Important Information

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

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

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

TIBCO, TIBCO ActiveMatrix BusinessWorks, TIBCO Rendezvous, TIBCO Enterprise Message Service, TIBCO Business Studio, TIBCO Enterprise Administrator, TIBCO ActiveSpaces, TIBCO Runtime Agent, TIBCO Designer, and Two-Second Advantage are either registered trademarks or trademarks of TIBCO Software Inc. in the United States and/or other countries.

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

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

THIS SOFTWARE MAY BE AVAILABLE ON MULTIPLE OPERATING SYSTEMS. HOWEVER, NOT ALL OPERATING SYSTEM PLATFORMS FOR A SPECIFIC SOFTWARE VERSION ARE RELEASED AT THE SAME TIME. SEE THE README FILE FOR THE AVAILABILITY OF THIS SOFTWARE VERSION ON A SPECIFIC OPERATING SYSTEM PLATFORM.

THIS DOCUMENT IS PROVIDED “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT.

THIS DOCUMENT COULD INCLUDE TECHNICAL INACCURACIES OR TYPOGRAPHICAL ERRORS. CHANGES ARE PERIODICALLY ADDED TO THE INFORMATION HEREIN; THESE CHANGES WILL BE INCORPORATED IN NEW EDITIONS OF THIS DOCUMENT. TIBCO SOFTWARE INC. MAY MAKE IMPROVEMENTS AND/OR CHANGES IN THE PRODUCT(S) AND/OR THE PROGRAM(S) DESCRIBED IN THIS DOCUMENT AT ANY TIME.

THE CONTENTS OF THIS DOCUMENT MAY BE MODIFIED AND/OR QUALIFIED, DIRECTLY OR INDIRECTLY, BY OTHER DOCUMENTATION WHICH ACCOMPANIES THIS SOFTWARE, INCLUDING BUT NOT LIMITED TO ANY RELEASE NOTES AND “READ ME” FILES.

Copyright © 2001-2014 TIBCO Software Inc. ALL RIGHTS RESERVED.

TIBCO Software Inc. Confidential Information
# Contents

**Figures** ................................................................................................................... 5

**TIBCO Documentation and Support Services** .......................................................... 6

**Changing Help Preferences** ...................................................................................... 7

**Application Development Overview** ........................................................................ 8

**Application Design Considerations** ............................................................................ 10

**Process Design Considerations** .................................................................................. 13
  Service Design Considerations ......................................................................................... 15

**TIBCO Business Studio Essentials** ............................................................................. 17

**Entity Naming Conventions** ..................................................................................... 20

**Developing a Basic Process** ....................................................................................... 21
  Creating an Application Module ....................................................................................... 21
  Creating a Shared Module ................................................................................................. 22
  Creating a Process ........................................................................................................... 22
  Configuring a Process ....................................................................................................... 23
  Creating a Sub Process ...................................................................................................... 24
  Adding Activities .............................................................................................................. 25
  Working with Transitions ................................................................................................. 27
  Working with Standard Activity Features ......................................................................... 28
  Input and Output ............................................................................................................... 29

**Using Additional Features** ......................................................................................... 34
  Using Scopes ................................................................................................................... 34
    Adding Scope Variables ................................................................................................. 34
  Defining and Using Shared Variables .............................................................................. 37
    Retrieving and Assigning a Value of a Shared Variable .................................................. 39
    Working with Critical Section Groups ........................................................................... 39
  Using Fault Handlers ....................................................................................................... 39
  Using Conversations ....................................................................................................... 41
  Using Checkpoints .......................................................................................................... 42
  Configuring Database for the Engine .............................................................................. 42
  Configuring Group Provider for the Engine ..................................................................... 44

**XPath** ......................................................................................................................... 46
  XPath Basics ................................................................................................................... 46
  XPath Expression ............................................................................................................. 48
  XPath Builder .................................................................................................................. 50

**Developing a SOAP Service** ....................................................................................... 52
  Consuming SOAP Services ............................................................................................. 55

