **Export**. In the Export dialog, expand the **General** node, select **Binary Shared Modules to Archive**, and click **Next**.
   - Select **Export** > **Binary Shared Modules to Archive**.



2. Select the checkbox next to the shared module that you want to convert to a binary shared module.
3. In the **To Archive File** field, navigate to the folder where you want it created and enter a name for the binary shared module you want to create and click **Save**.
4. Click **Finish** in the Export Project dialog.

Result

The shared module is exported as a binary shared module.

To confirm the shared module was successfully exported as a binary shared module, import the binary shared module into a new workspace, and expand the project. All application folders and details, with the exception of the folders under the Module Descriptors folder, are hidden. Optionally, check the `MANIFEST.MF` file, and confirm the `TIBCO-BW-SharedModuleType` header is set as follows:

```
TIBCO-BW-SharedModuleType: binary
```

CLI

To export a shared module as a binary shared module from the command line, follow these steps:

Prerequisites

- Start the bwdesign utility. To do this, follow these steps:

1. Open a terminal and navigate to `BW_HOME\bin`.
2. Type `bwdesign -data <TIBCO_BusinessStudio_workspace_absolutePath>`. For example, `bwdesign -data C:\myWorkspace`.

- Back up the shared module by exporting the project as a zip or EAR file. To do this, type `-export [options] [projects] [outputfolder]`

Type `export -binary <shared_module>` . For example, `export -binary shared_petstore`. Optionally, type `export -bin <shared_module>`. For more details about the **`-binary`** and **`-bin`** commands, type **`export --help`** .

Result

The shared module is exported as a binary shared module.

To confirm the shared module was exported as binary shared module, import the binary shared module into a new workspace by typing `bwdesign -data <TIBCO_BusinessStudio_workspace_absolutePath>`. After doing this, expand the project in the Project Explorer to verify that all application folders and details, with the exception of the folders under the Module Descriptors folder, are hidden. Optionally, check the `MANIFEST.MF` file, and confirm the `TIBCO-BW-SharedModuleType` header is set as follows:

```
TIBCO-BW-SharedModuleType: binary
```

Using Binary Shared Modules in your Project

To use a binary shared module, you begin by importing the archive into your workspace where it appears like any other shared module, except that the internal details of the shared module are not visible. You use a binary shared module in the same way as you would use any other shared module. You can see the processes in the Project Explorer but cannot view their diagrams in the Process Editor or open them with a text editor to decipher their models.

You can see the following artifacts associated with a binary shared module:

- Process and package name
- XML schema files associated with the module

Since the schema files are in plain text, you will be able to modify them. Keep in mind though that if and when you import a newer version of the module into your workspace, your modifications to the schema files will be overwritten.

- Shared resources - you can reference them, but cannot edit them
- Module Descriptor folder - only the Overview item is available under this folder
- Module Descriptor editor will be able to display the Overview page only. All other fields will be disabled

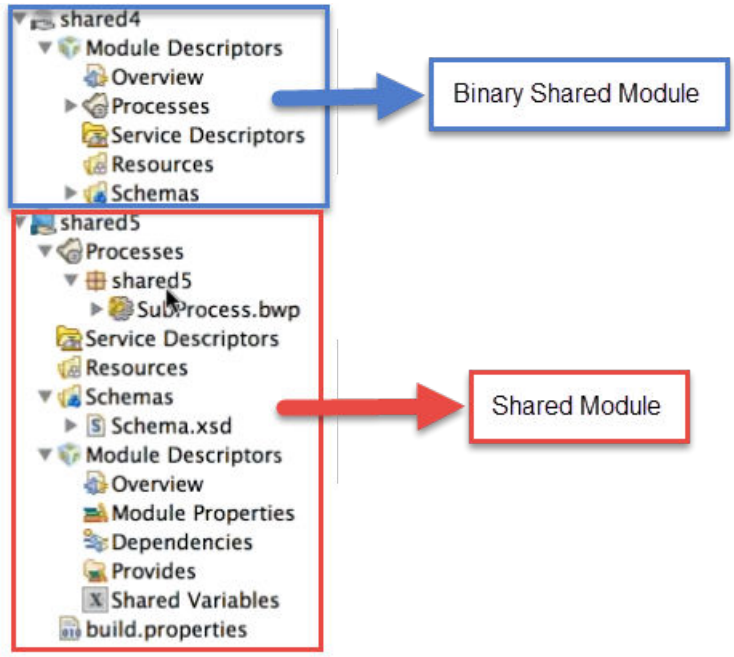
You can implement a Call Process activity that invokes the functionality in the binary shared module. When deploying your application, the binary shared modules are included in the application like any other shared module.

Difference between a Shared Module and a Binary Shared Module

This section describes the difference between a shared module and a binary shared module.

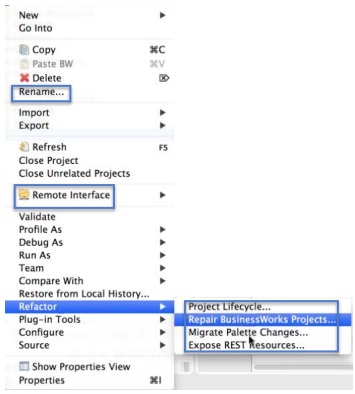
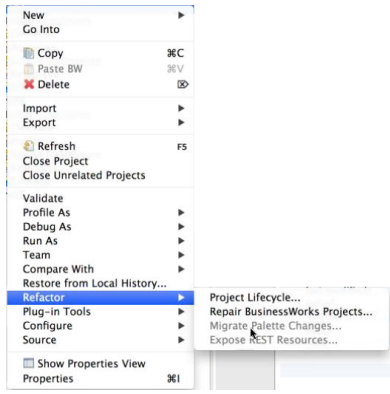
In Project Explorer

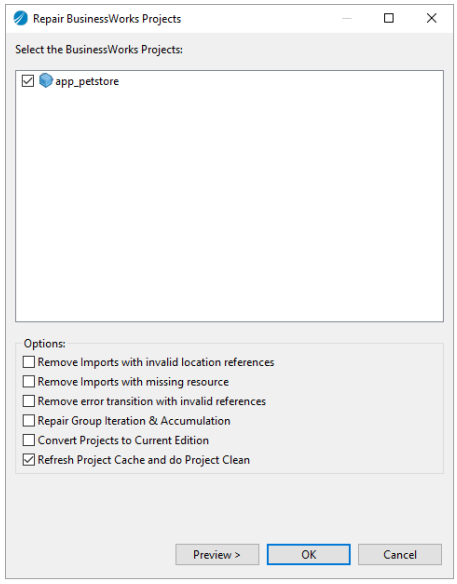
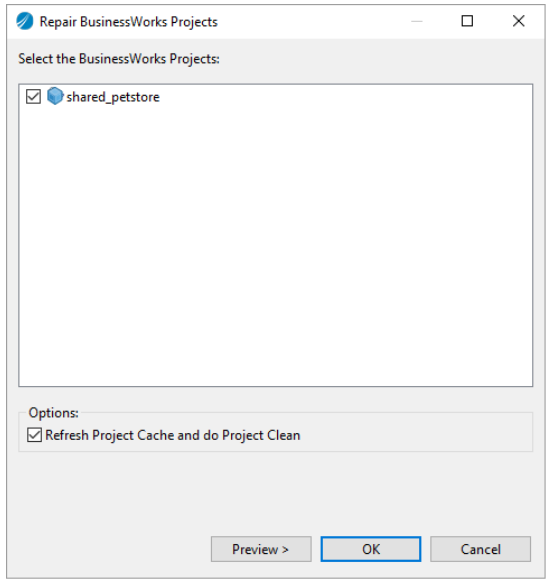
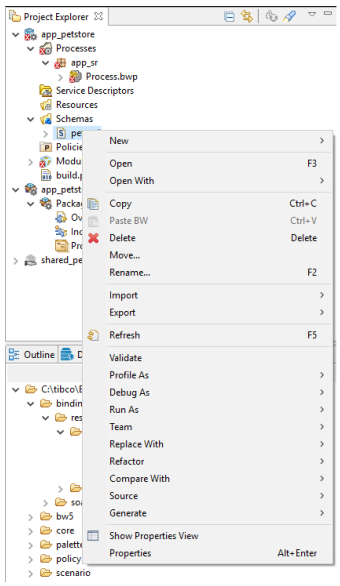
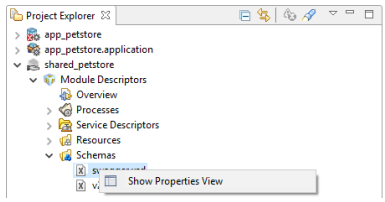
The image below shows you the difference between a shared module (shared5, in the image below) and a binary shared module (shared4). Notice that almost all the editable artifacts (such as Module Properties, Dependencies and Shared Variables) are missing from the binary shared module tree. This is one way to prevent the binary shared module from being edited.



Menu Items

At the project level some of the context menus items are disabled in the binary shared modules. At the resource level all the menu items except for Show Properties View are disabled.

Context Menu	Shared Module	Binary Shared Module
At the project level: Right-click menu from process name		

Context Menu	Shared Module	Binary Shared Module
Repair BusinessWorks Project dialog		
Context menus at Processes, Service Descriptors, Resources, and Schemas level		

Public Processes and Internal Processes

A binary shared module can contain two types of processes - public processes and private (internal or inline) processes. While a public process in a binary shared module can be called by an application, a private process within the module is meant for consumption by the public processes within that binary shared module only. By default, the private processes are **not** visible in the Project Explorer.

To view the private processes in the Project Explorer, do the following:

1. In the Project Explorer, click the **View Menu** button (▾) and select **Customize View**.

2. In the Available Customizations dialog, uncheck the **BW binary private processes** check box and click **OK**.

Creating a Process

Processes are always contained in a process package. When creating a process, either create a new process package or select an existing package in which the new process is to be created.

Prerequisites

A module must exist to which processes can be added. If a module does not exist, create a new module before creating a process.

The BusinessWorks Process Creation wizard helps create a generic business process. By default, it is configured to create a process with name `Process`. There are multiple ways to launch the wizard:

- From the main menu, select **File** > **New** > **BusinessWorks Resources** and then select **BusinessWorks Process**.
- From the **Module Descriptors** > **Overview** getting started area, click **Create a New BusinessWorks Process**.
- Right-click on the **Processes** folder in the Project Explorer view, and then select **New** > **BusinessWorks Process**.

Specify the values for the following fields in the wizard:

1. **Process Folder**: Name of the module and the Process special folder where the process will be located. You can add multiple folders in Project Explorer and then update this field to select the new folder. For example: `bw.test.app/Processes`.
2. **Package**: Name of the package in the module where the new process is added. Accept the default package, or browse to select a different package name. For example: `bw.test.app.main`.
3. **Process Name**: Name of the new process. For example: `MainProcess`
4. **Modifiers**: Designate whether the process will be public or private. This can be changed later.
5. **Patterns**: Choose the pattern **Empty Process** when creating a process.

   To create a subprocess, choose the pattern **Subprocess**. See Creating Sub-Processes for details on creating a subprocess.
6. Click **Finish** to create a new empty process.

Result

A process with the specified name is created and opened in the Process Editor.

What to do next

After creating the process proceed to:

- Configure the process as described in Configuring a Process.
- Add activities to the process as described in Adding Activities.

Configuring a Process

Process configuration defines the behavior of a process at runtime. You can specify (or edit) the modifiers, mode, and activation type for a process. You can also define process properties and process variables, add or remove services and references, and configure the process dependencies.

Prerequisites

Open a process in TIBCO Business Studio™ Container Edition if it is not already open and go to the Properties view.

Procedure

- Configure the general properties for a process by selecting the **General** tab in the Properties view.

Property Name	Description
Package	Displays the name of the package containing the package. This field is not editable.
Name	Name of the process. This field is not editable.
Target Namespace	Target namespace for the process. You can specify a different target namespace for the process.
Modifiers	Modifiers define the visibility of the process outside its package: • `Public`: can be invoked by processes that are defined either inside or outside the package. • `Private`: can be invoked only by processes that are part of the same package.

Creating a Subprocess

Subprocesses are designed for complex business processes to make the main process easier to understand and debug. Subprocesses are called inside the main process and their output is used in the main process.

The BusinessWorks Process Creation wizard helps create a subprocess. There are multiple ways to launch the wizard:

- From the main menu, select **File** > **New** > **BusinessWorks Resources** and then select **BusinessWorks Sub Process**.
- From the **Module Descriptors** > **Overview** getting started area, click **Create a New BusinessWorks Sub Process**.
- Right-click on the **Processes** folder in the Project Explorer view, and then select **New** > **BusinessWorks Sub Process**.
- Right-click on the **Processes** folder in the Project Explorer view, and then select **New** > **BusinessWorks Process**.

Specify the values for the following fields in the wizard:

1. **Process Folder**: Name of the module and the special folder where the subprocess will be located.
2. **Package**: Name of the package in the module where the new subprocess is to be added. Accept the default package, or browse to select a different package name.

3. **Process Name**: Name of the subprocess.
4. **Modifiers**: Designate whether the process will be public or private. This can be changed later.
5. **Interface Mechanism**: Select either **Direct** or **Service**.
    - **Direct**: Select this option to create a non-WSDL-based subprocess. When you select this option, a new subprocess, containing a **Start** and **End** activity, is created.
    - **Service**: Select this option to create a WSDL-based subprocess. Next, choose one of the following options:
        - **Default**: Select **Inline** to create an inline subprocess. Select **Standalone** to create a standalone subprocess.
        - **Custom**: Select this option and click **Next** to create a new WSDL interface or use an existing WSDL interface for the subprocess.
6. Click **Finish** to create a service subprocess.

Result

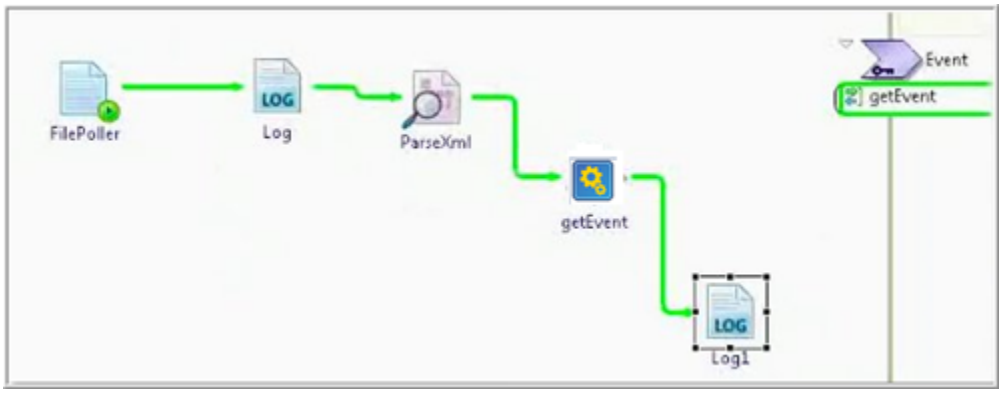
A subprocess with the specified name, and containing a **Start** and **End** activity, is created and opened in the Process Editor.

Parent Process and a SubProcess Example

Consider an example that illustrates how a parent process is designed to call a subprocess and collect data from that subprocess.

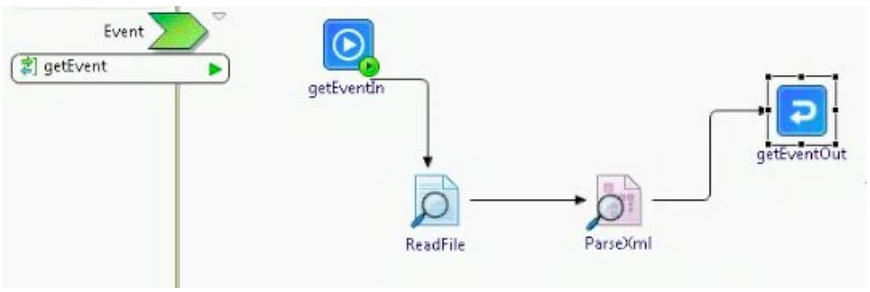
The parent process consists of a **getEvent** activity that calls the subprocess.

Parent Process



The subprocess implements the interface `getEvent` and returns the output back to the parent process. The parent process then logs the output received from the subprocess in a log file.

Sub Process

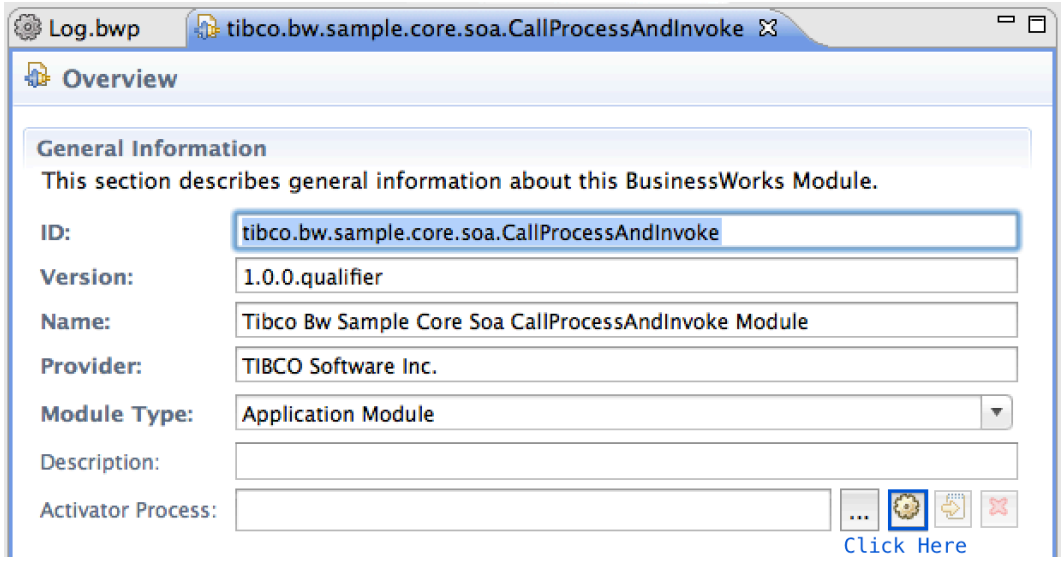


Creating an Activator Process

An activator process consists of two service operations, On StartUp and On ShutDown, which can be used to perform tasks when an application starts or after an application stops.
An application module can contain only one activator process. The following steps describe how to create an activator process for an application module.

Procedure

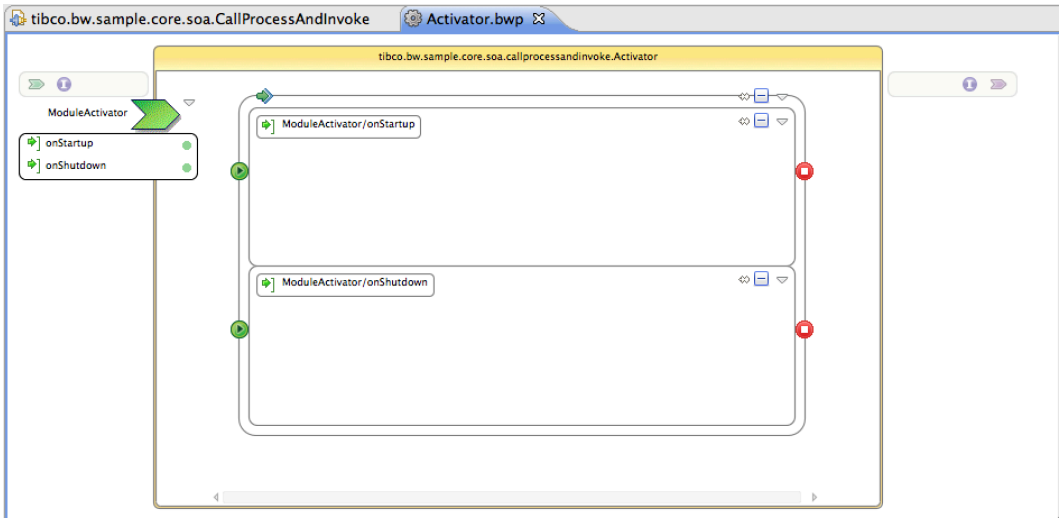
1. From the Module Descriptors > Overview > General Information area, click the  icon in front of the Activator Process field.



2. Review the fields in the Create Activator Process wizard and click **Finish** to create an activator process.

Result

An activator process with the service operations On StartUp and On ShutDown is created.



Adding Activities

Activities are the individual units of work in a process.

There are multiple ways to add activities in a process: from the right-click menu on the Process Editor, from the palettes, and from the Project Explorer.