**Developing a RESTful Service** .................................................................................... 57
  Implementing a REST Service Provider ........................................................................... 57
<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Developing Java Applications</td>
<td>59</td>
</tr>
<tr>
<td>Using a Simple Java Invoker Activity</td>
<td>59</td>
</tr>
<tr>
<td>Accessing Module Properties from Java Global Instance</td>
<td>60</td>
</tr>
<tr>
<td>Creating an Application</td>
<td>61</td>
</tr>
<tr>
<td>Generating Deployment Artifacts</td>
<td>62</td>
</tr>
<tr>
<td>Deploying an Application</td>
<td>64</td>
</tr>
<tr>
<td>Using the Debugger</td>
<td>67</td>
</tr>
<tr>
<td>Configuring the Debugger</td>
<td>68</td>
</tr>
<tr>
<td>Testing an Application in TIBCO Business Studio</td>
<td>68</td>
</tr>
<tr>
<td>Remote Debugging</td>
<td>69</td>
</tr>
<tr>
<td>Using the bwdesign Utility</td>
<td>72</td>
</tr>
<tr>
<td>Best Practices</td>
<td>74</td>
</tr>
</tbody>
</table>
Figures

Approaches to Application Development ......................................................... 9
TIBCO Business Studio Workbench ................................................................. 17
Parent Process ................................................................................................. 25
Sub Process ....................................................................................................... 25
Drag-and-Drop a Resource ............................................................................. 26
Drag-and-Drop a Resource ............................................................................. 27
Input Tab ......................................................................................................... 29
Input Editor Tab ............................................................................................... 30
Output Tab ...................................................................................................... 32
Output Editor Tab ............................................................................................. 33
Fault Handler Attached to an Inner Scope ..................................................... 40
Schema Elements in Data Source ................................................................... 47
XPath Builder ................................................................................................. 50
TIBCO Documentation and Support Services

All TIBCO documentation is available on the TIBCO Documentation site, which can be found here:
https://docs.tibco.com

Product-Specific Documentation

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

TIBCO_HOME/release_notes/TIB_BW_version_docinfo.html

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

- Concepts
- Installation
- Getting Started
- Application Development
- Administration
- Bindings and Palettes Reference
- Samples
- Error Codes
- API Reference
- Migration

How to Contact TIBCO Support

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

- 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 TIBCOmmunity

TIBCOmmunity 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. TIBCOmmunity offers forums, blogs, and access to a variety of resources. To register, go to:

http://www.tibcommunity.com
Changing Help Preferences

By default, documentation access from TIBCO Business Studio 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.

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 ActiveMatrix BusinessWorks and press Enter.
3. Select the TIBCO ActiveMatrix BusinessWorks product from the list. This opens the product documentation page for the latest version.
4. Click Download All.
5. A 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, click Window > Preferences. On Mac OS X, click TIBCO Business Studio > 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.
TIBCO ActiveMatrix BusinessWorks™ applications can be developed using either the traditional phases of waterfall development, or using an incremental and iterative approach such as Scrum. The Application Development guide explains the following:

- Approaches to application development.
- Considerations to be made when building an application.
- Information on how to work with various software components and how to generate the deployment artifact.

Application development consists of the following phases:

- **Analysis** - Analyze the business problem and identify the applications, modules, services, and processes that need to be created to solve the problem.
- **Application Creation/Design** - Create one or more applications identified during the analysis phase. TIBCO Business Studio™ provides the design-time environment to design an application and its components that implement the business logic.
- **Service Design** - Create the services identified in the analysis phase. The services can be accessed by processes that are used to implement the business logic.
- **Process Design** - Create the processes that implement the business logic. These processes can access the services configured.
- **Generating Deployment Artifacts** - Create a deployment artifact — an archive file, after creating and configuring the processes and services.

If any changes to the design or configurations are made, the archive file must be regenerated.

There are two main approaches to application development: top-down and bottom-up.

Top-down is a design approach that begins with a holistic view of the application, specifying the major functions or interfaces it will need before the next level of details. This process is repeated until the most granular pieces are designed and implemented. The application is then ready for testing. In ActiveMatrix BusinessWorks™ 6.x, top-level services and processes can be designed and developed first before moving to the lower levels.

In the bottom-up approach, the basic elements of the application are first specified and developed as building blocks. These reusable parts are then connected to form functional units that serve a higher purpose. This process is repeated until the design grows in complexity and completeness to form an application. In ActiveMatrix BusinessWorks 6.x, the building blocks can be created as layers of services, subprocesses, and shared resources. These building blocks are assembled together to form application modules or shared modules. These modules are then assembled together to form an application.

In practice, even a new application can have existing services to leverage from. As a result, a problem can be approached from both top and bottom, resulting in a hybrid approach. The bottom part can start creating reusable shared modules to encapsulate existing system services that are well defined. The top part can start with the business requirements and break it down to intermediate layers, until the reusable modules are reached.

Either top-down or bottom-up approaches can be used with service-driven or process-driven design patterns. Service-driven means 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 is where these SOA design principles can be followed: standardized service contract, loose coupling, service abstraction, service reusability, service statelessness, and service compositability.
Process-driven means the business processes or integration flows are first realized and captured. Service contracts may or may not be applicable in a process-centric application, especially for batch or EAI-type automation scenarios.

**Approaches to Application Development**

Each of these approaches can be followed in conjunction with the waterfall or Scrum development methods.

The generation of the deployment artifact indicates that the application can be deployed to the runtime. Any further changes to the design-time configurations require the deployment artifact to be regenerated. See *TIBCO ActiveMatrix BusinessWorks Administration* for deployment and administration details.
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 table, Salient features of integration styles, provides guidelines to choose a high-level integration style for your applications.

**Salient features of integration styles**

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

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. However, you can use conversations to correlate the related messages between two or more parties.

Choosing the Modularity

An application module is the smallest unit of resources that is named, versioned, and packaged as part of an application, and then executed in the ActiveMatrix BusinessWorks runtime. It cannot provide capabilities to other modules.

A shared module is the smallest unit of resources that is named, versioned, and packaged as part of an application and can be used by other modules that are part of the same application. Shared modules export their functionality (process, shared resources, and schema namespaces) to application modules or to other shared modules. When creating a new module, select a shared module if the business logic needs to be shared across multiple applications. Shared modules can also be used if XML collisions exist.
### Differences between Application and Shared Modules

<table>
<thead>
<tr>
<th></th>
<th>Runtime</th>
<th>Reusability</th>
<th>Encapsulation</th>
<th>XML Namespace Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application Modules</td>
<td>Can be executed by the run time</td>
<td>Can be used by one or more applications.</td>
<td>Processes within an application module are visible to each other. However, the processes are not visible outside of the module.</td>
<td>Namespace can be provided by multiple documents.</td>
</tr>
<tr>
<td></td>
<td>when packaged as part of an application.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Shared Modules</td>
<td>Cannot be executed by the run time unless utilized by an application module.</td>
<td>Can be used by one or more application modules or shared modules.</td>
<td>Processes within a shared module are visible to each other. However, only the processes defined as public are visible outside of the shared module.</td>
<td>Only one document can provide the namespace; that is, two documents cannot have the same namespace. All schemas and WSDL files are visible to other modules that depend on the shared module.</td>
</tr>
</tbody>
</table>

### Choosing Implementation Technologies for the Modules

When implementing the business logic, ActiveMatrix BusinessWorks provides flexibility ranging from developing applications graphically without coding, to using existing Java classes (or libraries), to writing custom code. Application modules or shared modules typically consist of one or more business processes that define the business logic. Create an application or shared module using the GUI to leverage the rich orchestration capabilities provided by ActiveMatrix BusinessWorks.

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 when creating an application module. Alternatively, create a module that contains existing Java code or custom code.
### Differences between Process Modules and Java Modules

<table>
<thead>
<tr>
<th>Orchestration Capabilities</th>
<th>Visibility</th>
<th>Granularity</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Process Modules</td>
<td>High</td>
<td></td>
<td>Better suited for coarse-grained functionality and process constructs.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>High visibility of process flow logic, services, and bindings.</td>
<td>Account opening, mortgage loan, and so on.</td>
</tr>
<tr>
<td>Java Modules</td>
<td>Low</td>
<td>Low</td>
<td>Better suited for fine-grained functionality that has a very specific function, and often requires very little or no process constructs.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Low</td>
<td>Query flight status, update product description, and so on.</td>
</tr>
</tbody>
</table>
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 Stateful and Stateless Processes

Stateful processes maintain the state across multiple operations. They are better suited when you need the server to maintain the state across operations. For processes that involve related message exchanges between the same or different consumers, conversations can be used to maintain state across operations.

Stateless processes do not maintain state. They are better suited when you need to process higher loads of requests as each operation is executed independently. They do not require correlation or conversations between multiple operations in a process, thus allowing the server to process each operation without maintaining any state information. The client can choose to maintain the state information, if needed.

<table>
<thead>
<tr>
<th>Maintains State</th>
<th>Data Sharing</th>
<th>Conversations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stateful Processes</td>
<td>Across multiple operations and interfaces.</td>
<td>Data can shared by activities across operations that executing as part of the same job.</td>
</tr>
<tr>
<td>Stateless Processes</td>
<td>Does not maintain state.</td>
<td>Data is not shared.</td>
</tr>
</tbody>
</table>

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. See Choosing Between Process Properties, Module Properties, and Application Properties for details.

Variables are used to save the state at different levels. They can be classified into process variables, scope variables, and shared variables. See Choosing Between Process Variables, Scope Variables, and Shared Variables for details.