Adding Activities from the Palettes

To add an activity to a process using the palette:

1. In the Palette view, select a palette from the library. All the activities available in the palette are displayed.
2. Select the activity that you want to add and drop it onto the process in Process Editor.
3. Configure the activity by specifying the values for the properties in the Properties view. The configuration properties are grouped under different tabs such as **General**, **Description**, **Input**, **Output**, and so on. For example, upon adding a **Log** activity, you can configure it by specifying the values for the properties under the tabs: **General**, **Description**, and **Input**. See Working with Standard Activity Features for details.

General and **Description** tabs are available for all activities to enter their name and a short description. Depending on the activity, these tabs may include additional fields such as variables, time, shared configurations, and other values that are required by the activity. Activities can also contain additional tabs such as **Input**, **Output**, **Fault**, and so on.

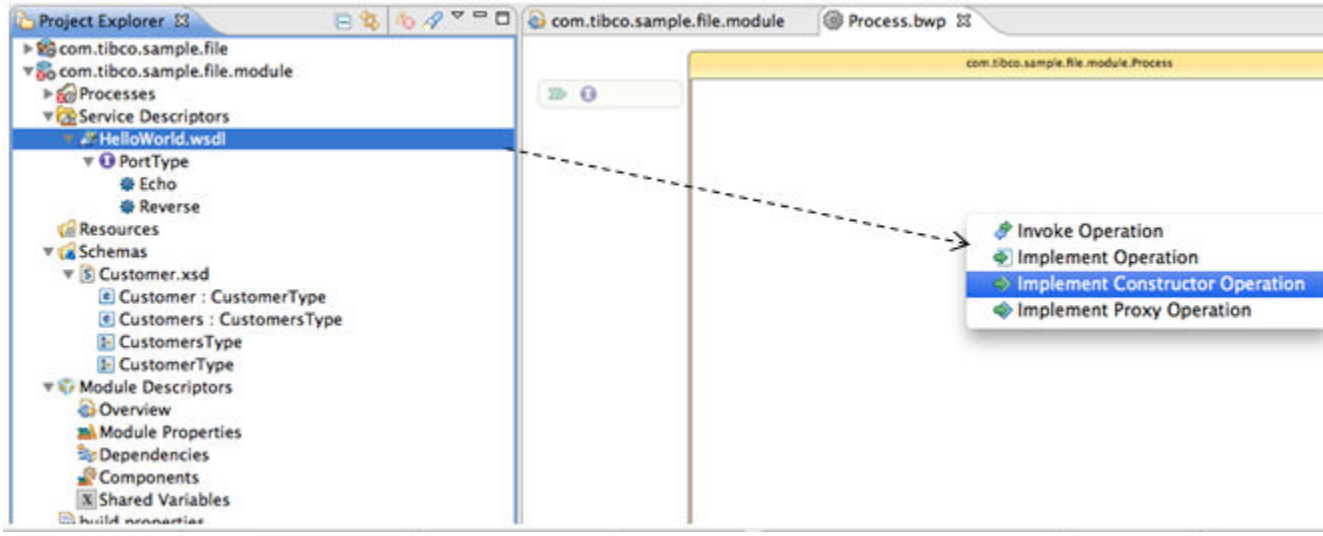
Adding Activities From the Project Explorer

You can add pre-configured activities to a process by dragging-and-dropping a selected resource such as a schema (XSD) or WSDL file from the Project Explorer. To do so, follow these steps:

1. In the Project Explorer, select a file such as a WSDL file that you want to use to create an activity.
2. Drag and drop the resource onto an existing process. The software parses the resource and provides a menu consisting of a list of pre-configured activities.
3. From the menu, select the activity you want to add to the process.

In the example, drag and drop the file `HelloWorld.wsdl` from the Project Explorer onto the process. A menu with a list of activities is presented. Select an activity to be added to the process.

Drag-and-Drop a Resource



An activity is connected to another activity by dragging the [+] symbol, positioning and dropping it, and then selecting the next activity from the menu selection. See Working with Transitions for details.

Working with Transitions

Transitions are used to connect two activities to represent the flow of process execution from one activity to the other.

Transitions are added between activities in a process and are configured to fit the process goal.

Adding a Transition

You can choose to add a transition in one of the following ways:

- Click the **Create a Transition** icon  in the Palette view's toolbar and draw a line between two activities that are to be connected.
- Select the beginning activity of the transition, click the icon  and drag and drop it on to the ending activity of the transition.

Configuring a Transition

After creating a transition specify the configuration information in the **General** tab of the Properties view:

1. **Label**: Add a label for the transition that will be available in the diagram. This label can be changed later.
2. **Fill Color**: Select **Color** for the transition from the basic colors or define a custom color. Color coding helps you distinguish among different transitions based on the conditions that are defined for them. The default color for Error is red, while the default color for other transition types is black.
3. **Condition Type**: Select the type of the condition for the selected transition: Success, Success with condition, Success with no matching condition, and Error.

You can define several types of conditions for a transition:

Success

Take this transition unconditionally. If the activity completes successfully, always transition to the activity the transition points to. This is the default condition for transitions.

Success with Condition

Specify a custom condition using XPath. If the activity completes successfully, and the condition evaluates to true, take the transition to the pointed-to activity.

Success with no Matching Condition

Take this transition when the activity completes successfully but *only* if no other transitions are taken. This is useful when multiple transitions with conditions are drawn to other activities. This condition type can be used to handle any cases not handled by the conditions on the other transitions.

Error

Take this transition if there is an error during the activity processing.

Error Transitions

Error transitions are taken if there is an error during the processing of an activity or group. When an activity or group throws an error or fault, none of the success conditions are taken; only the error transition is executed. An error transition can be added to process starter activities, signal-in activities, regular activities, and groups.

Activities and groups only support one error transition at a time.

Working with Standard Activity Features

Specify the required configuration elements to make the activity work. These configuration elements are available in the Properties view.

Each activity usually has two or more of the following tabs for specifying the characteristics of the activity:

General

This tab is available for all activities. In addition to the name of the activity, it also sets other parameters such as questions about directories and overwriting for file activities, class name for Java activities, host name, and port number for mail activities, modifiers, mode, and activation settings.

Description

This tab is available for all activities. You can write down any information you need to preserve for the activity.

Statement

This tab is available for query activities; used to define, validate, and execute a query.

Advanced

You can specify any advanced configuration parameters here.

Input Editor

Used to edit an output element by adding a complex anonymous type, complex element, primitive element, and so on. Not all activities have this option enabled. For more details see Input and Output.

Input

Using the tab you can map and transform output data from the previous activities in the process (including the event that starts the process) to input data for**Input** the activity. For more details see Input and Output.

Output Editor

This tab is used to choose or configure the output header element. Not all activities have this option enabled. For more details see Input and Output.

Output

This tab displays the output of the activity's data to the activities that follow in the process definition. For more details see Input and Output.

Fault

Lists the activity faults or various exceptions that might occur with this activity, such as `FileNotFoundException` or `IllegalCopyException`.

Input and Output

The **Input** tab is used to enter the data for an activity and the **Output** tab displays the output schema of an activity.

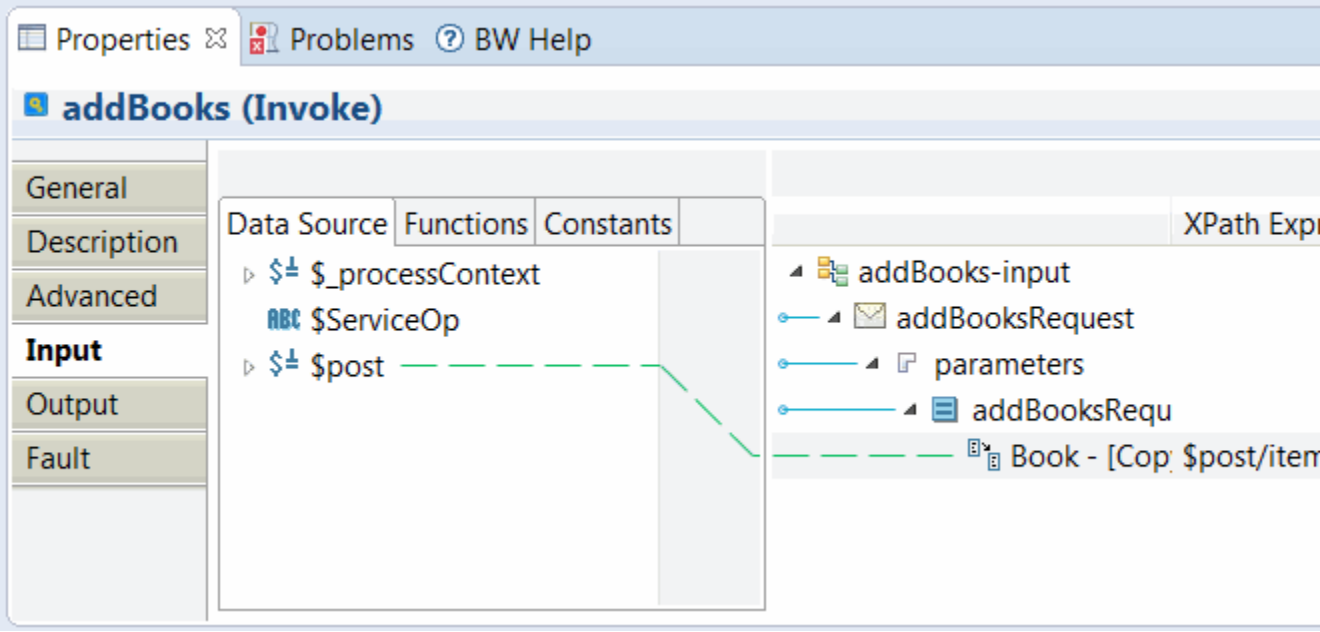
Configuring the Input Tab

The **Input** tab is available in the Properties view and is used to enter data for an activity. Input data for an activity can be any of the following:

- **Constant/Literal** specified using numbers or strings enclosed in quotes.

- **Regular Expression** specified using an existing schema item or by keying in a constant expression in the field.
- **Mapping** the output from previous activities to the current activity's input. Using the mapper, you can choose functions or constants from the **Functions** and **Constants** tabs with the mapped data.

Input Tab



1. Click on the desired item in the available schema in the Data Source panel, such as "name". Drag the item to the desired item in the Activity Input panel, such as "Message".
2. To type in a constant or expression, click on the schema item ("Message") in the Activity Input panel and type the constant or expression into the field.

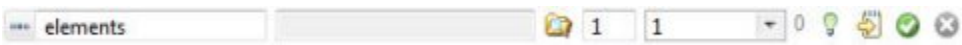
Configuring the Input Editor Tab

Using the **Input Editor** tab you can configure the input data for an activity.

Instead of specifying a constant or an expression for the schema item, you can first configure the sequence in which this message will appear by setting up the element it is contained in.

You can define the sequence of an element using the icons on the right:

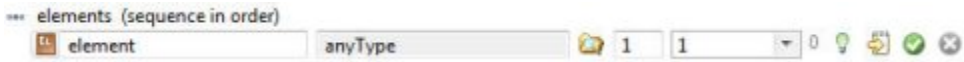
1. **Add Complex Anonymous Type**: Adds an element sequence that is defined by the following:



   a. Schema type definition or creating a new type definition.
   b. Number of Minimum Occurs (default is 1).
   c. Number of Maximum Occurs (1 or unbounded).
   d. Number of references to this resource (generated, in this case it is 0).
   e. Initiate Rename Schema Element: rename the schema element by entering the New Name and choosing the option whether to update the references to this element.

f. The remaining icons are Go To , Accept Changes , and Delete , which invoke the general editing tools.

2. Add Complex Element: This option adds a complex element that you can further define by the following:



a. The schema type definition or a new type definition (default is `anyType`)
b. Number of Minimum Occurs (default is 1).
c. Number of Maximum Occurs (1 or unbounded).
d. Number of references to this resource (generated, in this case it is 0).
e. Initiate Rename Schema Element: rename the schema element by entering the New Name and choosing the option whether to update the references to this element.

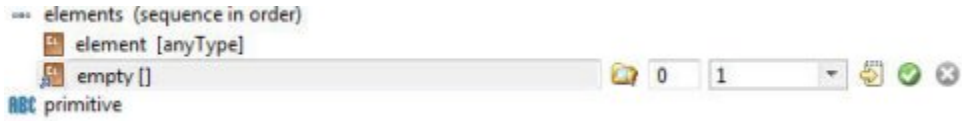
f. The remaining icons are Go To , Accept Changes , and Delete , which invoke the general editing tools.

3. **Add Primitive Element**: This option adds a primitive element that you can further define by the following:

a. Choosing by the Primitive Types: String, Integer, Decimal, Boolean, Date&Time, Binary, URI or Any.
b. Choosing by the Primitive Sub Types: String, Normalized String, Token, Language, Name. NC-Name, Q-Name, Name Token, Name Tokens, ID, ID ref, ID refs, Entity, and Entities.
c. Number of Minimum Occurs (default is 1).
d. Number of Maximum Occurs (1 or unbounded).
e. Number of references to this resource (generated, in this case it is 0).
f. **Initiate Rename Schema Element**: rename the schema element by entering the New Name and choosing the option whether to update the references to this element.

g. The remaining icons are **Go To** , **Accept Changes** , and **Delete** , which invoke the general editing tools.

4. **Add Reference Element**: This option adds a reference element that you can further define by the following:



a. The schema type definition or a new type definition.

b. Specifying the Minimum Occurs number (default is 0).

c. Selecting from the drop-down list the Maximum Occurs number (1 or unbounded.)

d. The remaining icons are **Go To** , **Accept Changes** , and **Delete** , which invoke the general editing tools.

5. **Add Attribute**: This option adds an attribute that you can further define by the following:

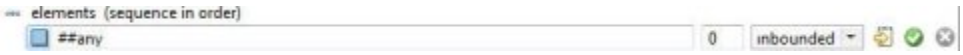
a. Choosing by the Primitive Types: String, Integer, Decimal, Boolean, Date&Time, Binary, URI or Any.

b. Choosing by the Primitive Sub Types: String, Normalized String, Token, Language, Name. NC-Name, Q-Name, Name Token, Name Tokens, ID, ID ref, ID refs, Entity, and Entities.

c. Use Optional/Required (default is Optional).

d. The remaining icons are **Go To** , **Accept Changes** , and **Delete** , which invoke the general editing tools.

6. Add Any Element: This option adds an element that you can further define by the following:



a. Wildcard Namespace (a space-delimited list of the namespaces can be entered).

b. Entering the Minimum Occurs number (default is 0).

c. Selecting from the drop-down list the Maximum Occurs number (1 or unbounded.)

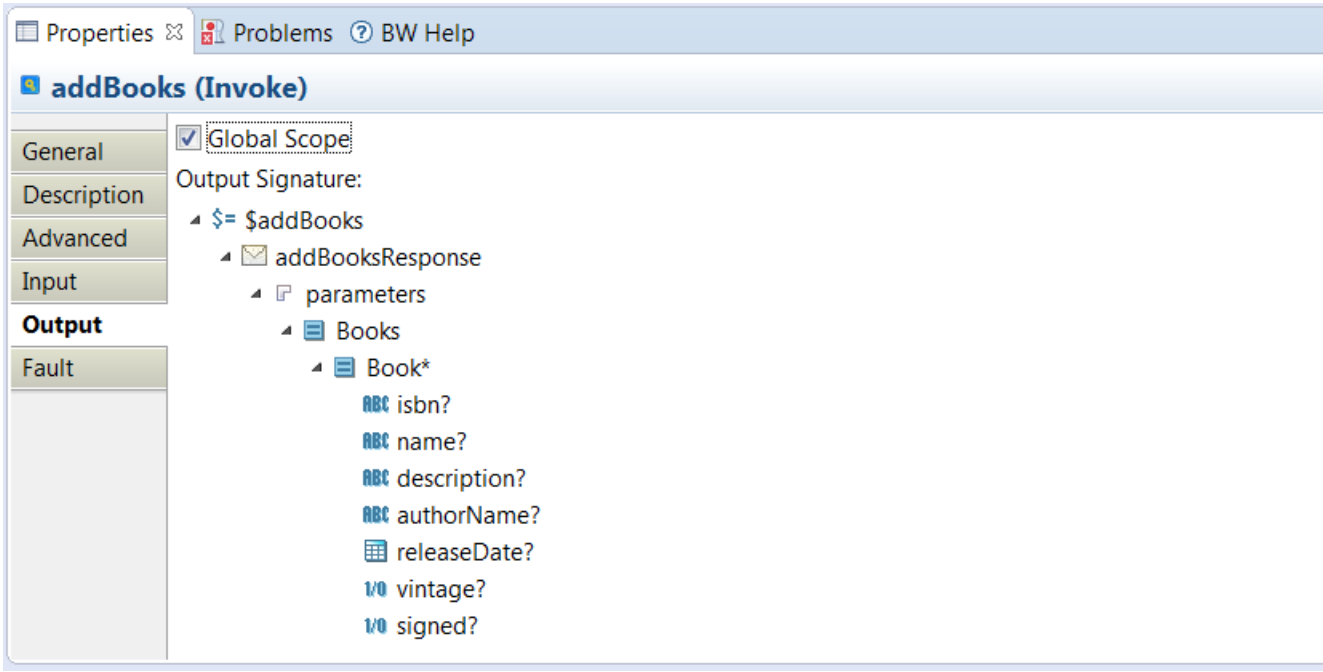
d. The remaining icons are **Go To** , **Accept Changes** , and **Delete** , which invoke the general editing tools.

Viewing the Output Tab

The **Output** tab is available in the Properties view and is used to display the activity output schema. The output of an activity is displayed for informational purposes only and cannot be modified or altered.

The output tab displays the activity output schema. This name appears in subsequent activities input tabs. The activity output data is displayed for informational purposes only and cannot be modified or altered.

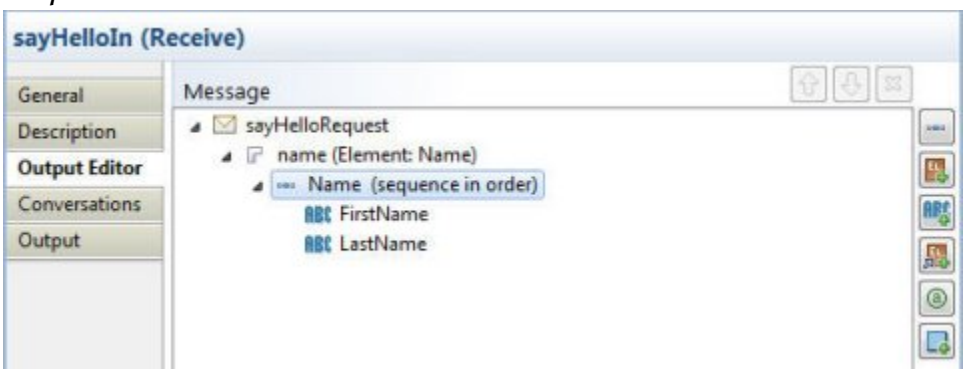
Output Tab



Configuring the Output Editor Tab

Input Editor allows for GUI based approach in configuring the output data.

Output Editor Tab



Using the icons on the right, additionally define the Name element. The icons have same meaning as when used for the Input Editor.

Importing WSDLs

Follow these steps to import WSDL files from the internet into TIBCO Business Studio Container Edition .

Procedure

1. Right-click the **Service Descriptors** folder, and select **Import** > **Import WSDL from URL**
2. Enter the URL of the WSDL in the **Resource URL** field.
   The **Import Location** field is automatically populated with the import location of the WSDL and XSD files being imported. By default, WSDL file are imported to the **Service Descriptors** folder and XSD files are imported to the **Schemas** folder. You can update these import locations in the wizard.

If you enter a remote WSDL URL, and the WSDL contains dependencies, these dependencies will be listed in the **Dependencies** section of the Import WSDL from URL wizard.

3. Optional. Unselect the **Update import location attribute and include location attribute to reference WSDL and XSD files imported locally** check box if you do not the import location to be automatically updated with the relative locations of corresponding and already imported WSDL and XSD files.

Using Additional Features

Complex business processes make use of additional features such as process scopes, fault handlers, checkpoints, and so on.

The following sections describe how to use the specified feature when developing a process.

Adding Scope Variables

A scope variable saves the state within the scope.

To add scope variables, select the scope in the Process Editor and then select the **Variables** tab in the Properties view.

Adding a Complex Type Variable

Click the icon  **Add complex type Variable** and select an existing schema or create a new schema to be added from the Select Schema Element Declaration dialog box.

Select Schema Element Declaration

Field/Action	Description
Workspace	When selected, the variable is valid only during the design-time.
Current and Dependent Modules	When selected, the variable is valid for the current module and the modules that are dependent on it.
Current Module	When selected, the variable is restricted to the current module.
Display all XSD Elements	Select the check box to display all the XSD elements in the module. This check box is selected by default.
Include Process Inline Schemas	Select the check box to display the process inline schemas in the module.
Include WSDL Inline Schemas	Select the check box to display the WSDL inline schemas in the module.