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

<table>
<thead>
<tr>
<th>Scope/Visibility</th>
<th>Values</th>
<th>Additional Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Process Properties</td>
<td>Visible within a process.</td>
<td>Literal, shared resource reference, or a module property reference.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Module Properties</td>
<td>• Visible within the module.</td>
<td>• Literal or a shared resource reference.</td>
</tr>
<tr>
<td></td>
<td>• Not visible or changeable from Administrator.</td>
<td>• 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</td>
</tr>
</tbody>
</table>

TIBCO ActiveMatrix BusinessWorks™ Application Development
### 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.

See the Concepts guide and Using Shared Variables for more information about shared variables.

### 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 faults thrown at the selected level.

Error transitions specify the transition in case of an error and can be used to handle error conditions for an activity. Error transitions are available on selected activities that can throw exceptions; they are not available for process starters, signal-in activities, and groups.

### 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 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 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.

<table>
<thead>
<tr>
<th></th>
<th>Data Abstraction</th>
<th>State Information</th>
<th>Overhead of Additional Parameters (Headers or other SOAP elements)</th>
</tr>
</thead>
<tbody>
<tr>
<td>REST Services</td>
<td>Resources</td>
<td>Stateless</td>
<td>Less</td>
</tr>
<tr>
<td>SOAP Services</td>
<td>Operations</td>
<td>Stateful</td>
<td>High</td>
</tr>
</tbody>
</table>
TIBCO Business Studio Essentials

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

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

Using TIBCO Business Studio, designers implement an executable application that can be deployed to ActiveMatrix BusinessWorks runtime.

Starting TIBCO Business Studio

To start TIBCO Business Studio on Windows, select Start > All Programs > TIBCO_HOME > TIBCO Business Studio 3.6 > Studio for Designers. On Linux or Mac OS, select the TIBCO Business Studio executable located at `<TIBCO_HOME>/studio/3.6/eclipse/`.

On the Workspace Launcher dialog, accept the default workspace or browse to create a new workspace, and then click OK. TIBCO Business Studio 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 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, perform one of the following:

- 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 Development Environment

TIBCO Business Studio 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 ActiveMatrix BusinessWorks projects are stored.

TIBCO Business Studio 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 Capabilities
  - Create a new BusinessWorks Application Module
  - Create a new BusinessWorks Shared Module
  - Debug
  - Run

- **Perspectives**: Contain an initial set and layout of views that are needed to perform a certain task. TIBCO Business Studio 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 ActiveMatrix BusinessWorks applications, modules, and other resources in your workspace, and the Properties view displays the properties for the selected resource. To open a view, select **Window > Show View > view_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 ActiveMatrix BusinessWorks 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 to implement the business logic. See Developing a Basic Process.

**Testing and Debugging an Application**

Using TIBCO Business Studio 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.
Entity Naming Conventions

Most of the ActiveMatrix BusinessWorks 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 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.
- From the Module Descriptors > Overview getting started area, click Create a 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.

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 an 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.
- From the Module Descriptors > Overview getting started area, click Create a 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.
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.

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.
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 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.

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Package</strong></td>
<td>Displays the name of the package containing the package. This field is not editable.</td>
</tr>
<tr>
<td><strong>Name</strong></td>
<td>Name of the process. This field is not editable.</td>
</tr>
<tr>
<td><strong>Target Namespace</strong></td>
<td>Target namespace for the process. You can specify a different target namespace for the process.</td>
</tr>
<tr>
<td><strong>Modifiers</strong></td>
<td>Modifiers define the visibility of the process outside its package.</td>
</tr>
<tr>
<td>Property Name</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>-------------</td>
</tr>
<tr>
<td>Public</td>
<td>can be invoked by processes that are defined either inside or outside the package.</td>
</tr>
<tr>
<td>Private</td>
<td>can be invoked only by processes that are part of the same package.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Mode</th>
<th>Mode defines whether the process depends on the engine to maintain its state:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stateful</td>
<td>relies on the ActiveMatrix BusinessWorks engine to maintain its state across invocations.</td>
</tr>
<tr>
<td>Stateless</td>
<td>requires the ActiveMatrix BusinessWorks engine to maintain its state across invocations</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Activation</th>
<th>Activation mode for a process defines the way in which processes are activated at runtime.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multiple AppNodes: At runtime, the application is distributed and activated on all the AppNodes in the AppSpace. In the event of a failure on one of the AppNodes, the application continues to run with fewer AppNodes.</td>
<td></td>
</tr>
<tr>
<td>Single AppNode: At runtime, the application is activated on only one AppNode in the AppSpace. In the event of a failure, another AppNode will be activated and any check pointed data can be recovered.</td>
<td></td>
</tr>
</tbody>
</table>

This feature requires the engine persistence mode to be set to group and the database and group provider to be configured. See Configuring Database for the Engine and Configuring Group Provider for the Engine for details.

Creating a Sub Process

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 Sub 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.

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. Click **Finish** to create a subprocess.

**Result**

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

**Parent Process and a Sub Process 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**

![Parent Process Diagram](image)

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**

![Sub Process Diagram](image)

**What to do next**

Once the subprocess is created, you can proceed to:

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

**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 File Explorer or 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, Conversation, 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, Conversation, 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.

**Adding Activities From the File Explorer**

You can add pre-configured activities to a process by dragging-and-dropping a selected file such as an XML file from the File Explorer. To do so, follow these steps:

1. In the File Explorer, select a file 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 from the File palette.
3. From the menu, select the activity you want to add to the process.

In the example, drag and drop the file Book-0001.xml from the File 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.

**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 activity processing. When an activity throws an error or fault, none of the success conditions are executed; only the error transition is executed.

**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.

**Event**
For activities that wait for incoming events, such as incoming TIBCO Rendezvous® messages, this tab specifies the timeout for the incoming event.

**Conversations**
Used to add new conversations. See Using Conversations for details about conversations.

**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 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.

**Input Editor Tab**

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

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:

![Diagram of primitive element](image)

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:

![Diagram of reference element](image)

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:

![Diagram of attribute](image)

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:

- Wildcard Namespace (a space-delimited list of the namespaces can be entered).
- Entering the Minimum Occurs number (default is 0).
- Selecting from the drop-down list the Maximum Occurs number (1 or unbounded).

- 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**

![Output Tab Image]

**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.
Using Additional Features

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

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

Using Scopes

A scope is a group without any conditions that is used to encapsulate activities and variables from the outer scope.

Prerequisites

Select the activities you want to add to a Scope.

Procedure

1. Right-click on the selection and select **Create Group > Scope**. The selected activities will be encapsulated in a new scope.
2. Configure the new scope from the Properties view.
   a) **General** tab
      - **Name**: Specify a name for the scope.
      - **Group Type**: Default is set to **Scope**, which is a group of activities without any conditions. Change the group type to create a group with conditions.
   b) **Description** tab
      - **Description**: Enter a description for the new scope.
   c) **Variables** tab
      You can add local variables to the group from the **Variables** tab. See **Adding Scope Variables** for details on adding variables.
      A scope variable can override a process variable if they have the same name. Use the **Assign** activity to override a process variable with the scope variable.

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**

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

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**

<table>
<thead>
<tr>
<th>Field/Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resource Name</td>
<td>Specify a name for the new schema.</td>
</tr>
<tr>
<td>Workspace Location</td>
<td>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.</td>
</tr>
<tr>
<td>Choose a Root Element</td>
<td>Add a primitive element to the new schema using the icon <strong>Add Primitive Element</strong>.</td>
</tr>
<tr>
<td></td>
<td>The new primitive element will appear listed under the root element. Double-click the element to configure it.</td>
</tr>
<tr>
<td>Primitive Types</td>
<td>Select the primitive type for the element from the drop-down list:</td>
</tr>
<tr>
<td></td>
<td>• String</td>
</tr>
<tr>
<td></td>
<td>• Integer</td>
</tr>
<tr>
<td></td>
<td>• Decimal</td>
</tr>
<tr>
<td></td>
<td>• Boolean</td>
</tr>
<tr>
<td></td>
<td>• Date &amp; Time</td>
</tr>
<tr>
<td></td>
<td>• Binary</td>
</tr>
<tr>
<td></td>
<td>• URI</td>
</tr>
<tr>
<td></td>
<td>• Any</td>
</tr>
</tbody>
</table>
### Field/Action

<table>
<thead>
<tr>
<th>Field/Action</th>
<th>Description</th>
</tr>
</thead>
</table>
| 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](image) | Use to rename the schema element. You can choose to preview and update all references to the element. |
| ![Go To](image) | |
| ![Accept Changes](image) | Accept the changes entered for the new schema element. |
| ![Cancel Changes](image) | Cancel the changes accepted for the new schema element. |
| ![Remove Selected Element](image) | 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](image). Select the variable type from the drop-down list and specify a default value.