If you chose an existing schema, click **OK** to select it. If you choose to create a new schema, click **Create New Schema** to create a new XML schema.

Create XML Schema

Field/Action	Description
Resource Name	Specify a name for the new schema.
Workspace Location	Specify a location to store the new schema. The wizard displays the default location for the particular module. You can choose to keep the default or browse to select a different location.

Field/Action	Description
Choose a Root Element	Add a primitive element to the new schema using the icon **Add Primitive Element**  . The new primitive element will appear listed under the root element. Double-click the element to configure it. 
Primitive Types	Select the primitive type for the element from the drop-down list: • String • Integer • Decimal • Boolean • Date & Time • Binary • URI • Any
Subtypes	Select the subtypes for the element from the drop-down list: • String • Normalized String • Token • Language • Name • NC-Name • Q-Name • Name Token • Name Tokens • ID • ID ref • ID refs • Entity • Entities
Number of references to this resource	Displays the number of references to this resource.
Initiate Element Rename Refactoring	Use to rename the schema element. You can choose to preview and update all references to the element.

Field/Action	Description
Accept Changes	Accept the changes entered for the new schema element.
Cancel Changes	Cancel the changes accepted for the new schema element.
Remove Selected Element	Any of the element added to the schema can be deleted using this option.

Click **OK** when you are done editing the XML schema.

Adding a Simple Type Variable

Add a simple variable by clicking the icon **Add simple type Variable**. Select the variable type from the drop-down list and specify a default value.

Variable Type	Default Value
String	None.
Integer	1
Decimal	1
Boolean	true (You can select false from the drop-down list.)
Date & Time	None. Enter a date and time.
XSD Element	To select an XSD element, follow the instructions provided in Adding Scope Variables

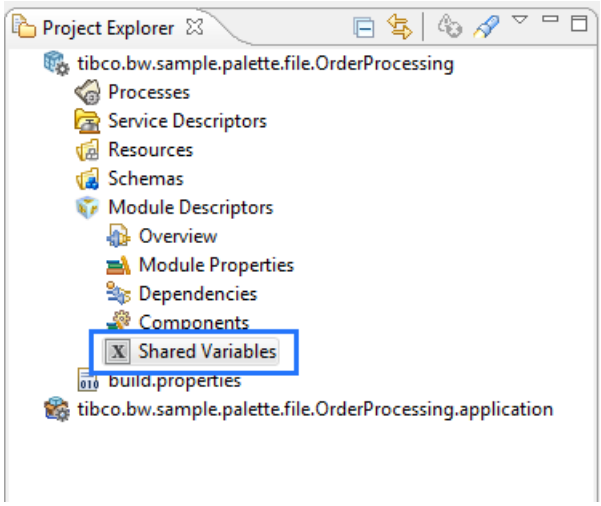
Defining and Using Shared Variables

Shared variables are defined at a module level.

Defining a Shared Variable

Procedure

1. In the Project Explorer view, double-click **Shared Variables** under the **Module Descriptors** to open the **Shared Variables** tab.



2. Click one of the following icons in the respective sections to define a module shared variable or a job shared variable:

   -  - Add a complex element. You can choose from an existing schema declaration or create a new schema.
   -  - Add a simple element.

3. In the Properties view, provide the information as described in the following table.

General Tab	Name	Name of the shared variable
	Type	Data type of the shared variable. Select one from the following options available in the drop-down list: • String • Integer • Boolean • Date&Time • Complex Element...
Description Tab	Description	Description for the shared variable.

Initial Value Tab	Initial Value	Enter an initial value for the shared variable. Select one from the following options: • None: Specifies that no initial value is set for the shared variable. Ensure that you set the value using the **Set Shared Variable** activity in the business process before you retrieve the value of the variable using the **Get Shared Variable** activity. • Select Value: Select this option to browse and select a file containing the initial value for the shared variable. • Build Value: Select this option to enter an initial value for the shared variable.

Retrieving and Assigning a Value of a Shared Variable

To retrieve the value of a shared variable, use the  **Get Shared Variable** activity in the General Activities palette. To assign a value to a shared variable, use the  **Set Shared Variable** activity in the General Activities palette.

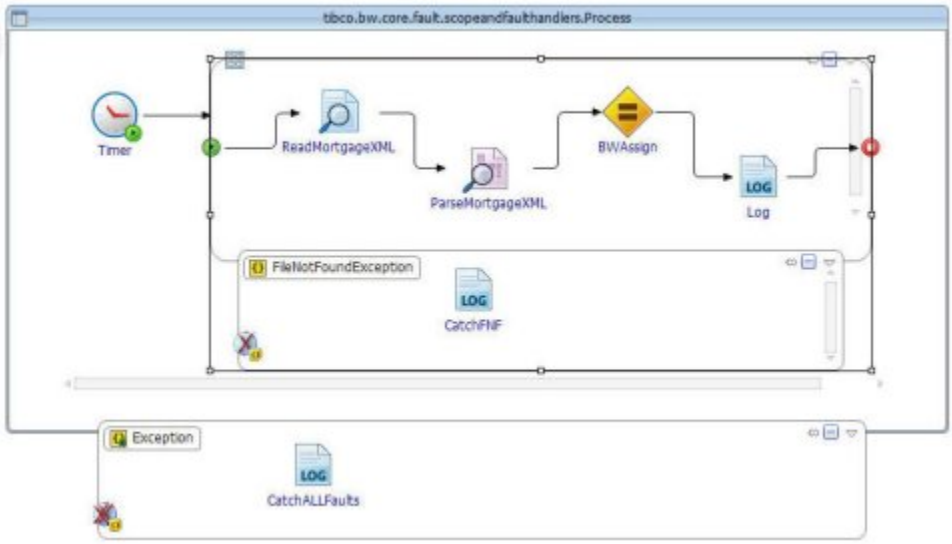
Using Fault Handlers

Fault handlers are used to catch faults or exceptions and create fault-handling procedures to deal with potential errors.

Fault handlers are defined at the scope level allowing you to catch faults or exceptions thrown by activities within a scope. There are two types of fault handlers: **Catch Specific Fault** and **Catch All Faults**.

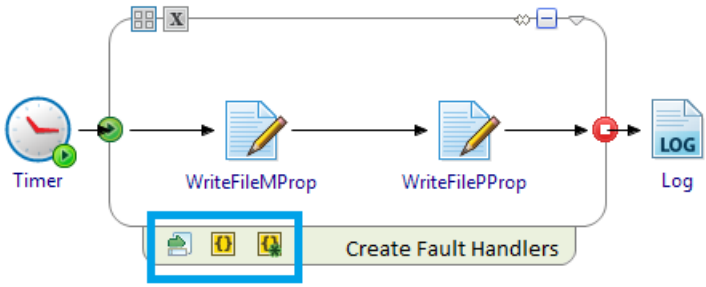
Fault handlers can be defined at the process level, or at a scope level within a process. The diagram below shows two fault handlers - one defined at the process level and the other defined at an inner scope level.

Fault Handler Attached to an Inner Scope



Procedure

1. Select the activities inside the process where the exception is expected to occur and select **Create Scope** > **Scope** from the right-click menu.
2. Move the cursor right underneath the scope's lower border to view the icons to create fault handlers.



3. Click on one the following:

   - Create Catch  to create a fault handler for a specific exception.
   - Create Catch All  to create a fault handler to catch all exceptions.

   A new fault handler is added under the scope.
4. Add activities and configure the fault handling procedure inside the fault handler area. For example, add a **Log** activity inside the fault handler area to record messages from the exception.

Password Obfuscator Utility

This utility is used to encrypt sensitive data such as passwords when configuring user defined services or environment variables.

See `<TIBCO_HOME>/bwce/version/bin/bwobfuscator` for more information.

XPATH

XML Path Language (XPath) is used to navigate through elements and attributes in an XML document. XPath uses path expressions to navigate through XML documents. XPath also has basic manipulation functions for strings, numbers, and booleans.

TIBCO BusinessWorks™ Container Edition uses XPath as the language for defining conditions and transformations.

For a complete description of XPath, refer to the XPath specification (from http://www.w3.org/). This section covers the basics of XPath and its use in the product.

XPath Basics

This product uses XPath (XML Path Language) to specify and process elements of data schema. These data schema are either process variables or input schema for an activity. You can also use XPath to perform basic manipulation and comparison of strings, numbers, and boolean.

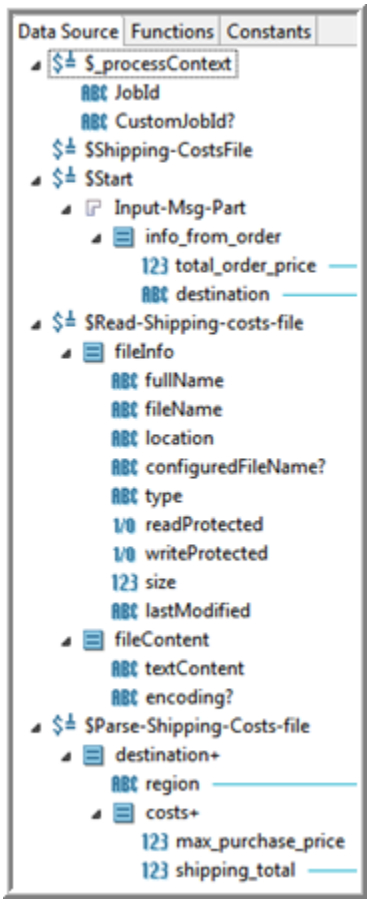
To use XPath in the product, you need to be familiar with the basic XPath concepts. However, to learn more about XPath when building complex expressions refer to the XPath specification from http://www.w3.org/

Addressing Schema Elements

All data source and activity input are represented as an XML schema. The data is represented as a schema tree regardless of where the data is derived from or its format. The data can either be simple (strings, numbers, boolean, and so on), or it can be a complex element. Complex elements are structures that contain other schema elements, either simple elements or other complex elements. Both simple and complex elements can also repeat. That is, they can be lists that store more than one element of the type specified.

XPath is used to specify which schema element you refer to. For example, the following schema might be available for an activity's input.

Schema Elements in Data Source



The data source area of the example **Input** tab shows the output schema of the activities in the process. There are two output schema, each a root node in the data source area: **`Read-Shipping-Costs-file`** and **`Parse-Shipping-Costs-file`**. Each of these schema has its own associated structure, for example, **`Read-Shipping-Costs-file`** has a set of simple values and **`Parse-Shipping-Costs-file`** has simple data and other complex data.

To reference a particular data item in any of these schema, start with the root node and then use slashes (/) to indicate a path to the desired data element. For example, if you want to specify the **`region`** attribute in the **`destination`** complex element that is in the Parse-Shipping-Costs-file node, use the following syntax:

```
$Parse-Shipping-Costs-file/destination[<< Filter >>]/region
```

The path starts with a dollar ($) sign to indicate it begins with a root node and continues with node names using slashes, like a file or directory structure, until reaching the desired location name.

Namespaces

Some schema elements need to be prefixed with their namespace. The namespace is automatically added to elements that require this element when creating mappings on the **Input** tab of an activity or when dragging and dropping data in the XPath builder.

Search Predicates

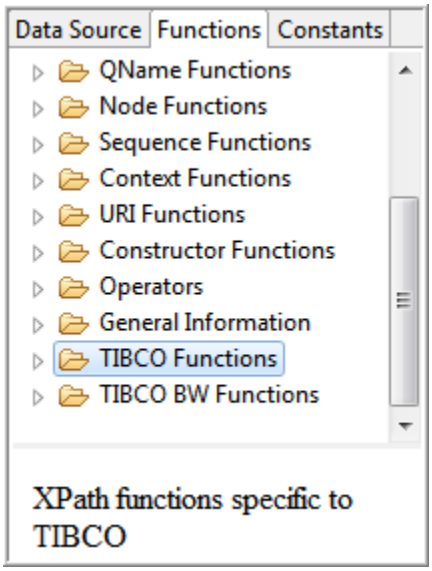
An XPath expression can have a search predicate. The search predicate is used to locate a specific element in a repeating schema element. For example, the $Parse-Shipping-Costs-file/destination/region item is a repeating element. To select only the first item in the repeating element, specify the following:

```
$Parse-Shipping-Costs-file/destination[1]
```

The [1] specifies the first element of a repeating item. Sub-items can also be examined and used in a search predicate. For example, to select an element whose destinationID is equal to "3A54", specify the following:

```
$Parse-Shipping-Costs-file/destination["3A54"]
```

See the online documentation available in the XPath Builder for a list of the available operators and functions in PATH.

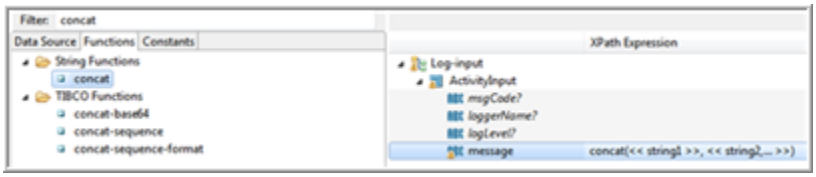


You can also use the Custom XPath Function Wizard to create your custom XPath function group. For more information, refer to **Creating Custom XPath Functions** topic in the *Bindings and Palettes Reference* guide.

XPath Expression

The XPath expression is used to creating transformations on the **Input** tab of any activity.

When the function is placed into the **XPath Expression**, placeholders are displayed for the function's parameters.

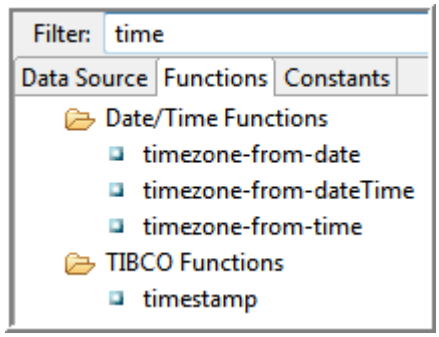


You can drag and drop schema elements from the **Data Source** tab into the function's placeholders.

XPath Builder Formula Elements

The following table shows the different elements of XPath Builder.

Elements	Description
Data Source	Displays the data source schema tree. All elements in this tree are available to drag and drop into the **XPath Expression** field.

Elements	Description
Functions	Displays the available XPath functions. These are categorized into groups and each function can be dragged from the function list into the **XPath Expression** field. When the function is placed into the **XPath Expression**, placeholders are displayed for the function's parameters. You can drag and drop schema elements from the **Data Source** tab into the function's placeholders. For more information about XPath functions, select XPath functions in XPath builder. The description of the function is displayed.
Filter	Use this field for a refined function search in the mapper. Clicking the **Functions** tab displays the **Filter** field. For example, type "time" in the **Filter** field to obtain consolidated results relating to "time" function. 

Elements	Description
Constants	Displays the constants available for use in XPath expressions. These are categorized into groups and each constant can be dragged from the constants list into the **XPath Expression** field. Constants are useful for inserting special characters, such as quotes, symbols, and so on, into XPath formulas. Constants are also defined for commonly used items, such as date/time formats. Constants can also be used for inserting the following **TIBCO BW Predefined Module Properties**. • **Activity Name** - returns the name of the activity on which the module property is set. • **Application Name** - returns the application name. • **Application Version** - returns the version of the application specified in the **Version** field under the **Overview** tab of the application. • **AppNode Name** - returns the name of the AppNode on which the application is deployed. • **AppSpace Name** - returns the name of the AppSpace on which the application is deployed. • **Deployment Unit Name** - returns the ID of the application specified in the **ID** field under the **Overview** tab of the application. • **Deployment Unit Type** - returns the deployment unit type as `application`. • **Deployment Unit Version** - returns the deployment unit version specified in the **Version** field under the **Overview** tab of the application. • **Domain Name** - returns the name of the domain in which the application is deployed. • **Module Name** - returns the name of the application module. • **Module Version** - returns the version of the module specified in the **Version** field under the **Overview** tab of the application module. • **Process Name** - returns the name of the process in which the module property is used. • **Process Stack** - returns the entire process path including the nested subprocesses, and the parent process. For example `main.Process/SubProcess1->sm.SubProcess1/SubProcess2->sm1.SubProcess2` • **Engine Name** - returns the name of the engine. By default, the name of the engine is `Main`. You can change the engine name by setting the property `bw.engine.name=Main` in the appspace `config.ini` file.

Elements	Description
Documentation Panel	Describes each selected function. On clicking a function in the **Function** tab, the documentation panel gives a brief description of the selected function with one or more examples.
XPath Expression	Displays the XPath formula you want to create. You can drag and drop items from the **Data Source** tab or the **Functions** tab to create the formula.

XPath Builder

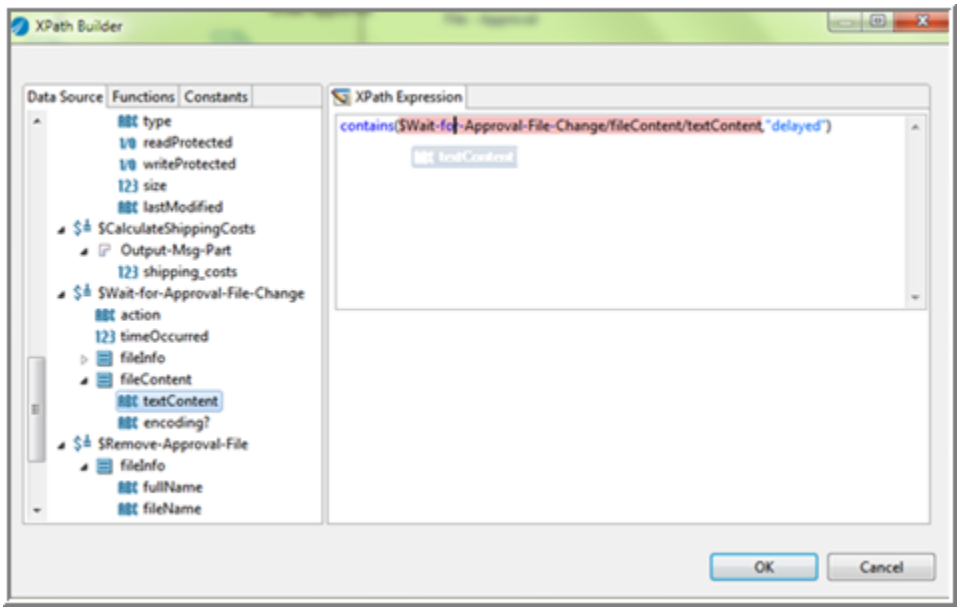
Using XPath Builder, you can drag and drop schema elements and XPath functions to create XPath expression.

Click the Transition in the process. In the **General** tab, select **Success with condition** option in the **Condition Type** field. This displays the **Expressions** field. Click icon to open the XPath Builder window.

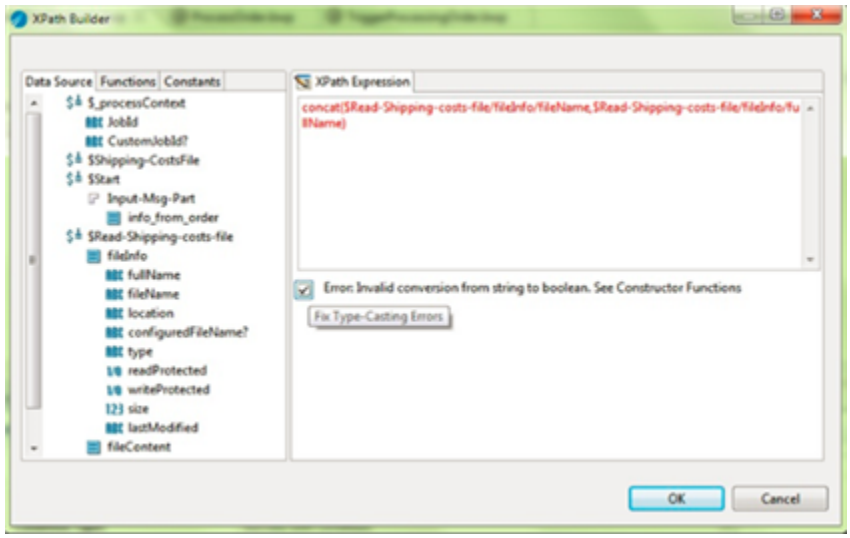
XPath Builder is also available from **Sequence Key** field and **Custom Job Id** field of all process starter activities (such as **Timer**, **File Poller**, and so on).

The following image shows how you can use XPath Builder to drag and drop schema elements into function placeholders.

XPath Builder



See the following image for the displayed result of evaluating the formula shown in the **XPath Expression** field. The errors in the formula are displayed here.



TIBCO BW Functions

XPath Builder can be used to fetch process related information for any activity. These functions are listed under the **TIBCO BW Functions** group.

- **getModuleProperty**: Returns the value of a module property. Also see **TIBCO BW Predefined Module Properties** under the **Constants** section.
- **getSystemProperty**: Returns the value of a Java system property.
- **restartedFromCheckpoint**: Returns `true` if the process instance recovered from a checkpoint, otherwise returns `false`.
- **generateEPR**: Returns an 'Endpoint Reference' as a string. This value can be used as an input to the **Set EPR** activity.
- **getHostName**: Returns host name of the host machine.

The XPath function `xsd:string()` saves double values in scientific notation if the double value has 7 or more digits before the decimal point. For example, if the value is 1000000.333, the `xsd:string()` function renders the value as 1.000000333E6.

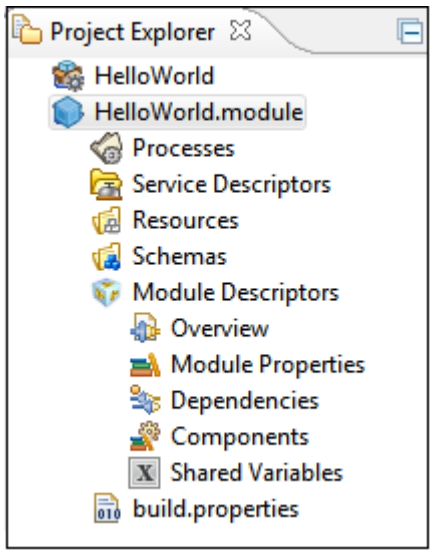
Developing a SOAP Service

A SOAP service makes a Process service available as a SOAP web service. You can achieve this by applying a SOAP service binding on the target process service.

Implementing a SOAP Service Provider

Procedure

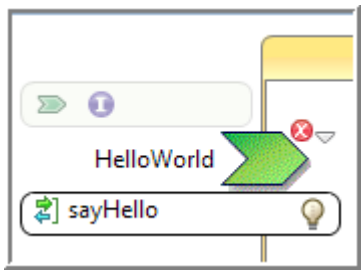
1. Click on the process package, for example, "HelloWorld", and then click on the **Create a new Business Works Process** icon.



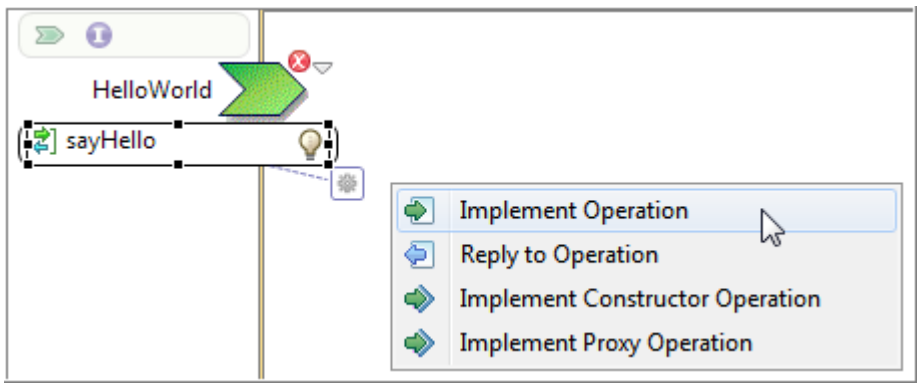
2. Select a process on which you want to add a service, and click the **Create Service** icon.

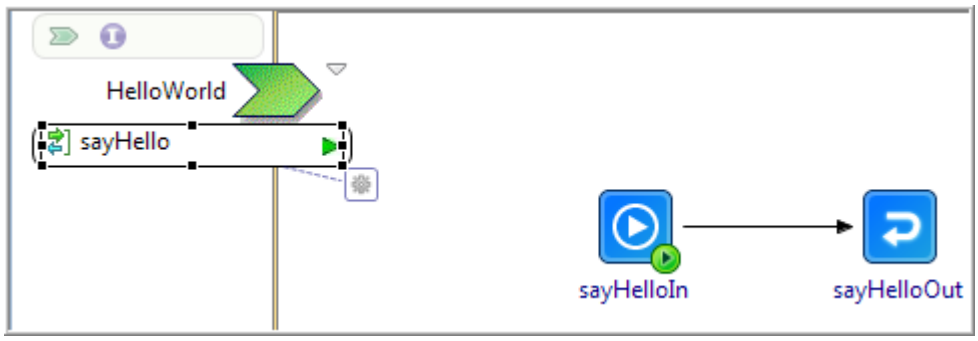
   The New Service dialog opens.

3. In the New Interface section specify the **Interface Name** as `HelloWorld` and **Operation Name** as `sayHello`. Click **Finish**.



4. To implement the operation, drag and drop the `sayhello` operation, and select **Implement Operation**.

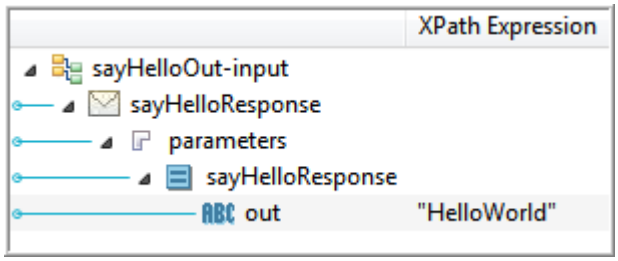




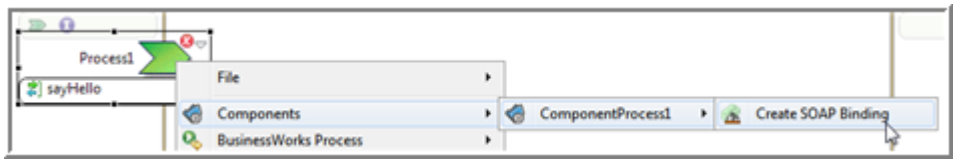
Choose **Implement Constructor Operation** option, if there are multiple operations in a Port type.

The option **Implement Operation** implements a single operation and creates a single Receive activity and a Reply. The option **Implement Constructor Operation** implements a constructor. A constructor provides for multiple operations. Use this option if the PortType has multiple operations which must be implemented by this process.

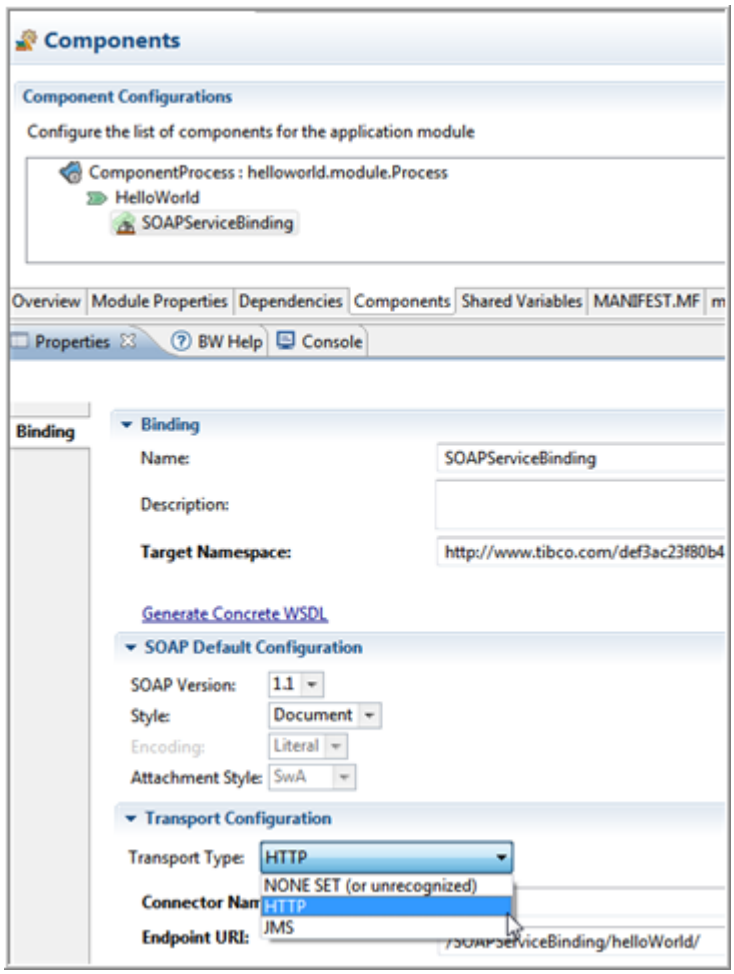
5. Click on the **Reply** activity (**`sayHelloOut`**) and under the Properties view, click the **Input** tab. Configure Reply message.



6. Right-click on green chevron and select **Components** > **ComponentsProcess** > **Create SOAP Binding**. The Binding Configuration dialog displays.



7. To configure transport on the SOAPServiceBinding, select **HTTP** from the **Transport Type** drop-down list in **Transport Configuration**.



8. Click on **Create Shared Resource** button and click **Finish** on the Create HttpConnResource Resource Template.

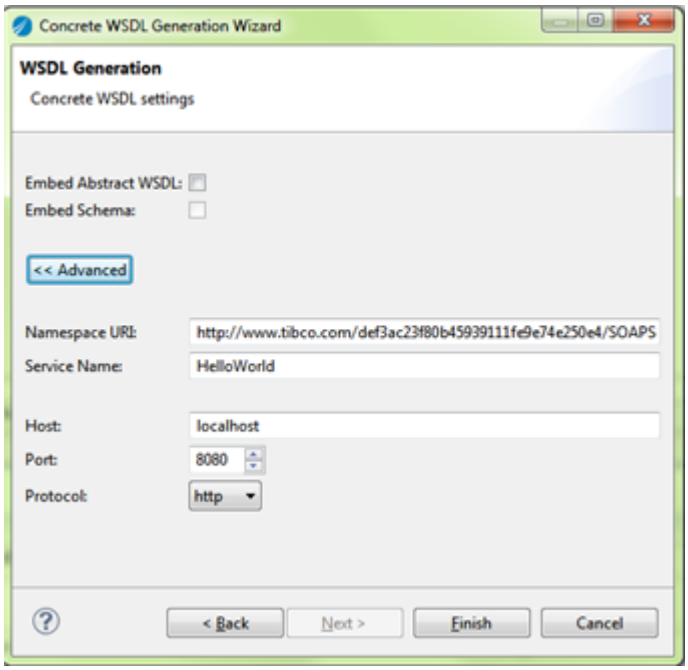
   The default port used by this shared resource is 8080. The service binding is now created.

9. To generate the concrete WSDL of the SOAP service created in the above steps, click **Generate Concrete WSDL** link.

10. Click **Workspace**. In the Folder Selection window and select the **Service Descriptor** folder of the current module and click **OK**.
    The Generate Concrete WSDL screen will now show the specified location and the name of the WSDL.

To create the Concrete WSDL in a desired location other than the workspace location, specify it by using **File System** button and click **Finish**.

11. To avoid namespace resolution error, click **Next** and clear the **Embed Abstract WSDL** and **Embed Schema** check boxes and click **Finish**.



Click on the **Advanced** tab to override the **Namespace URI**, **Service Name**, **Host**, **Port**, and **Protocol** fields.

The concrete WSDL is generated at the specified location.

Developing a RESTful Service

Services are used to invoke a process and to call out of the process so that a process receives data from a service and routes data to a service.

The key abstraction of information in REST is a resource. REST ignores the details of component implementation and protocol details. TIBCO BusinessWorks™ Container Edition currently allows the following HTTP methods to be performed on resources: GET, PUT, DELETE, and POST. Both XML and JSON are supported as data serialization formats along with support for definition of custom status codes, path(URL) parameters, key-value parameters, query parameters, and custom HTTP headers.

General Restrictions

- No wildcards or attribute wildcards. For example, `any` element and `any` attribute is not supported.
- Complex types may not contain both an attribute and a child element with the same local name.
- Complex types may not be of the pattern "simple type plus attributes".
- Complex types may not contain mixed content.
- Attributes that are not part of the default(empty) namespace, cannot be used for Complex Elements.
- The 'choice' and 'sequence' compositors may not have `maxOccurs > 1` (same as the restriction on 'all' in the schema specification).
- Substitution groups are not supported.
- Element of simple type with an attribute is not supported.
- The **`elementFormDefault`** can only be qualified for schemas used by REST binding and JSON activities.
- Schemas should not contain cyclic dependencies on other schemas
- Schemas should not have a type that has two child members with the same local name, but different namespaces.

Restrictions on JSON

- Arrays must have homogeneous content.
- Arrays cannot directly contain arrays ([[...], [...]])

Not currently supported

- Binary content in JSON as a special case

Implementing a REST Service Provider

A REST service provider exposes the resources in a process definition that can be invoked by clients using one of the following operations- POST, GET, PUT, PATCH, and DELETE.

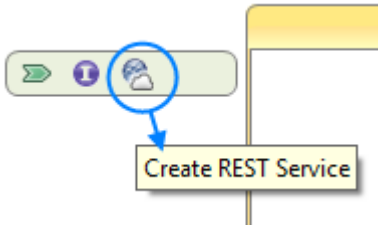
Prerequisites

If a schema definition does not exist, create (or import) a schema definition in the process to which you want to add the REST service.

Procedure

1. In the Project Explorer, select the process to which you want to add the REST service. There are multiple ways to invoke the wizard to create a REST service.

- From the main menu, select **File** > **New** > **BusinessWorks Resources** > **BusinessWorks REST Resource**.
- Right-click the menu, select **New** > **BusinessWorks REST Resource**.
- Click on **Create REST Service** in the process editor area. (Note that REST services can only be created in stateless BusinessWorks processes.)

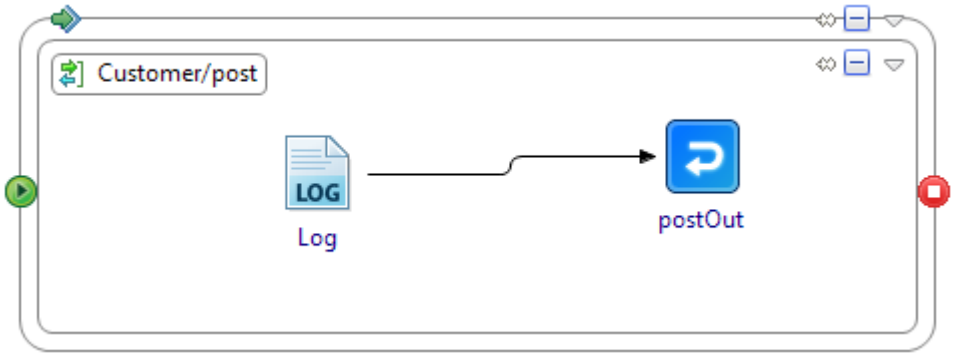


For more information, see the *REST Binding* in the *Binding and Palettes Reference* guide.

2. In the Create a New REST Service wizard, configure the REST service implementation by specifying the values for Resource Service Path, Type of Resource, Operations, and Implementation Data.
   - **`Summary`** about the new REST service.
   - **`Resource Service Path`**: Specifies the URI that is used to access the REST service.
   - **`Type of Resource`**: Select if the service works on a single resource or a collection.
   - **`Operations`**: By default, the GET operation is selected. Select or deselect the operations as needed.
   - **`Resource Schema`**: Select a resource schema for the REST service, if needed.
   - **`Implementation Data`**: Choose between structured and opaque implementation data.
3. Optionally, click **Next** to configure the selected operations individually to specify the nickname for the operation (default nickname is of the format *<operation><resource_name>*), summary, and the request and response elements and their data types.
4. Click **Finish**.
   The wizard adds the REST service and the selected operations, and also creates a process definition with the multiple operations.

   The REST service always implements the constructor operator.

5. Add activities to the process and configure them appropriately. For example, update the POST process to add a **Log** activity to log the requests and connect the postOut activity to **Log** activity.



6. Configure the input and output properties for the activities. For example, select postOut activity and then select **Properties** > **Input**. Expand the data tree in the **Data Source** tab and map the post

element from the left to the post Response element on the right to echo the element. Similarly, for **Log** activity, map the post element on the left to the ActivityInput message element on the right.

7. Save your changes.

Result

The REST service is built and can be tested using the built-in tester Swagger UI. For more information on Swagger UI, see *Testing the REST Service* in the *Getting Started* guide.

Developing Java Applications

The enhanced Java development tooling in TIBCO Business Studio™ Container Edition can be used to develop and debug the Java code. Using the software, you can develop applications graphically (without coding), use existing Java classes, or write custom Java code.

Adding Java-Specific Behavior to Projects

Eclipse projects use the **`project nature`** definition to tag a project as a specific kind of project. By configuring a project to use the **`Java nature`**, you can leverage on the enhanced Java development tooling available in TIBCO Business Studio™ Container Edition to develop Java application. A project with Java nature contains a default source folder for Java classes, `src`, in addition to other folders.

You can choose a different source folder by configuring the specified folder as the source folder and including the folder in the build path.

You can specify the project nature for an application module in one of the following ways:

- When creating a new application module, select the **Use Java configuration** check box.
- For an existing application module, right-click the project name in the Project Explorer view and select **Configure** > **Convert to Java project**.

Accessing Java Classes or Libraries from an TIBCO BusinessWorks™ Container Edition Application

An TIBCO BusinessWorks™ Container Edition application can invoke Java classes or reference libraries containing the Java code, using activities from the **Java** palette. Depending on the use case, the Java classes or libraries can reside in one of the following locations:

- Within the same application module as the TIBCO BusinessWorks™ Container Edition process: when the Java code need not be accessible from other applications, include the Java class within the same application module. See Using a Simple Java Invoke Activity for details.
- In a shared module or Eclipse plug-in project : when the Java code must be shared by multiple applications, use a shared module with Java nature or an Eclipse plug-in project to contain the Java code.
- External to the TIBCO BusinessWorks™ Container Edition application: when you do not have access to the Java source files and only the Java classes are available, you can invoke the Java methods stored in JAR files.

Using a Simple Java Invoke Activity

The **Java Invoke** activity can invoke a Java method from a class that resides in the same application module, a shared module or an eclipse Plug-in project.

Prerequisites

The project must be configured with Java nature.

See *Adding Java Nature to a Project* in the *Bindings and Palette Reference*.

Procedure

1. In the Project Explorer view, expand the application module project and right-click the Java source folder, `src` (default), and select **New** > **Class**.

2. In the New Java Class wizard, specify the package name and name of the Java class, and click **Finish** to create the Java class in the specified package. For example, type `com.tibco.myjavapackage` for the package name and `HelloWorld` for the class name.
3. Add one or more methods to the class. For example, add a static method, `sayHello`, which echoes a message "`Hello World!`" when invoked.

```
public static String sayHello(String input){
}
```

You can invoke static or non-static methods using **Java Invoke** activity. See the *Bindings and Palettes Reference* guide for details about the **Java Invoke** activity.

4. Add the implementation for the methods. For example, add the following implementation code to the sayHello method as shown:

```
public static String sayHello(String input){
    return "Hello " + input;
}
```

After implementing Java methods, you can proceed to design the process in the Process Editor.

5. Open the process in the Process Editor where you want to invoke the Java method and add a **Java Invoke** activity from **Java Palette**. Add transitions to the activity as required.
6. Configure the **Java Invoke** activity from the Properties view of the activity as described.
   - Click **Browse** in front of the **Class Name** field. In the Class Selection dialog, type the first few letters of the class name to search for the class you want to access. From the list of matching items, select the class you want to access. For example, select `HelloWorld`. Click **OK**.
   - From the drop-down list, select the method you want to invoke. For example, select `sayHello`.
   - If the method requires input parameters, provide the values for the input parameters from the **Input** tab of **Java Invoke** activity. For example, in the sayHello method, add the string "`World!`" to the input parameter.
7. Complete configuring your process and map the inputs for the activities as required. Then save the process. You can run or debug the application module in TIBCO Business Studio™ and verify the output of the **Java Invoke** activity.

Accessing Module Properties from Java Global Instance

You can access module properties from Java Global Instance so that at the time of deployment, these properties can be configured.
To access the TIBCO BusinessWorks™ Container Edition Module Properties in a user-defined Java code referenced in Java Global Instance, follow these steps:

Procedure

1. In the TIBCO BusinessWorks™ Container Edition module, specify a dependency on the package "**com.tibco.bw.palette.shared.java**" using **Import-Package**.
   a) Double-click **Dependencies** located under **BusinessWorks Container Edition Module** > **Module Descriptors**. This opens **BW Manifest Editor**.
   b) In the **Imported Packages** section, click the **Add** tab to add the dependency on the **com.tibco.bw.palette.shared.java** package.
2. Add the **@ModuleProperties** annotation to the method that accepts only one parameter of type **java.lang.HashMap**.
   Through this HashMap you can access the name or value pair of BusinessWorks Container Edition module properties.