<table>
<thead>
<tr>
<th>Variable Type</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>String</td>
<td>None.</td>
</tr>
<tr>
<td>Integer</td>
<td>1</td>
</tr>
</tbody>
</table>
Variable Type | Default Value
---|---
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.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>General Tab</td>
<td>Variable Name</td>
</tr>
<tr>
<td>Field Name</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>-------------</td>
</tr>
</tbody>
</table>
| 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... |
| Persistent (applicable to module shared variables only) | By default, the value of a module shared variable is stored in memory and the current state of the module shared variable would be lost in case the engine (or the AppNode) crashes.  
Select the check box to persist the current value of the module shared variable. The current state of the variable in the engine’s persistent storage is only updated when the value of the variable changes. Also, a persistent module shared variable can be made visible across AppNodes in an AppSpace when the engine persistent mode is set to "group".  

<table>
<thead>
<tr>
<th>Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>The engine persistence must be configured for the current value of the module shared variable to persist. See Configuring Database for the Engine for details.</td>
</tr>
<tr>
<td>Job shared variables cannot be configured to be persistent.</td>
</tr>
<tr>
<td>Description Tab</td>
</tr>
</tbody>
</table>
| 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.

Working with Critical Section Groups

Critical Section groups and shared locks can be used to synchronize access to shared variables. A Critical Section group allows only one process instance to execute the Critical Section group and its contents at a time. Use a Critical Section group to contain the activities that access the shared variables, Set Shared Variable and Get Shared Variable. Once a process instance begins executing a Critical Section group, other concurrently running process instances that are associated with that Critical Section group wait at the start of the group until the currently running process instance exits the Critical Section group. This ensures that the value of the shared variable is not modified while another process instance is accessing it. See Bindings and Palettes Reference > Basic Activities Palette > Critical Section for more information about using Critical Section groups and shared locks.

Best Practices

Critical section groups cause multiple process instances to wait for one process instance to execute the activities in the group. As a result, there may be performance implications when using these groups. When creating critical section groups, use the following guidelines to avoid potential performance issues:

- Keep the duration of a Critical Section group as short as possible. That is, put only a very few activities in a Critical Section group, and only use activities that execute very quickly.
- Avoid nesting Critical Section groups. If you must use nesting, ensure that Lock shared configuration resources are used in the same order in all process definitions. Deadlocks can occur if you do not specify the Lock resources in the same order in nested Critical Section groups for all process definitions.
- Do not include any activities that wait for incoming events or have long durations, such as Request/Reply activities, Wait For, Sleep, or other activities that require a long time to execute.

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**

![Diagram of a process with fault handlers]

**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 of 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.
Using Conversations

Conversations are used for stateful business processes, which means that for completion, processes require correlation of message exchanges. Such processes can be reentrant and so the previous process context is maintained for continuity.

Conversations are always initiated by one activity and joined by other activities. All operations that are part of the stateful process must generate a conversation ID and reply to the original client that contains the conversation ID.

For example, an operation Submit Purchase Order in a stateful process gets the Purchase Order ID in response. If the client wishes to cancel the purchase order, the client must use this correlation ID (Purchase Order ID) to invoke the Cancel Purchase Order operation.

Building a Conversation

Procedure

1. Right-click the activity that needs to initiate the conversation and select Conversation > Create New Conversation.
   The Conversations tab in the Properties view displays the conversation name and action 'Initiate'.

2. Right-click the activity that needs to join the conversation and select Conversation > Join Conversation > Conversation_Name.
   The Conversations tab in the Properties view displays the conversation name and action 'Join'.
   The Conversations tab of any activity that participates in conversations lists all the conversations it is participating in.

3. Click on the conversation name to specify the key data.

   The initiating key is returned as a part of the response, and the client must provide the same key when calling a related operation the next time. This ensures that the first and second operations are called by the same client and the two operations are part of the same conversation.
Using Checkpoints

A **Checkpoint** activity saves the state and data of a process instance so that it can be recovered at a later time, in case of a failure.

If an ActiveMatrix BusinessWorks engine fails, all processes that have a **Checkpoint** activity can be recovered and resume execution from the last checkpoint executed in the process instance.

Only the most recent state is saved by a checkpoint. If you have multiple checkpoints in a process, only the state from the last executed checkpoint is available for recovering the process.

Checkpoints save the state of the entire process instance. A process (parent process) can call another process (sub-process) in two ways: in-line and non-inline. An in-line sub-process is executed as part of the parent process job, while the non-inline sub-process spawns a new job. When a checkpoint activity is executed in an in-line sub-process, the checkpoint saves the state of the current process instance, including the state of the parent processes. However, when a checkpoint occurs in a non-in-line sub-process, the checkpoint saves the state of the spawned process instance only.

Checkpoints require the engine persistence mode to be either **datastore** or **group**. To configure the persistence modes, see:

- For Datastore: [Configuring Database for the Engine](#)
- For Group: [Configuring Database for the Engine](#) and [Configuring Group Provider for the Engine](#)

If the engine persistence mode is not configured with the correct value, an application with **Checkpoint** activity encounters an error at deployment.

Recovering After a Failure

Following a crash, when the engine is restarted, the process instances are recovered from the last checkpoint automatically. That is, all process instances that were check pointed will continue processing from the last executed checkpoint activity.

Ensure that the process has all of the data required to continue processing following a recovery. When placing your checkpoint in a process, be careful with certain types of process starters or incoming events, so that a recovered process instance does not attempt to access resources that no longer exist. For example, consider a process with an HTTP process starter that takes a checkpoint after receiving a request but before sending a response. In this case, when the engine restarts after a crash, the recovered process instance cannot respond to the request since the HTTP socket is already closed. As a best practice, place the response activity before the checkpoint so that any response is sent before a checkpoint is taken. Also, do not place the **Checkpoint** activity in a critical section or an event handler.

Configuring Database for the Engine

Checkpoint activity and other persistence features require the engine persistence mode (`bw.engine.persistenceMode`) to be configured for a **datastore** or **group** mode. When the engine persistence mode property is configured for **datastore** or **group** mode, the engine requires a database configuration.

**Procedure**

1. Create the engine database by executing the bundled script `create.sql`. Scripts for creating engine database for various database types are located at `BW_HOME/config/dbscripts/engine`.
2. To change the engine persistence mode, run the utility to set the persistence mode property `bw.engine.persistenceMode` to `datastore` or `group`, and then configure the engine database connection details.

```
bw.engine.persistenceMode=[datastore | group]
```
The database connection configuration can be specified at the AppSpace or the AppNode level. The database connection details specified at the AppSpace level will apply to all AppNodes within the AppSpace.

When the engine persistence mode property is set to group, the database connection configuration must be specified only at the AppSpace level.

When the engine persistence mode property is set to datastore, the database connection configuration cannot be shared by two or more AppNodes in the same AppSpace. As a result, the database connection configuration can be specified at the AppSpace level only if the AppSpace contains a single AppNode. For an AppSpace that contains two or more AppNodes, each AppNode requires a unique database and the database connection configuration must be specified at the AppNode level.

3. To set database configuration properties at the AppSpace level:
   a) Copy the existing AppSpace config.ini file (located in the root of the AppSpace folder), or the AppSpace config.ini template file appspace_config.ini_template (located in BW_HOME/config/) to a temporary location.
   b) Edit the engine persistence mode property, bw.engine.persistenceMode, and set it to datastore or group.
      
      bw.engine.persistenceMode=[datastore | group]
   c) Configure the following database connection properties in the BW Engine datastore configuration section of the config.ini file:
      
      #-----------------------------
      # Section:  BW Engine datastore configuration.  
      # The properties in this section are applicable to the BW Engine database.  
      # All the properties in this section are required if the BW Engine general 
      # property  "bw.engine.persistenceMode" is set to "datastore" or "group". 
      #-----------------------------
      # BW Engine Database Driver.  NOTE: The HSQLDB JDBC driver 
      # "org.hsqldb.jdbc.JDBCDriver" is not supported for the "group" mode and 
      # other BusinessWorks supported JDBC driver must be used for the 
      # group mode (bw.engine.persistenceMode=group). 
      bw.engine.db.jdbcDriver=org.hsqldb.jdbc.JDBCDriver
      
      # BW Engine Database URL. 
      bw.engine.db.url=jdbc:hsqldb:%CONFIG_AREA%/databases/enginedb
      
      # BW Engine Database User Name. 
      bw.engine.db.userName=sa
      
      # BW Engine Database User Password. 
      bw.engine.db.password=
      
      # BW Engine Database Connection Pool Size. 
      bw.engine.db.maxConnections=15

      When setting the password property (bw.engine.db.password), the default format is plain text. However, you can encrypt the password using the bwadmin command and use the generated encrypted text as the password.

      The bwadmin bwenginedb command will display BW engine datastore configuration settings.

4. To set database configuration properties at the AppNode level:
   a) Copy the existing AppNode config.ini file (located in the root of the AppNode folder) to a temporary location.
   b) Set engine persistence mode property bw.engine.persistenceMode to datastore or group and configure engine database connection details.
      
      bw.engine.persistenceMode=[datastore | group]
   c) Configure the engine database connection properties in the BW Engine datastore configuration section of the config.ini file. By default, the AppNode config.ini file does not contain these
properties. Copy these properties from the AppSpace config.ini template file, appspace_config.ini_template, located in BW_HOME/config to the AppNode config.ini file and provide the database connection details.