Accessing Module Properties from Java Invoke Activity

You can access the TIBCO BusinessWorks™ Container Edition module properties and Java system properties from the user-defined code invoked from the Java Invoke activity and Java Event Source.

Procedure

1. Under the TIBCO BusinessWorks™ Container Edition module, click **Module Descriptors** > **Descriptors**.
   This opens BW Manifest Editor.
2. In the **Imported Packages** section, click **Add**.
   The Package Selection dialog opens.
3. Select the **com.tibco.bw.palette.shared.java** and **com.tibco.bw.runtime** package and click **OK**.
4. Add the @BWActivityContext annotation to the method which accepts only one parameter of type **com.tibco.bw.runtime.ActivityContext**.

   The module property can be accessed from ActivityContext class using the methods "registerModuleProperty" and "getModuleProperty". Refer to *API Reference* for more details on how to use these methods.

Accessing Module Properties in User-Defined Java Code Referenced in JavaProcessStarter

Procedure

- Retrieve EventSourceContext from the getEventSourceContext() method of abstract Java class "JavaProcessStarter".

  The module property can be accessed from EventSourceContext class using the methods "registerModuleProperty" and "getModuleProperty".

Creating an Application

The New BusinessWorks Application wizard helps create an application. There are multiple ways to launch the wizard:

- From the main menu, select **File** > **New** > **BusinessWorks Resources** and then select **BusinessWorks Application**.
- From the **Module Descriptors** > **Overview** getting started area, click **Create a BusinessWorks Application**.
- Right-click in the Project Explorer view and select **New** > **BusinessWorks Application**.

Specify the values for the following fields in the wizard:

1. **Project name**: Name of the application.
2. **Use default location**: Specifies the location on disk to store the application's data files. By default, this value is set to the workspace. To change, clear the check box and browse to select the location to be used.
3. **Version**: Version of the application.
4. **Create Application Module**: Selected by default to create an application module with the specified name. Clear the check box if you do not want to create an application module.
5. Click **Finish**.

Result

An application with the specified name is created and opened in the workbench. If the option to create an application module was selected, the application module with the specified name is also created.

Generating Deployment Artifacts

A deployment artifact is an archive file that contains all the information required to deploy the application to runtime. It is the only artifact that is handed from the design phase to the run time as it contains all the bundles and metadata that is required to deploy and run the application.

If any further changes to the design or configurations are made, the deployment artifact (archive file) must be regenerated.

When creating an archive file for an application, the application packager also generates the TIBCO BusinessWorks™ Container Edition processes in SVG format.

There are multiple ways to create a deployment artifact:

- From the Project Explorer view in TIBCO Business Studio™ Container Edition, open **Project.application** > **Overview** and click **Export Application for Deployment** link. In the EAR Export window, specify the location for the archive file and provide a custom name to the archive file, if needed, by clearing the **Use Default EAR file name** check box. Click **Finish** to create the deployment artifact (archive file).
- By selecting the project application in the Project Explorer and dropping it in the File Explorer an archive file for the application is created. If needed, change the default location in the File Explorer

  by using the  **Open Directory to Browse** option in the File Explorer and select a custom folder. For example `c:/tmp`.

When you deploy an application, each application in an AppSpace is identified by its unique name and a *major.minor* version number. The version number is important as it provides traceability and helps troubleshoot in case of an error at run time. If any further modifications are made to the application, the archive file must be regenerated with an updated version number and then deployed to Cloud Foundry.

Refactoring a Shared Resource or Policy Package

Follow these steps to refactor a resource or policy.

Renaming a Resource or a Policy Package

Follow these steps to change the name of a resource, or a policy, package and to update its corresponding references in the project.

Procedure

1. To rename a resource package, right-click on the package under the Resources folder and select **Refactor > Rename Resource Package**. To rename a policy package, expand the Policies folder, right-click on the policy package, and select **Refactor > Rename Policy Package**.
2. Enter a new name for the resource, or policy, package and select **OK**.
3. Optional. In the Rename Package Name dialog, confirm the changes that will happen, and the resource references that will be updated, by ensuring the correct resources are selected.
4. Select **OK**.

Changing the Location of a Resource or a Policy

Follow these steps to change the location of a resource, or a policy, and to update its corresponding references in the project.

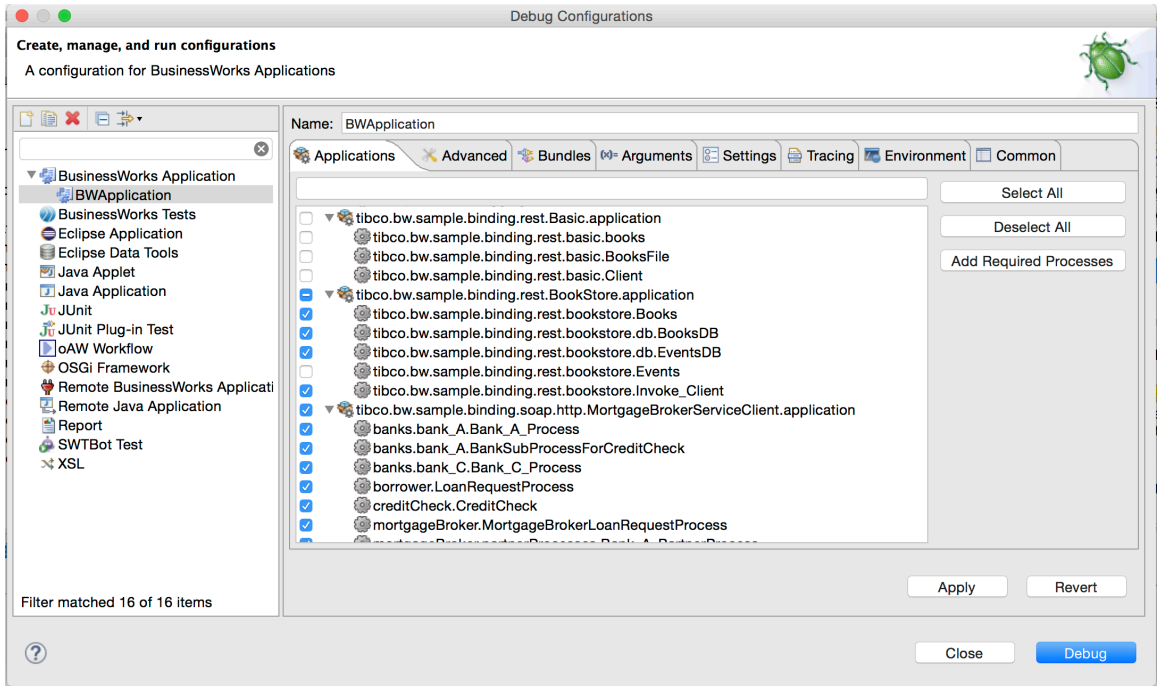
Procedure

1. To update the location of a resource, right-click on the package under the Resources folder and select **Refactor > Rename Resource Package**. To update the location a policy, expand the Policies folder, right-click on the package containing the policy you want to rename, and select **Refactor > Rename Policy Package**.
2. Enter a new location for the resource, or policy, and select **OK**. For example, to change the location of a Basic Authentication policy residing in **refactoringproject.TestPackage**, modify the package name to **refactoringproject.NewPackage.TestPackage**.
3. Optional. In the Rename Package Name dialog, confirm the changes that will happen, and the resource references that will be updated, by ensuring the correct resources are selected.
4. Select **OK** .
   A new folder structure under the Resources, or Policies, folder is created, and the resource is moved to the newly created location.

Using the Debugger

The debugger enables different configurations of an application to be run in design phase.

You can select applications, and processes in an application, to launch in the debugger.



The Debug perspective consists of set of views which are related to the debugging task. Some views, for example the Project Explorer view, are not available in the Debug perspective, while others, such as Debug and Breakpoints, are available in the Debug perspective. There are multiple ways to open the Debug perspective:

- From the main menu, select **Window** > **Open Perspective** > **Other** and then select **Debug**.
- From the **Module Descriptors** > **Overview** Testing area, click **Launch BusinessWorks Debugger**.

The Debug perspective consists of the following views, starting from the upper left corner clockwise:

- **Debug**: Shows the list of debug launches and allows you to manage them using the icon bar as follows:
  - Remove All Terminated Launches
  - Resume
  - Suspend
  - Terminate
  - Disconnect
  - Step Into, Step Over, Step Return, Drop to Frame
  - Use Step Filters

  - Remove Completed Process
- **BusinessWorks Jobs**: shows all running jobs and allows you some basic management such as, to Clear All Jobs .
- **Servers**: shows the servers that are available. You can also define a new server using the New Server Wizard, which allows you to define a new server as well as to download additional server adapters.
- **Variables**: shows the variables associated with the process being debugged. The main management tasks associated with the variables are:
  - Show Type Names
  - Show Logical Structure
- **Breakpoints**: shows the breakpoints used for debugging. The main management tasks associated with the breakpoints are:
  - Show Breakpoints Supported by Selected Target
  - Go to File for Breakpoint
  - Skip All Breakpoints
  - Link with Debug View
  - Add Java Exception Breakpoint
- **Job Data**: shows available information about the running process instances.
- **Process Launcher**: shows available sub-processes that can be launched. Inputs to the process instance can be provided in the process launcher.
- **Properties**: shows available information about the properties in the process being debugged.
- **Tasks**: shows all debugging tasks listed by their resource, path, location, and type.
- **Console**: gives the output of the debugging task.

Configuring the Debugger

You can use Debug configuration to create, manage, and run configurations in TIBCO Business Studio Container Edition .

There are multiple ways to access Debug Configurations window:

- From the menu **Run** > **Debug Configurations**.
- From the **Module Descriptors** > **Overview** Testing area, click **Launch BusinessWorks Debugger**.

Using the Create, manage, and run configurations dialog you can select the following:

- Applications to debug.
- Advanced configurations such as logging configuration and engine debug port.
- Arguments: program arguments such as the target operating system, target architecture, target web services, working directory, and so on, and VM arguments such as *TIBCO_HOME*, *port number*, or any engine properties that need to be set when running the application.
- Settings that define the Java Runtime Environment such as Java executable and runtime JRE, configuration area, and so on.

- Tracing criteria for the available OSGi bundles. By default, tracing is disabled. When enabled, you can choose among the available OSGi bundles, and then select the desired tracing criteria for each of them.
- Environment variables such as *PATH*, *LD_LIBRARY_PATH*, and so on.
- Common settings where you can save the configuration either as a local or a shared file and display them in the favorites menu (Debug and/or Run), define encoding for the files, and so on.

  After selecting the options, click **Apply** to apply the changes or **Debug** to launch the debugger with the selected debug configuration.

Testing an Application in TIBCO Business Studio™ Container Edition

Using TIBCO Business Studio™ Container Edition you can test your applications during design phase using the debugger.
The debugger provides the runtime environment to test your application in TIBCO Business Studio™ Container Edition by starting the TIBCO BusinessWorks™ Container Edition engine, domain (BWEclipseDomain), AppSpace (BWEclipseAppSpace), and AppNode (BWEclipseAppNode) from within TIBCO Business Studio™ Container Edition. When you run an application using the debugger, the Console view displays all messages when executing the application.

Procedure

1. Open the application module in TIBCO Business Studio™ Container Edition and select the component process in the Project Explorer.
   The selected process opens in the Process Editor.
2. From the menu, click **Run** > **Debug Configurations**.
3. In the Debug Configurations window, expand the tree under **BusinessWorks Application** and select **BWApplication**.
4. Click the **Applications** tab. If multiple applications are selected, click **Deselect All**. Then select the check box next to the application name you want to run.

   If needed, specify additional information such as engine properties in the debug configuration. See Configuring the Debugger for details.
5. Click **Debug** to run the application in Debug mode.
   The engine and the runtime entities such as domain (BWEclipseDomain), AppSpace (BWEclipseAppSpace), and AppNode (BWEclipseAppNode) are started and the selected application deploys. The Console view displays a log of the execution.
6. After completing the execution, click the **Terminate** icon to stop the process.

Unit Testing

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

Setting Up a Test File

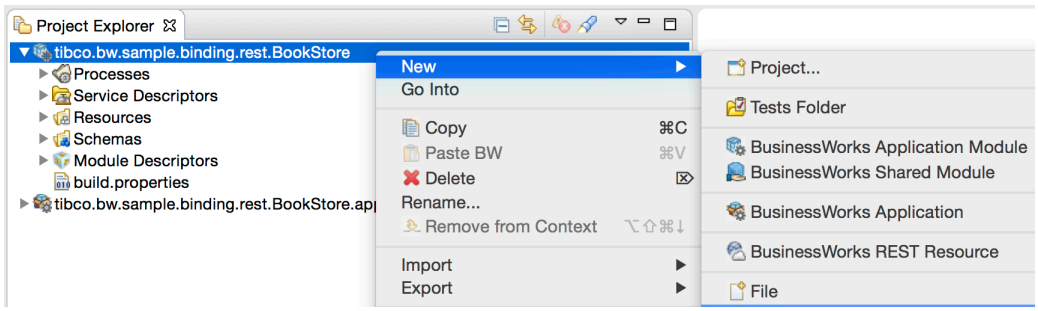
Set up a test file to create unit tests for processes.

Prerequisites

Before you create a new test file, your application module must contain a **Tests** folder. You only need to create this folder once, and it will hold all of the test files you create.

To create the **Tests** folder, follow these steps:

1. From the Project Explorer, right-click on the application module, and select **New** > **Tests Folder**.

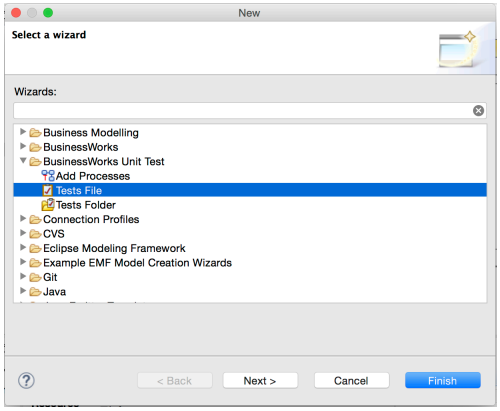


2. Follow the commands in the New Test Folder wizard, and click **Finish** to create the **Tests** folder.

Create a Test File

Once the **Tests** folder exists in your application module, you can create a new test file. There are multiple ways to create this file:

- Select the process you want to test, click the **Properties** tab, and select the **Tests** tab. Click the icon, and follow instructions in the wizard to create a new test file.
- Right-click on the **Tests** folder, and select **New** > **Other**. In the wizard, expand **BusinessWorks Unit Test** to select **Tests File**, and click **Next**. Complete instructions in the wizard, and click **Finish** to create a new test file.



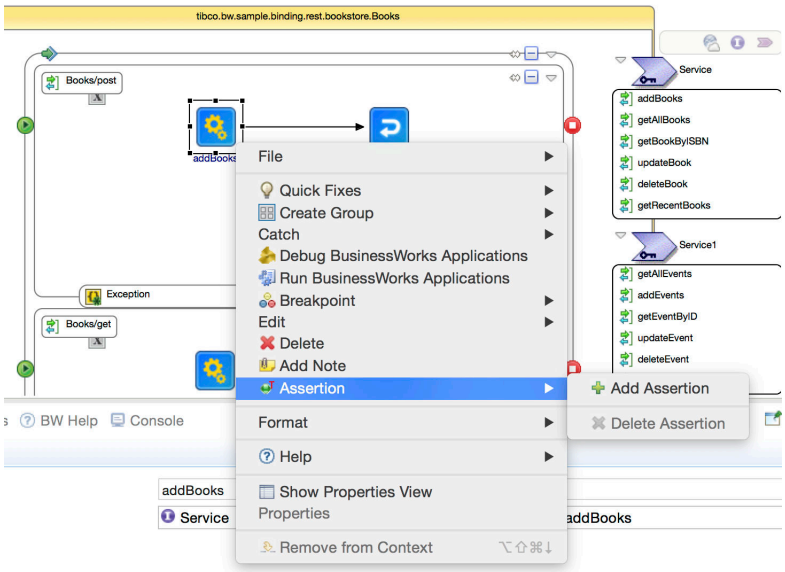
- In the Process Editor pane, right-click on the process title, and select **Process** > **Add Process to Test File**. If a test file has not been created, the New Test File wizard displays. If a test file has been created, the Add Process to Test File wizard displays. Follow the instructions in the wizards to add the process to a test file.

Add Assertions to the Test File

To add an assertion to the test file, apply it to an activity in the process.

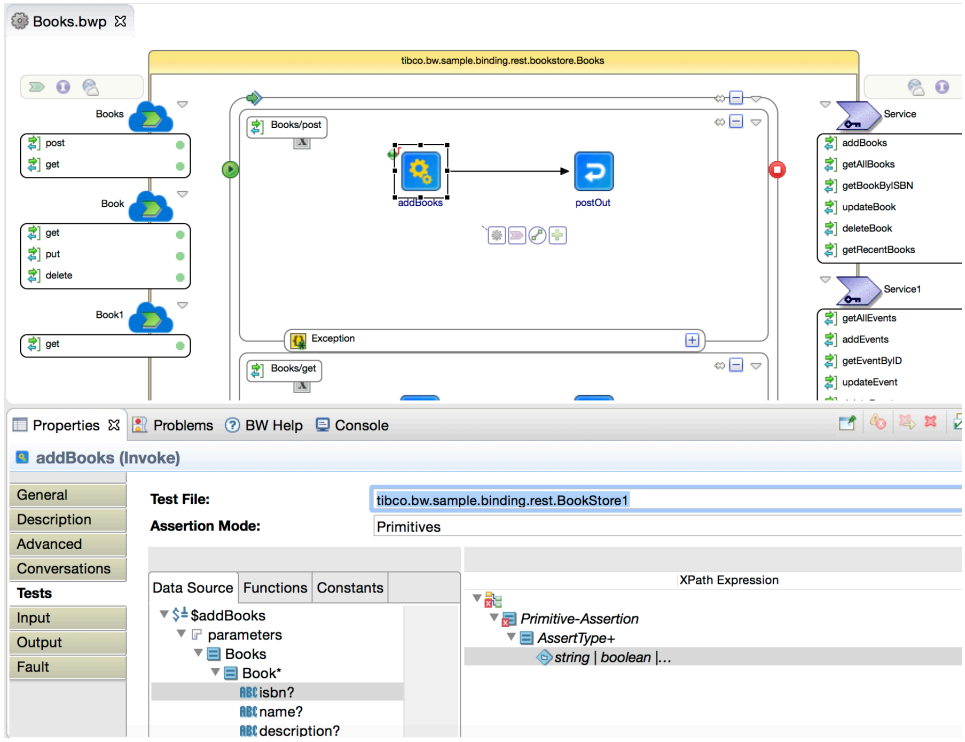
Procedure

1. Select the process.
2. Right-click the activity you want to add an assertion on, and select **Assertion** > **Add Assertion**. In this example, an assertion is being added to the **addBooks** activity in **books.bwp**.

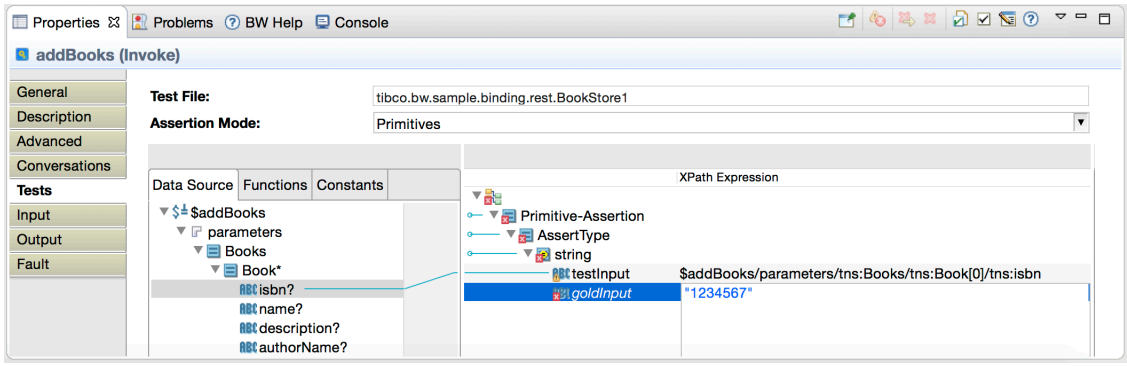


A green icon displays in the upper left corner of an activity with an assertion on to it.

3. From the **Tests** tab, go to the **Data Source** tab and expand output schema of the activity you are adding an assertion to.
4. Expand the **AssertType** attribute, and drag and drop the **string | boolean |** element to an element in the data source schema tree.



5. Select a datatype for the output value, and click **Finish**.
6. Drag and drop a schema element from the data source schema tree into the assertion elements into the **XPath Expression** field. The **`testinput`** element is the data that the process receives, and the **`goldInput`** element is the value you expect the process to receive. In other words, the **`goldInput`** is the statement, or the assertion, that you are making.
   In this example, the expected value for the**`isbn`** element on the **addBooks** activity is `"1234567`.



Adding a Process to a Test File

Multiple processes can be added to a test file.

Prerequisites

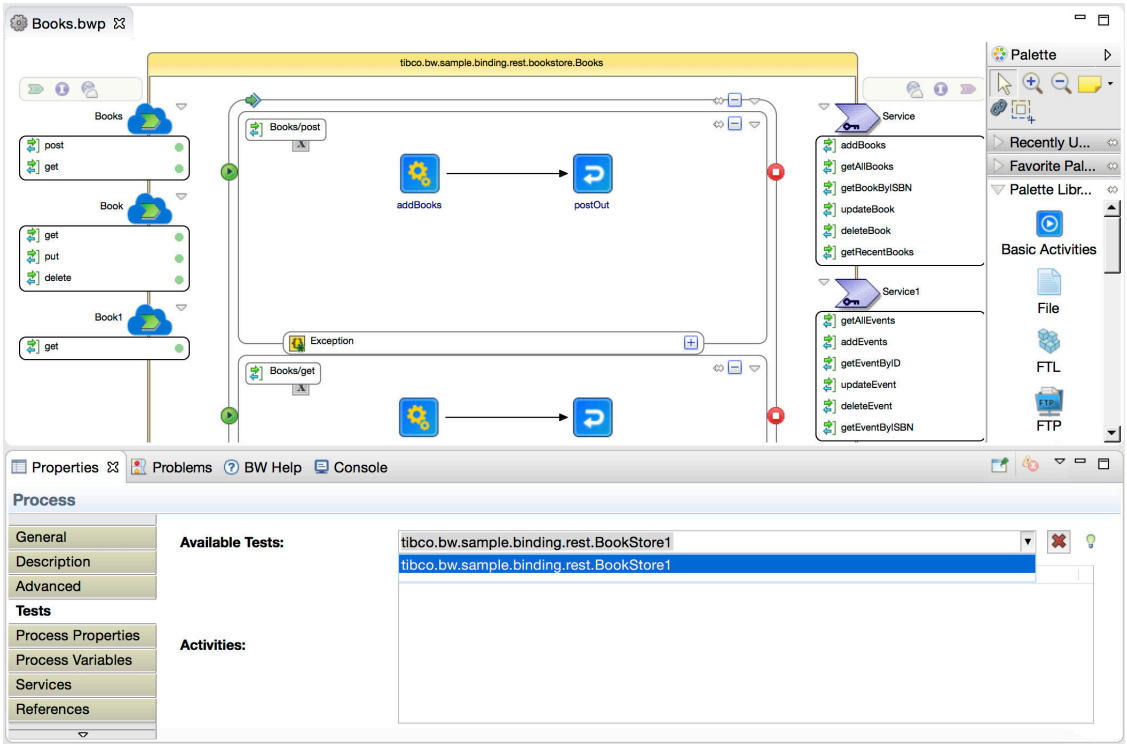
A **Tests** folder and a test file must be present in the application module.

There are two ways to add a process to a test file:

- When you create a new test file, there is a pane in the New Test File wizard that prompts you to select the process, or processes, that you want to add to the test file.

- Select the process you want to add to the test file, right-click anywhere in the **Process Editor** pane, and select **Process** > **Add Process to Test File**. Follow instructions in the wizard and click **Finish** once you are done.

After a process has been added to a test file, you can confirm this by checking the **Available Tests** field. To access this field, go to the **Properties** tab, select the **Tests** tab, and click the drop-down menu in the **Available Tests** field. In this example, selecting the drop-down menu for the **Available Test** field shows that the **books.bwp** process from the Bookstore sample has been added to the tibco.bw.sample.binding.rest.Bookstore1 test file.



Check the **Activities** field directly below to view activities with assertions. This field provides an overview of the assertions in the test file you selected.

Uploading and Deploying a Test File

Upload and deploy a test file with TIBCO Business Studio™ Container Edition or with REST APIs.

TIBCO Business Studio™ Container Edition

Prerequisites

The application process must be added to the test file you are deploying.

Procedure

1. Open the process you have added to the test file. For example, expand the application, and double click on the **books.bwp** process to open it in the process editor.
2. Start the debugger and launch the application.
   a) Click **Run > Debug Configurations**.

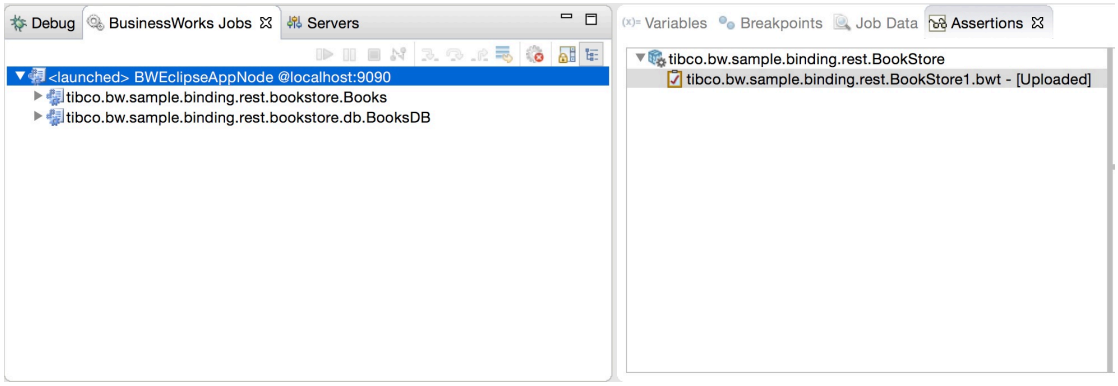
b) In the left-hand tree of the Debug Configuration wizard, expand **BusinessWorks Application** and select **BWApplication**.

c) Click the **Applications** tab and then click **Deselect All** if you have multiple applications, and select the application you want to debug. For example, select the check box next to tibco.bw.sample.binding.rest.BookStore.application.

d) Click **Debug**.

3. From the Debugger view, select the **Assertions** tab.

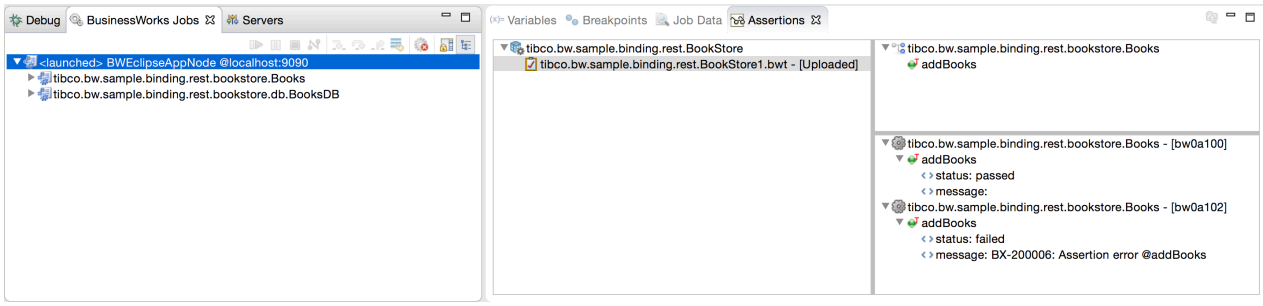
4. Click the test file that displays under the launched process, and click the icon to upload the test file to deploy the file to run time. After the test file has been uploaded, the file status changes to **[Uploaded]**. In this example, the tibco.bw.sample.binding.rest.BookStore1.bwt test file has been successfully uploaded.



5. Invoke the process.

6. From the **BusinessWorks Jobs** tab, confirm whether the activity that you have added an assertion to receives the expected output. If the activity did not receive the expected output, the **Faulted** status displays.

From the **Assertions** tab, click the test file to view asserted activities, and click the asserted activities to view assertion test results. For example, clicking the tibco.bw.samplebinding.rest.BookStore1.bwt test file from the **Assertions** tab, shows that an assertion has been added to the **addBooks** activity from the Books process in the Bookstore sample application. Clicking on **addBooks** displays the results returned from testing the asserted activity.



Using the bwdesign Utility

The **bwdesign** utility provides a command line interface for creating, validating, importing or exporting resources stored in a workspace.

To view arguments and options for a command, open a terminal, navigate to the `BW_HOME\bin` folder, and type **bwdesign help** *command* at the command line.

Command Name and Syntax	Description
cd SYNTAX: `cd [path]`	Changes the current working directory to the specified folder.
clear SYNTAX: `clear`	Clears the command line console.
execute SYNTAX: `execute [filename]`	Executes a batch script file containing a set of commands to execute in sequence.
ls SYNTAX: `ls [options]`	List the projects in current workspace or the files in current working directory.
pwd SYNTAX: `pwd`	Prints the location of the current working directory.
quit SYNTAX: `quit`	Exits the command line console.
system:create SYNTAX: `system:create [options] [outputfolder]`	Creates resource(s) in the workspace.
system:export SYNTAX: `system:export [options] [projects] [outputfolder]`	Exports BW artifacts from the specified projects in the workspace to a folder. The artifacts can be ZIP or EAR files.
system:import SYNTAX: `system:import [options] files`	Imports flat or ZIP projects into the current workspace.

Command Name and Syntax	Description
system:validate SYNTAX: `system:validate [options] [modules]`	Validates BW modules in the current workspace.
setedition SYNTAX: `system:setedition -name`	Converts projects from their existing editions to this edition of TIBCO Business Studio Container Edition . Select the option **-name**, and provide the names of the projects to be converted. Provide comma separated values to convert multiple projects. If the option **-name** is not selected this command sets the edition of all the projects in the workspace to the current edition of TIBCO Business Studio™ Container Edition .
exit SYNTAX: `exit`	Exits the command line console.

Procedure

1. To use the bwdesign utility, open a terminal and navigate to `BW_HOME\bin`.
2. Type: `bwdesign -data <TIBCO_BusinessStudio_workspace_absolutePath>`. For example, `bwdesign -data C:\myWorkspace`.

Messaging

Integrating with TIBCO FTL®

This topic describes how to add the TIBCO FTL 5.0 Native Libraries to the TIBCO BusinessWorks™ Container Edition runtime environment.

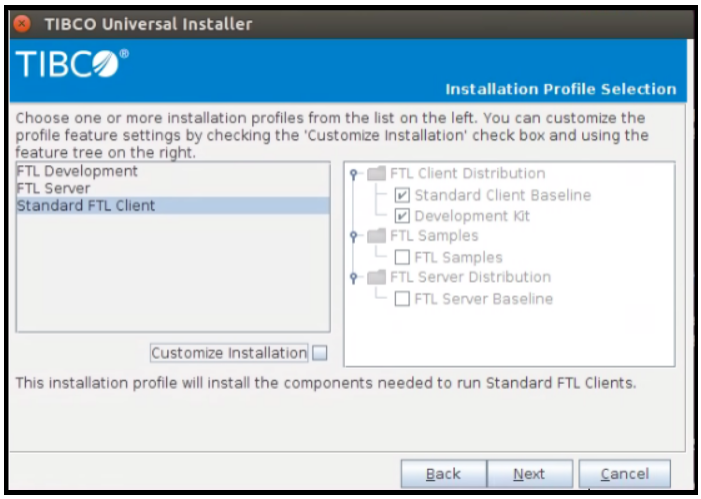
Designtime

1. Install TIBCO FTL 5.0 on your local machine.
2. Ensure that the `FTL_HOME` environment variable is set to your TIBCO FTL installation before launching TIBCO Business Studio™ Container Edition.

Runtime

To add TIBCO FTL 5.0 native libraries into TIBCO BusinessWorks™ Container Edition runtime environment:

1. Download TIBCO FTL 5.0 on a 64-bit Linux machine.
2. During the installation process select the **Standard FTL Client** profile as shown.



3. Create the TIBCO FTL client libraries.

   From a temporary folder execute:

   ```
   ls <FTL-TIBCO-HOME>/lib/*.so* | zip -j ftl_linux_client_libraries -@
   ```

4. Cloud Foundry only

   a. Extract the TIBCO BusinessWorks™ Container Edition buildpack into a temporary folder, *<buildpack-temp-folder>* and copy `<temp folder>/ftl_linux_client_libraries.zip` into the `<buildpack-temp-folder>/resources/addons/lib` folder.

   b. Zip the contents of the temporary folder and push the customized buildpack to your Cloud Foundry environment.

5. Docker only

   a. Build the TIBCO BusinessWorks™ Container Edition base Docker image. For more information refer to Creating the TIBCO BusinessWorks™ Container Edition Base Docker Image.

   b. From the temporary folder created in Step 3 above, use the Docker file given below to copy this zip file into the base Docker image:

   ```
   FROM tibco/bwce:latest
   COPY *.zip /resources/addons/lib
   ```

   c. From the temporary folder, build the Docker image:

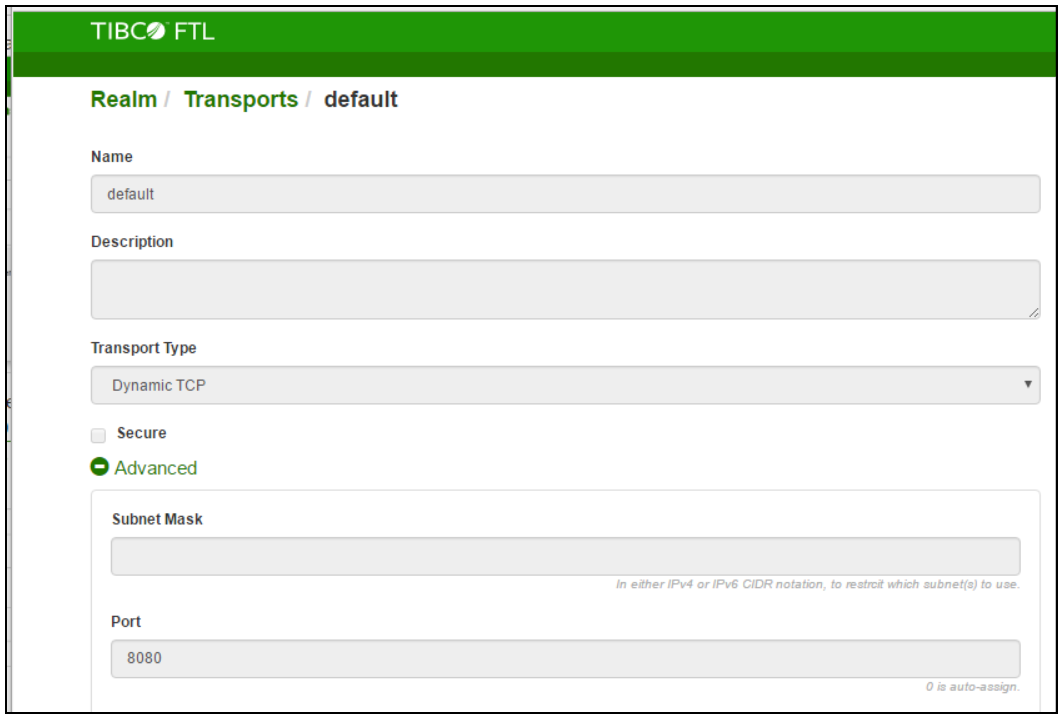
   ```
   docker build -t TAG-NAME .
   ```

   Alpine Linux is not supported.

Limitations

- Due to ephemeral nature of containers, only the **Dynamic TCP** transport type should be used on both Docker and Cloud Foundry environments.
- The following limitations are due to the single port restriction of Cloud Foundry:
  - Activities from the FTL palette should not be used along with the HTTP Receiver activity or with a REST service.

- An application should not contain more than one instance of the FTL Shared Resource.
- Port `8080` must be configured for **Dynamic TCP**.



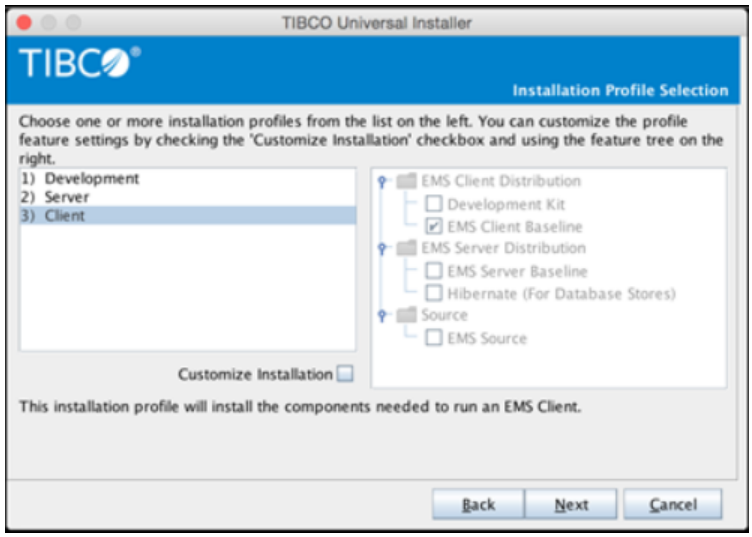
Integrating with TIBCO EMS

Prerequisites

Make sure that the TIBCO Enterprise Message Service™ client OSGi bundles are present in the *<EMS-HOME>*`/components/1.0/plugins` folder.

If you do not have these client libraries, install the TIBCO Enterprise Message Service™ libraries by selecting either the Development or Client installation profiles.

Alternatively, copy the *<EMS-HOME>*`/components` folder from an existing installation to your local machine.



Designtime

To install the TIBCO Enterprise Message Service™ Client libraries, run the following command from the *<BWCE_HOME>*`/bin` folder and follow the prompts:

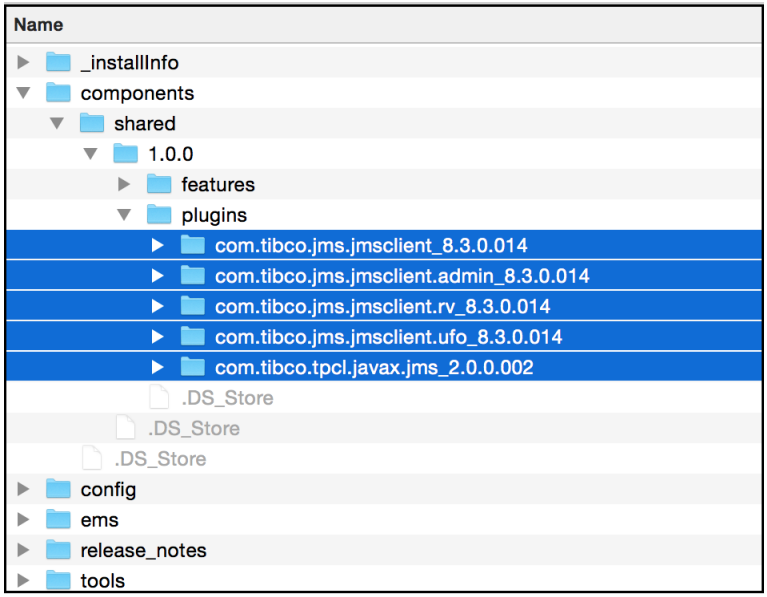
```
bwinstall ems-driver
```

Runtime

To add the TIBCO Enterprise Message Service™ client libraries to the TIBCO BusinessWorks™ Container Edition runtime environment:

Cloud Foundry

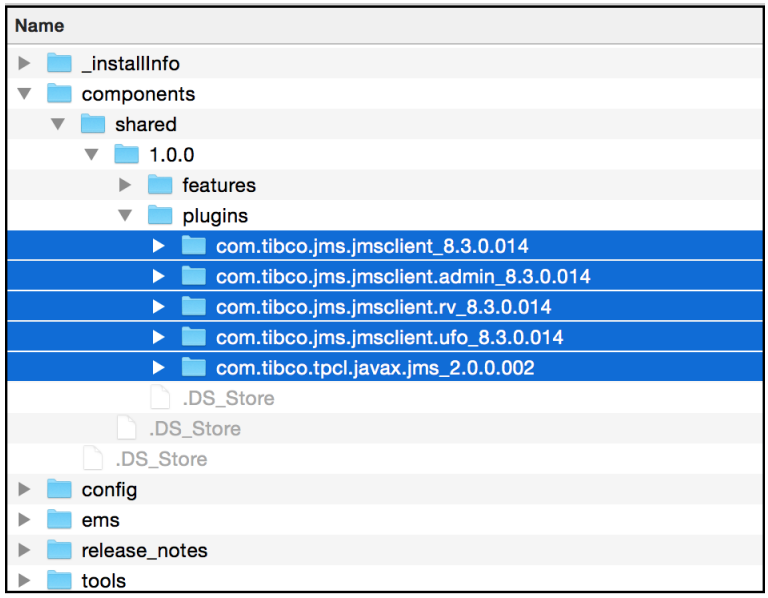
1. Extract the TIBCO BusinessWorks™ Container Edition buildpack to a temporary folder
2. Copy the TIBCO EMS OSGi client libraries from *<EMS-HOME>*`/components/1.0/plugins` into *<temporary folder>*`/resources/addons/jars`.



3. Zip the contents of the temporary folder and push the customized buildpack to your Cloud Foundry environment.

Docker

1. Build the TIBCO BusinessWorks™ Container Edition base Docker image. For more information, refer to Creating the TIBCO BusinessWorks™ Container Edition Base Docker Image.
2. Copy the TIBCO Enterprise Message Service™ OSGi client libraries from *<EMS-HOME>*`/components/1.0/plugins` into a temporary folder.



3. From the temporary folder use the Docker file given below to copy these jars into the base Docker image:

```
FROM tibco/bwce:latest
COPY . /resources/addons/jars
```

4. From the temporary folder, build the Docker image:

```
docker build -t TAG-NAME .
```

Application Development for Cloud Foundry

The following section provides information about system module properties and environment variables as they apply to TIBCO BusinessWorks™ Container Edition.

Switching the Container Platform

Procedure

1. In TIBCO Business Studio™ Container Edition, click **Window** > **Preferences** .
   On Mac OS X platform, click **TIBCO Business Studio Container Edition > Preferences**.
2. In the Preferences dialog click **BusinessWorks Container Edition** > **Container Platform**.
3. Choose either **Cloud Foundry** or **Docker**.
4. Click **Apply** and then **OK**.

What to do next

TIBCO Business Studio™ Container Edition has to be restarted for the changes to take effect.

It is recommended that you clean your workspace after TIBCO Business Studio™ Container Edition is restarted.

This option can be accessed from the TIBCO Business Studio™ Container Edition menu **Project** > **Clean**.

The TIBCO BusinessWorks™ Container Edition Buildpack

The TIBCO BusinessWorks™ Container Edition buildpack must be uploaded in your Cloud Foundry environment before deploying any applications.

Customize the Buildpack

This buildpack can be customized for the supported third-party drivers, OSGI bundles, integration with application configuration management systems, and application certificate management.

Customization of the buildpack is not supported on the Windows platform.

Procedure

1. Download the TIBCO BusinessWorks™ Container Edition build pack zip file from http://edelivery.tibco.com.
   To download this file:
   a) Select **Container** from the **Operating Systems** drop down list.
   b) Read and Accept the **TIBCO End User License Agreement**.
   c) Select the radio button for **Individual file Download**.
   d) Click the **+** sign to view the individual components and select **bwce_buildpack_cf-vx.x.x.zip**.
2. Extract the contents of the zip file to a temporary location.
3. Customize the build pack for the database drivers.
   a) Follow steps outlined in **JBDC Connection** in the *Bindings and Palettes Reference* guide.
   b) Copy the appropriate driver bundle from *TIBCO_HOME*/bwce/*version*/config/drivers/shells/*<driverspecific runtime>*/runtime/plugins/ to the *<your local buildpack repo>*/resources/addons/jars folder in your temporary location.

4. Provision the OSGi bundle jar(s).

   Copy OSGified bundle jar(s) into *<your buildpack repo>*/resources/addons/jars folder in your temporary location.

5. Application Configuration Management

   See Using Configurations from Configuration Managament Services for more information.

   To add support for other systems, update *<your local buildpack repo>*/resources/java-profile-token-resolver/ProfileTokenResolver.java.

   This class has a dependency on the Jettison(1.3.3) JSON library. You can download this library from the web or from *TIBCO_HOME*/bwce/*version*/system/shared/com.tibco.bw.tpcl.org.codehaus.jettison.

6. Certificate Management

   Certificates are used by applications to connect to different systems. For example, a certificate to connect to Spring Cloud Config service or a certificate to connect to TIBCO Enterprise Message Service.

   Bundling certificates with your application is not a good idea as you would need to rebuild your application when the certificates expire. To avoid that, copy your certificates into the *<your local buildpack repo>*/resources/addons/certs folder in your temporary location.

   Once the certificates expire, you can copy the new certificates into the buildpack without rebuilding your application. Just deploy your application with the new buildpack. To access the certificates folder from your application, use the environment variable BW_KEYSTORE_PATH. For example, #BW_KEYSTORE_PATH#/mycert.jks in your application property.

7. Provision the Businessworks Container Edition Plug-in Runtime

   To add TIBCO certified plug-ins:

   1. Download appropriate Plug-in packaging, for example TIBCO ActiveMatrix BusinessWorks™ Plug-in for WebSphere MQ, from https://edelivery.tibco.com.
   2. Locate the plug-in zip file, *<product id>*_ePaas.zip or TIB_<version>_<build number>_bwce-runtime.zip from the downloaded artifacts and copy into *<your local buildpack repo>*/resources/addons/plugins.

   To add a plug-ins created using TIBCO ActiveMatrix BusinessWorks™ Plug-in Development Kit to runtime into your buildpack:

   1. Install the plug-in if it is not installed.
   2. Navigate to the TIBCO_HOME/bwce/palettes/*<plug in name>*/*<plugin version>* directory and zip the lib and runtime folders into <plugin name>.zip.
   3. Copy <plugin name>.zip to the *<your buildpack repo>*/resources/addons/plugins folder.

   Copy any required OSGi bundles e.g. driver bundles into *<your buildpack repo>*/resources/addons/jars

8. Provision the third-party client installation to runtime

   1. Package third party client installation into zip.
   2. Copy zip file to *<YOUR-BUILDPACK-REPO>*/resources/addons/thirdparty-installs folder

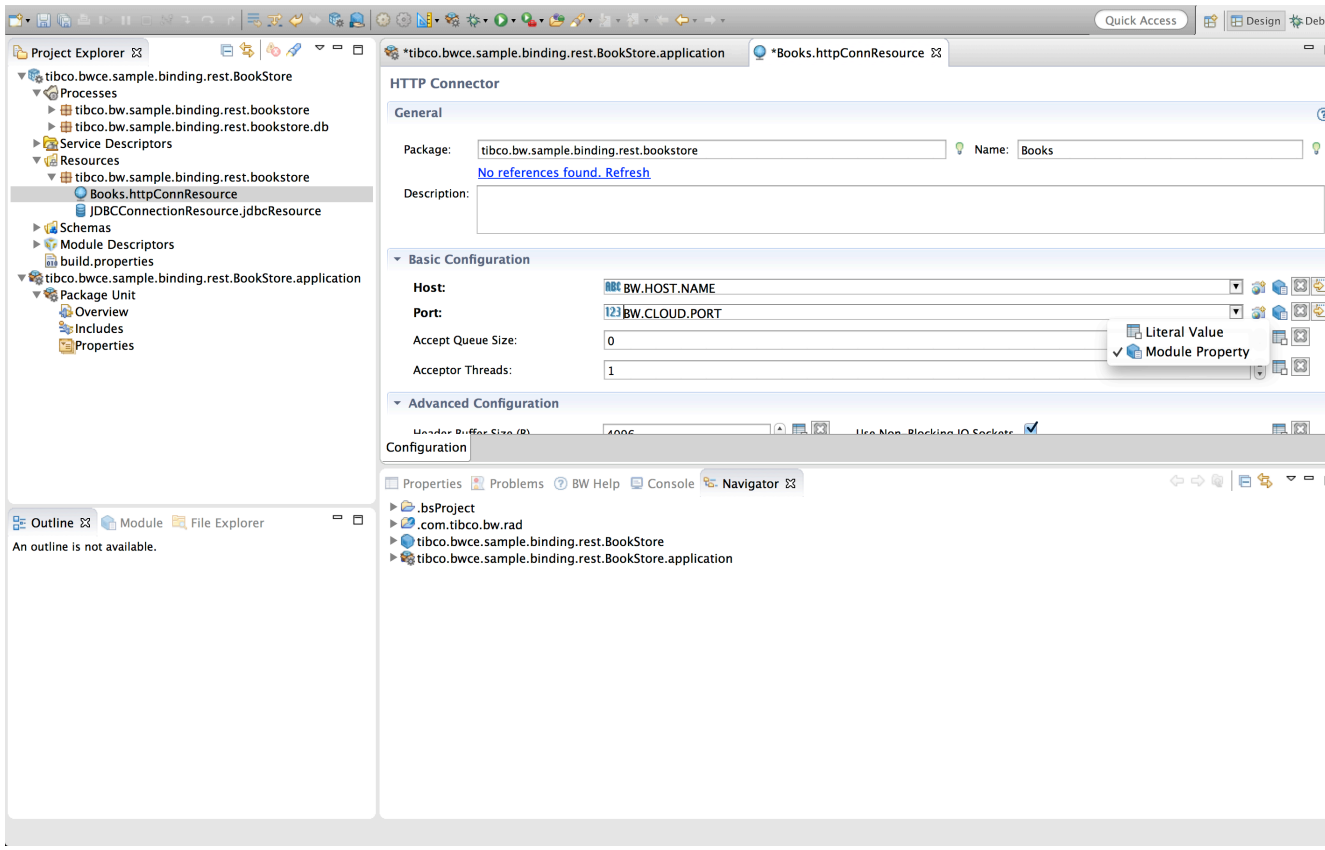
9. Provision to add custom JDBC driver

   1. Follow steps outlined in **Enabling Custom Drivers** section of **JDBC Connection** topic in the *Bindings and Palette Reference* guide.
   2. After the project has been exported as a plugin to the location you specified, locate the JAR file in the pulgin folder, and copy paste the JAR to *<YOUR-BUILDPACK-REPO>*/resources/addons/jars folder.

10. Zip the contents of the temporary location to create the TIBCO BusinessWorks™ Container Edition buildpack zip file.
11. Push the buildpack to the Cloud Foundry environment.

System Module Properties

The web applications deployed in the Cloud Foundry are expected to bind themselves to the port given by the Cloud Foundry. This can be achieved for TIBCO BusinessWorks™ Container Edition applications by configuring the `BW.CLOUD.PORT` system property for the HTTP Connector Shared Resource.



Environment Variables

This section lists the environment variables that can used for TIBCO BusinessWorks™ Container Edition application deployment.

Environment Variable	Default Values	Description
BW_LOGLEVEL	ERROR	Used to set a log level for the TIBCO BusinessWorks™ Container Edition application. The default value is ERROR. Supported values are: • INFO • DEBUG • WARN • ERROR

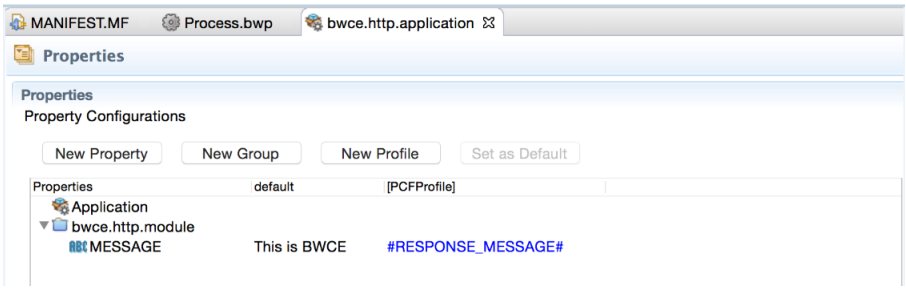
Environment Variable	Default Values	Description
BW_ENGINE_THREADCOUNT	8	Used to set engine thread count for the TIBCO BusinessWorks™ Container Edition application.
BW_ENGINE_STEPCOUNT	-1	Used to set engine step count for the TIBCO BusinessWorks™ Container Edition application.
BW_APPLICATION_JOB_FLOWLIMIT	n/a	Used to set flow limit for TIBCO BusinessWorks™ Container Edition application.
APP_CONFIG_PROFILE	default	Name of the application profile that is to be used from a configuration management system such as ZUUL, Spring Cloud Config etc.
BW_PROFILE	n/a	Used to set the name of the BusinessWorks profile from the application.
BW_JAVA_OPTS	n/a	Used to set Java properties that are used at run time. The properties are specified using name-value pairs and are separated by spaces. For example, `BW_JAVA_OPTS="-Dname=value -Dname=value"`
MASHERY_SERVICE_CONFIG		Applications can pass TIBCO Mashery configuration through the `MASHERY_SERVICE_CONFIG` environment variable. The value of the environment variable is a JSON string with the required TIBCO Mashery configuration. See Integrating with TIBCO Mashery for more information.
ENABLE_SERVICE_DIRECT_REGISTRATION	false	If set to `true`, an application will be registered using its host IP and port instead of the application route.
CONSUL_SERVER_URL		Used to set Consul server configuration. For example, `CONSUL_SERVER_URL=http://127.0.0.1:8085.` This must be set if you intend to use Consul for application configuration or for service registration and discovery.

Environment Variable	Default Values	Description
EUREKA_SERVER_URL		Used to set Eureka server configuration. For example, `EUREKA_SERVER_URL=http://127.1.0.1:8080/eureka` This must be set if you intend to use Eureka for service registration and discovery.
CF_TARGET	None	To add an Elastic Runtime self-signed certificate automatically at runtime, set `CF_TARGET` to the API endpoint of your Elastic Runtime instance. For example, `CF_TARGET=https://api.cf.demo.com.` This is useful when using Spring Cloud Services in your application.
BW_JMX_CONFIG	n/a	Used to set JMX configuration (JMX port) for monitoring TIBCO BusinessWorks™ Container Edition application. For example, `BW_JMX_CONFIG: 8050`
BW_JAVA_GC_OPTS	-XX:+UseG1GC	Used to set JAVA GC configuration. The value should be one of the standard Java GC VM Options. For example, `BW_JAVA_GC_OPTS: -XX:+UseParallelGC`
-e DISABLE_BWCE_EAR_VALIDATION=true	None	Used to deploy the ActiveMatrix BusinessWorks 6.x application `EAR` file on TIBCO BusinessWorks Container Edition without converting project to Container Edition and rebuilding EAR file from TIBCO Business Studio Container Edition  Ensure that the ActiveMatrix BusinessWorks 6.x EAR file is exported. ActiveMatrix BusinessWorks 6.x EAR file should only have TIBCO BusinessWorks Container Edition supported activities and features.

Using Configurations from Configuration Management Services

You can use configurations from the configuration management services such a ZUUL or Spring Cloud Config by defining a token such a *#<property name>#* in the application properties, where *<property name>* is the name of the configuration parameter.

For example, `#RESPONSE_MESSAGE#`.



The environment variable `APP_CONFIG_PROFILE` can be used to identify the profile name for your deployment environment, for example `QA` or `Prod`. See Environment Variables for Cloud Foundry for more information.

Integrating with Spring Cloud Config Server

Support for application configuration is provided through an integration with the Spring Cloud Config Server.

You first create an instance of the service using the Pivotal Cloud Foundry Apps Manager and then bind this service instance to your application.

TIBCO BusinessWorks Container Edition communicates with Spring Cloud® Services using HTTPS. If your Cloud Foundry installation uses a self-signed SSL certificate, this certificate should be added to the buildpack before your application can be deployed. To add self-signed SSL certificate to the buildpack, see the *Certificate Management* section of Customize the Buildpack.

Using Custom User Provided Services (CUPS) for Config Servers

The following are required when integrate your application with custom user provided services (CUPS) Spring Cloud Config or ZUUL Configuration service in Cloud Foundry:

1. Name of the Cloud Foundry service instance must contain either `spring-cloud-config` or `zuul-config`.
2. The service configuration must contain the `uri` parameter.

   For example, for ZUUL

```
{
   "credentials": {
       "uri": "http://{ZUUL-SERVER-IP}:{PORT}/zuul"
     },
  "label":"user-provided",
  "name": "zuul-config-server",
  "syslog_drain_url": "",
  "tags": []
}
```

   For Spring Cloud

```
{
   "credentials": {
       "uri": "http://{SPRING-CLOUD-SERVER-IP}:{PORT}"
     },
  "label":"user-provided",
  "name": "spring-cloud-config-server",
  "syslog_drain_url": "",
  "tags": []
}
```

The environment variable `APP_CONFIG_PROFILE` can be used to set `ProfileName`. If this variable is not set, `default` name is used for the profile.

Using Cloud Foundry Services

To integrate with the Cloud Foundry database or messaging services (CUPS or Managed) follow the steps outlined in **Modifying Application Properties** in the *Application Development* guide.

Modifying Application Properties

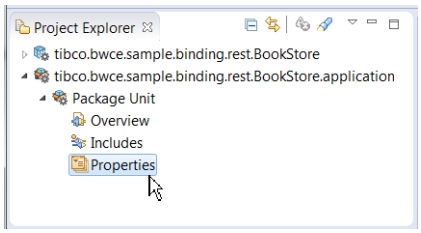
In order to make use of configuration from the Cloud Foundry runtime you need to modify the application properties.
Follow these steps to modify application properties.

The property names are specified in the format `#<serviceName>.credentials.<parameter>#`, where *<serviceName>* is the name of the service and *<parameter>* is the name of the configuration parameter.

For example, `#postgres.credentials.username#`.

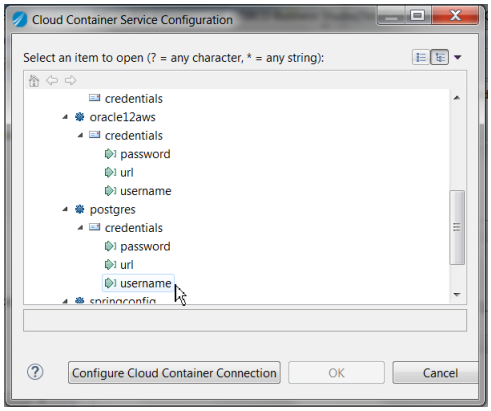
Procedure

1. In the **Project Explorer** view, expand the project application and double click **Properties**.



   The properties are now displayed in the Editor view.

2. To modify the value of a property, first click the property name and the click the icon.
   The **Cloud Container Service Configuration** window is displayed. This window initially contains no values.
3. Click **Cloud Container Connection**.
   The **Cloud Container Connection Configuration** window is displayed.
4. Specify the configuration properties and click **Test Connection**.
   If the connection is successful, variables are populated in the **Cloud Container Service Configuration** window as shown in the next figure.



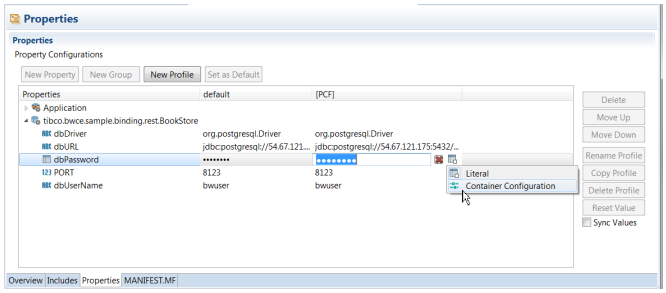
5. Choose the appropriate value and click **OK**.
6. Click **Save**.

Modifying Properties of Type Password or Integer

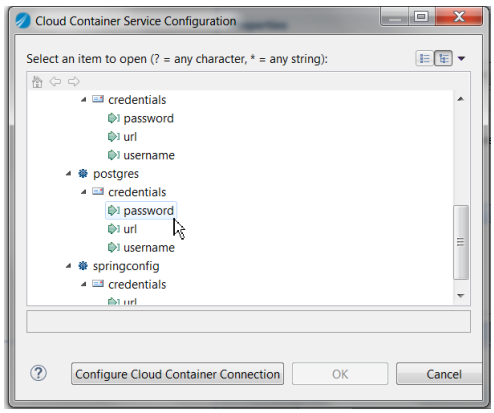
Follow these steps to modify properties of type Password or Integer.

Procedure

1. Click on the property of type Password or Integer and then click .
2. Choose either Literal or Container Configuration.



3.

Option	Description
Literal	Type the value for the property.
Container Configuration	Choose the value from the cloud container configuration. 1. Click the icon that is now visible. 2. Click **Configure Cloud Container Connection** in the **Cloud Container Service Configuration** window. 3. Specify the configuration properties and click **Test Connection**. 4. If the connection is successful, variables are populated in the **Cloud Container Service Configuration** window. 

4. Click **Save**.

Integrating with TIBCO Mashery®

The following options are available to integrate with TIBCO Mashery:

- Using a custom user provided service (CUPS)

In your cf CLI execute

```
cf cups mashery-service -p
"areaUuid,clientId,clientSecret,password,trafficManagerDomain,username,masheryApi
ServerUri"
```

All the service configuration parameters listed above are mandatory and the service name has to contain the word `mashery`.

- Using an environment variable The TIBCO Mashery configuration can be passed using the `MASHERY_SERVICE_CONFIG` environment variable. The value of this environment variable is a JSON string.

```
MASHERY_SERVICE_CONFIG: '{"username":"xxx@tibco.com","password": "xxxx",
"clientId":"xxxxxxxx","clientSecret":"xxxxxx","masheryApiServerUri": "https://
api.mashery.com","areaUuid": "xxxxxxxxx-xxxx-xxxx-xxxx-
xxxxxxxxx","trafficManagerDomain":"xxxxx.api.mashery.com"}'
```

The TIBCO BusinessWorks™ Container Edition runtime looks for an attached service or an environment variable for the TIBCO Mashery configuration. If found, that configuration is used to register HTTP based endpoints (HTTP, REST, or SOAP/HTTP).

Application Development for Docker

The following section provides information about system module properties and environment variables as they apply to TIBCO BusinessWorks™ Container Edition.

Switching the Container Platform

Procedure

1. In TIBCO Business Studio™ Container Edition, click **Window** > **Preferences** .
   On Mac OS X platform, click **TIBCO Business Studio Container Edition > Preferences**.
2. In the Preferences dialog click **BusinessWorks Container Edition** > **Container Platform**.
3. Choose either **Cloud Foundry** or **Docker**.
4. Click **Apply** and then **OK**.

What to do next

TIBCO Business Studio™ Container Edition has to be restarted for the changes to take effect.

It is recommended that you clean your workspace after TIBCO Business Studio™ Container Edition is restarted.

This option can be accessed from the TIBCO Business Studio™ Container Edition menu **Project** > **Clean**.

Starting TIBCO Business Studio™ Container Edition in the Docker Mode

To start TIBCO Business Studio™ Container Edition in the Docker mode add the property `ContainerTarget` and set the value to `Docker` in the following file:

- *`TIBCO_HOME`*`\studio\<version>\eclipse\configuration\config.ini`

Here is a snippet of a sample `config.ini` file:

```
eclipse.application=org.eclipse.ui.ide.workbench
eclipse.p2.data.area=@config.dir/../p2
osgi.bundles.defaultStartLevel=4
ContainerTarget=Docker
```

The preference is now set to `Docker` for every new workspace.

Environment Variables

This section lists the environment variables that can be used for TIBCO BusinessWorks™ Container Edition application deployment on Docker and Docker based platforms.

Environment Variable	Default Values	Description
BW_LOGLEVEL	ERROR	Used to set a log level for the TIBCO BusinessWorks™ Container Edition application. The default value is ERROR. Supported values are: • INFO • DEBUG • WARN • ERROR
BW_ENGINE_THREADCOUNT	8	Used to set engine thread count for the TIBCO BusinessWorks™ Container Edition application.
BW_ENGINE_STEPCOUNT	-1	Used to set engine step count for the TIBCO BusinessWorks™ Container Edition application.
BW_APPLICATION_JOB_FLOWLIMIT	n/a	Used to set flow limit for TIBCO BusinessWorks™ Container Edition application.
APP_CONFIG_PROFILE	n/a	Name of the application profile that is to be used from a configuration management system such as ZUUL, Spring Cloud Config etc.
BW_PROFILE	n/a	Used to set the name of the BusinessWorks profile from the application.
BW_JAVA_OPTS	n/a	Used to set Java properties that are used at run time. The properties are specified using name-value pairs and are separated by spaces. For example, `BW_JAVA_OPTS="-Dname=value -Dname=value"`

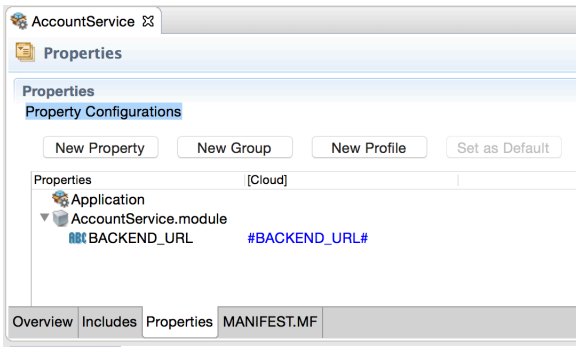
Environment Variable	Default Values	Description
MASHERY_SERVICE_CONFIG	n/a	Applications can pass TIBCO Mashery configuration information using the `MASHERY_SERVICE_CONFIG` environment variable. The value of the environment variable is a JSON string with the required TIBCO Mashery configuration. See Integrating with TIBCO Mashery for more information.
CONSUL_SERVER_URL	n/a	Used to set Consul server configuration. For example, `CONSUL_SERVER_URL=http://127.0.0.1:8085` This must be set if you intend to use Consul for application configuration or for service registration and discovery.
EUREKA_SERVER_URL		Used to set Eureka server configuration. For example, `EUREKA_SERVER_URL=http://127.1.0.1:8080/eureka` This must be set if you intend to use Eureka for service registration and discovery.
MEMORY_LIMIT	1024M	To optimize memory usage at runtime, set this environment variable to the configured memory. For example, `MEMORY_LIMIT=512M` when the Docker container is launched with `512M`.
BW_JMX_CONFIG	n/a	Used to set JMX configuration (RMI host and JMX port) for monitoring TIBCO BusinessWorks™ Container Edition application. The value should be provided in RMI_HOST:JMX_PORT format. For example, `BW_JMX_CONFIG=192.168.99.100:8050`
BW_JAVA_GC_OPTS	-XX:+UseG1GC	Used to set JAVA GC configuration. The value should be one of the standard Java GC VM Options. For example: `BW_JAVA_GC_OPTS=-XX:+UseParallelGC`

Environment Variable	Default Values	Description
-e DISABLE_BWCE_EAR_VALIDATION=true	None	Used to deploy the ActiveMatrix BusinessWorks 6.x application EAR file on TIBCO BusinessWorks Container Edition without converting project to Container Edition and rebuilding EAR file from TIBCO Business Studio Container Edition Ensure that the ActiveMatrix BusinessWorks 6.x EAR file is exported. ActiveMatrix BusinessWorks 6.x EAR file should only have TIBCO BusinessWorks Container Edition supported activities and features.

Using Configurations from Configuration Management Services

You can use configurations from the configuration management services such as Consul by defining a token such as *#<property name>#* in the application properties, where *<property name>* is the name of the configuration parameter.

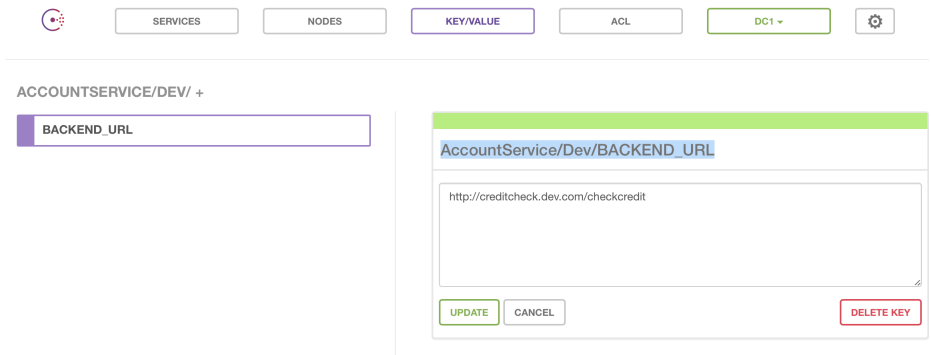
For example, `#BACKEND_URL#`.



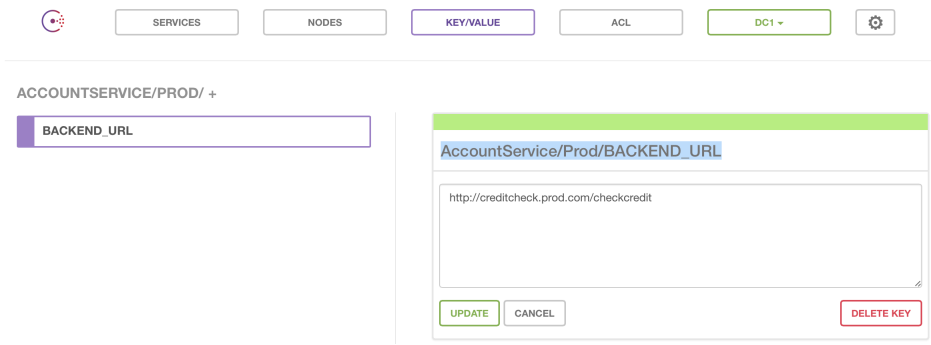
Follow these steps to use configurations from Consul:

1. Set the environment variable `CONSUL_SERVER_URL`. See Environment Variables
2. In your Consul service define the keys using the format `<BWCE_APP_NAME>/<PROFILE NAME>/<KEY Name>`.

   For example, `AccountService/Dev/BACKEND_URL`



`AccountService/Prod/BACKEND_URL`



Creating the TIBCO BusinessWorks™ Container Edition Base Docker Image

Prerequisites

- Download the TIBCO BusinessWorks™ Container Edition runtime zip file, `bwce-runtime-<version>.zip`, from http://edelivery.tibco.com.

  To download this file,

  1. Select **Container** from the **Operating Systems** drop down list.
  2. Read and Accept the **TIBCO End User License Agreement**.
  3. Select the radio button for **Individual file Download**.
  4. Click the **+** sign to view the individual components and select `bwce-runtime-<version>.zip`.
- An installation of Docker.

Procedure

1. Navigate to the *TIBCO_HOME*`/bwce/<version>/docker` directory.
2. Copy the `bwce-runtime-<version>.zip` file to the *TIBCO_HOME*`/bwce/<version>/docker/resources/bwce-runtime` folder.
3. Open a command terminal and execute the following command from the *TIBCO_HOME*`/bwce/<version>/docker` folder:

   ```
   docker build -t TAG-NAME .
   ```

   For example,

   ```
   docker build -t tibco/bwce:latest .
   ```

Result

The TIBCO BusinessWorks™ Container Edition base Docker image is now created. This base Docker image can now be used to create Docker application images.

Extending the Base Docker Image

The TIBCO BusinessWorks™ Container Edition base Docker image must be uploaded to your Docker environment before deploying any applications.

This base Docker image can be extended to support third-party drivers, OSGI bundles, integration with application configuration management systems, and application certificate management.

Prerequisites

Create the base Docker image following steps in Creating the TIBCO BusinessWorks™ Container Edition Application Docker Image.

The base Docker image can be extended in the following ways:

- **Provision supported JDBC drivers**

  1. Run bwinstall[.exe] help from <BWCE_HOME>/bin and follow instructions to add the driver to your TIBCO BusinessWorks Container Edition installation.
  2. Copy your driver jars from *<TIBCO_HOME>*/bwce/version/config/drivers/shells/<driverspecific runtime>/runtime/plugins/ to a temporary folder.
  3. From the temporary folder use the Docker file given below to copy these jars into the base Docker image:

     ```
     FROM tibco/bwce:latest
     COPY . /resources/addons/jars
     ```

- **Provision the OSGi bundle jar(s)**

  1. Copy OSGified bundle jar(s) into a temporary folder.
  2. From the temporary folder use the Docker file given below to copy these jars into the base Docker image:

     ```
     FROM tibco/bwce:latest
     COPY . /resources/addons/jars
     ```

- **Application Configuration Management**

  See Using Configurations from Configuration Management Services for more information.

  To add support for other systems, update *TIBCO_HOME*/bwce/*version*/docker/java-code/ProfileTokenResolver.java.

  This class has a dependency on the Jackson (2.6.x) JSON library. You can download this library from the web or from *TIBCO_HOME*/system/shared/com.tibco.bw.tpcl.com.fasterxml.

- **Certificate Management**

  Certificates are used by applications to connect to different systems. For example, a certificate to connect to TIBCO Enterprise Message Service.

  Bundling certificates with your application is not a good idea as you would need to rebuild your application when the certificates expire. To avoid that, copy your certificates into a temporary folder. From the temporary folder use the Docker file given below to copy these certificates into the base Docker image:

  ```
  FROM tibco/bwce:latest
  COPY . /resources/addons/certs
  ```

  Once the certificates expire, you can copy the new certificates into the base Docker image without rebuilding your application. Just deploy your application with the base Docker image. To access the certificates folder from your application, use the environment variable BW_KEYSTORE_PATH. For example, #BW_KEYSTORE_PATH#/mycert.jks in your application property.

- **Provision the TIBCO Businessworks™ Container Edition Plug-in Runtime**

  To add TIBCO certified plug-ins:

  1. Download appropriate Plug-in packaging, for example TIBCO ActiveMatrix BusinessWorks™ Plug-in for WebSphere MQ, from https://edelivery.tibco.com.

2. Locate the plug-in zip file, *<product id>*_ePaas.zip or TIB_<version>_<build number>_bwce-runtime.zip from the downloaded artifacts and copy into a temporary folder. From the temporary folder use the Docker file given below to copy these plugin runtime zip files into the base Docker image:

```
FROM tibco/bwce:latest
COPY *.zip /resources/addons/plugins
```

To add a plug-ins created using TIBCO ActiveMatrix BusinessWorks™ Plug-in Development Kit to runtime into your base Docker image:

1. Install the plug-in if it is not installed.
2. Navigate to the *TIBCO_HOME*/bwce/palettes/*<plug in name>*/<plugin version> directory and zip the lib and runtime folders into <plugin name>.zip.
3. Copy <plugin name>.zip to a temporary folder.
4. From the temporary folder use the Docker file given below to copy these plugin runtime zip files into the base Docker image:

```
FROM tibco/bwce:latest
COPY *.zip /resources/addons/plugins
```

- **Provision the third-party installations to runtime**

  To add the third-party installation to Docker image:

  1. Package third-party client installation into zip.
  2. Copy zip to a temporary folder.
  3. From the temporary folder use the Docker file given below to copy zip files into the base Docker image:

```
FROM tibco/bwce:latest
COPY *.zip /resources/addons/thirdparty-installs
```

  4. Build the Docker image:

```
docker build -t YOUR-TAG.
```

- **Provision to add custom JDBC driver**

  1. Follow steps outlined in **Enabling Custom Drivers** section of **JDBC Connection** topic in the *Bindings and Palette Reference* guide. After the project has been exported as a plugin, locate the JAR in the plugins folder and copy it to a temporary folder.
  2. From the temporary folder use the Docker file given below to copy these jars into the base Docker image:

```
FROM tibco/bwce:latest
COPY . /resources/addons/jars
```

What to do next

Open a command terminal from the temporary folder where you created the Dockerfile and execute:

```
docker build -t TAG-NAME .
```

For example,

```
docker build -t tibco/bwce-oracledriver:latest .
```

Integrating with TIBCO Mashery

You integrate TIBCO BusinessWorks™ Container Edition applications with TIBCO Mashery using an environment variable.

- Using an environment variable the TIBCO Mashery configuration can be passed using the `MASHERY_SERVICE_CONFIG` environment variable. The value of this environment variable is a JSON string.

```
MASHERY_SERVICE_CONFIG: '{"username":"xxx@tibco.com","password": "xxxx",
"clientId":"xxxxxxxx","clientSecret":"xxxxxx","masheryApiServerUri": "https://
api.mashery.com","areaUuid": "xxxxxxxxx-xxxx-xxxx-xxxx-
xxxxxxxxx","trafficManagerDomain":"xxxxxxxx.api.mashery.com"}'
```

  The TIBCO BusinessWorks™ Container Edition runtime looks for an environment variable for the TIBCO Mashery configuration. If found, that configuration is used to register HTTP based endpoints (HTTP, REST, or SOAP/HTTP).

Service Registration and Discovery

The service registration and discovery mechanism lets TIBCO BusinessWorks™ Container Edition services announce their availability and lets clients dynamically find these services.

- **Cloud Foundry**

  Service registration and discovery on Cloud Foundry is done using one of the following options:

  – Support for service registration and discovery is provided through an integration with the Eureka Service Registry service.

    You first create an instance of the service using the Pivotal Cloud Foundry Apps Manager and then bind this service instance to your application.

  – Support for Consul service registration and discovery is provided using the `CONSUL_SERVER_URL` environment variable. See Environment Variables for Cloud Foundry for more information.

TIBCO BusinessWorks™ Container Edition communicates with Spring Cloud® Services using HTTPS. If your Cloud Foundry installation uses a self-signed SSL certificate, this certificate should be added to the buildpack before your application can be deployed. To add self-signed SSL certificate to the buildpack, see the *Certificate Management* section of Customize the Buildpack.

- **Docker and Docker based platforms**

  – Support for Eureka service registration and discovery on Docker and Docker based platforms is provided using the `EUREKA_SERVER_URL` environment variable. See Environment Variables for Docker or Docker based Platforms for more information.

  – Support for Consul service registration and discovery on Docker and Docker based platforms is provided using the `CONSUL_SERVER_URL` environment variable. See Environment Variables for Docker or Docker based Platforms for more information.

Service Registration Configuration

For information on configuration see *HTTP Connector* in the *Shared Resources* section of the *Palette Reference*.

On Cloud Foundry to use the direct registration mode set the ENABLE_SERVICE_DIRECT_REGISTRATION environment variable to `true`.

See Environment Variables for Cloud Foundry for more information.

Service Discovery Configuration

For information on configuration see *HTTP Client* in the *Shared Resources* section of the *Palette Reference*.

Circuit Breaker Support

The Circuit Breaker feature is supported through the use of Hystrix libraries.

For more information see *HTTP Client* in the *Shared Resources* section of the *Bindings and Palette Reference*.

- Hystrix Dashboard Integration
  - Pivotal Cloud Foundry - Bind the Hystrix dashboard service to your application and use the dashboard to view the commands.

    Alternatively, you can retrieve the Hystrix command stream using `http://<routable url>/hystrix.stream`

    For example,

    ```
    http://myapp.demopcf.com:80/hystrix.stream
    ```
  - Docker or Docker based platforms - You can retrieve the Hystrix command stream using the command `http://<Container IP>:8090/hystrix.stream`

    For example,

    ```
    http://132.99.1.6:8090/hystrix.stream
    ```

Best Practices

As the business requirements become more complex, so do the business processes that are designed to implement them. TIBCO provides some best practices to help design processes that are readable, reusable, and manageable.

Control Visibility with Scopes

A scope is similar to a block concept in programming languages and is useful to isolate or encapsulate process variables, thus avoiding conflicts with variable names used elsewhere in the process. Use of scopes helps reduce the number of module properties needed for the entire application, which must be unique for all lexical scopes. When designing or viewing a process in TIBCO Business Studio™ Container Edition, scope constructs can be collapsed to enhance readability of the process and reduce clutter.

Promote Reuse with Sub-processes

A sub-process is similar to a sub-routine in programming languages and is useful to keep a block of code small and maintainable. Sub-processes, if declared public, can be called from other processes, thus enabling the logic to be reused.

Consolidate Literal Values

Keep the number of literal values in process logic and activity configurations to a minimum by consolidating them in the Process Properties tab at the process level. This makes it easier to view and maintain the literal values. In addition, the process properties can be promoted to module properties, which can then be controlled at the application level.

Externalize with Module Properties

Configuration parameters can be externalized as module properties. At runtime, the values from the module properties are injected into process and activity configuration parameters upon application startup. This allows environmental specific application properties to be set at the time of deployment or in some cases, post deployment. Database password is a good example of a module property.

Use Profiles for Staging

You can group module properties with the current set of property values into a named profile. An application can have multiple profiles, each having its own set of property values. At run time, you can deploy the same application and stage it multiple times using different profiles.

Defining Service Contracts

When designing complex business processes, ensure that the service contracts on the interfaces are well-defined.

Avoid XML Collisions

Avoid defining schema (XSD) or WSDL components with the same qualified names in the same module. Doing so may result in XML collisions at the module level.

If, for some reason, you need to define schema or WSDL components with the same qualifies names, then define the schema or WSDL components in separate shared modules.

Close Unnecessary Projects in Workbench

Keep the number of open projects in your Eclipse workbench to a minimum by closing the unnecessary projects. Having too many BusinessWorks Container Edition projects open in the Eclipse workbench may adversely affect the UI performance.

Use Project Clean

Sometimes TIBCO Business Studio™ Container Edition reports incorrect validation errors that are not related to design or development issues. It is recommended that you clean your project as it forces Eclipse to discard all build problems and states, and rebuild the projects from scratch. This option can be accessed from the menu **Project** > **Clean**.

Manage TIBCO Business Studio™ Container Edition Workspaces

If you are working with multiple major, minor, or service pack levels of the product, use different workspaces for different versions.

Increase Log Levels

When debugging issues at design-time, increasing the log levels can provide additional information on the issues. You can customize the log levels for configurations like Debug and Run by editing the respective `logback.xml` configuration files.

The logging configurations are accessible from **Run** > **Debug Configuration** > **Advanced** > **Logging Configuration**. Permissible log level values are `INFO`, `TRACE`, `DEBUG`, `WARN`, and `ERROR`. These levels can be applied to activities, shared resources, bindings, engine, and so on.