5. Use one of the following config admin commands to push the configuration to the AppSpace or the AppNode:
   - AppSpace:
     ```shell
     bwadmin[admin] > config -d myDomain -a myAppSpace -cf <temporaryLocation>/config.ini
     ```
   - AppNode:
     ```shell
     bwadmin[admin] > config -d myDomain -a myAppNode -cf <temporaryLocation>/config.ini
     ```

Result

You used the bwadmin command line to set the database configuration property. You can also use the Admin UI to set this property. See the following topics:

- Editing an AppSpace Configuration
- Editing an AppNode Configuration

Configuring Group Provider for the Engine

The managed fault tolerance feature requires the engine persistence mode to be configured for the group mode. The group mode also supports the Checkpoint activity and other persistence features. When configured for the group persistence mode, the engine requires both a database and a group provider such as TIBCO Enterprise Message Service™ (EMS) to be configured.

Procedure

1. Create the engine database and configure the database for the engine as described in Configuring Database for the Engine.
2. Set engine persistence mode property (bw.engine.persistenceMode) to group and configure the engine group configuration.
   a) Copy the existing AppSpace config.ini file (located in the root of the AppSpace folder), or the AppSpace config.ini template file appspace_config.ini_template (located in BW_HOME/config) to a temporary location.
   b) Edit the ActiveMatrix BusinessWorks engine persistence mode property, bw.engine.persistenceMode, and set it to group.
   ```ini
   bw.engine.persistenceMode=group
   ```
   c) Specify the group name and group provider technology in the config.ini file. The group name is optional and it defaults to domain and AppSpace names separated by an underscore (_). Only TIBCO Enterprise Message Service (EMS) is supported by the group provider technology.

```
# Section:  BW Engine Group Configuration.
#
# The properties in this section are applicable to the BW Engine group.
# Some of the properties in this section are required when the BW Engine # property "bw.engine.persistenceMode" is set to "group".
#
# BW Engine Group Name. This is an optional property and it specifies name of # the BW engine group. If this property is not specified, then the group name # defaults to "Group_<DomainName>_<AppSpaceName>".
#bw.engine.groupName=mytestgroup
#
# BW Engine Group Connection Provider Technology. This is a required
```
property
# when the persistenceMode is set to "group"
(bw.engine.persistenceMode=group)
# and it specifies the BW Engine group communication technology. The only
# supported value is "ems". The group connection provider technology
# property requires additional configuration. See section "BW Engine Group Connection
# Provider EMS" for additional configuration.
bw.engine.groupProvider.technology=ems

d) Specify the group provider configuration:

#******************************************************************************
# Section:  BW Engine Group Connection Provider EMS Configuration.
#
# Some of the properties in this section are required when the BW Engine Group
# Connection Provider Technology property
"bw.engine.groupProvider.technology"
# value is set to "ems".
#
#******************************************************************************

# BW Engine Group Connection Provider EMS URL. This property is required if
# the group provider technology is "ems".
bw.engine.groupProvider.qin.EMSServerUrl=tcp://localhost:7222

# BW Engine Group Connection Provider EMS User Name. This property is
required
# if the group provider technology is "ems".
bw.engine.groupProvider.qin.EMSUserName=admin

# BW Engine Group Connection Provider EMS User Password. This property is
required if the group provider technology is "ems".
bw.engine.groupProvider.qin.EMSPassword=

# BW Engine Group Connection Provider EMS Member Prefix. This property is
# optional and the default value is "EMSGMS".
#bw.engine.groupProvider.qin.EMSPrefix=EMSGMS

# BW Engine Group Connection Provider EMS Recovery Timeout in ms. This
# property is optional and the default value is "5000" ms.
#bw.engine.groupProvider.qin.EMSRecoveryTimeout=5000

# BW Engine Group Connection Provider EMS Recovery Attempt Delay in ms. This
# property is optional and the default value is "500" ms.
#bw.engine.groupProvider.qin.EMSRecoveryAttemptDelay=500

# BW Engine Group Connection Provider EMS Recovery Attempt Count. This
# property is optional.
#bw.engine.groupProvider.qin.EMSRecoveryAttemptCount=

# BW Engine Group Connection Provider EMS Connect Attempt Count. This property
# is optional.
#bw.engine.groupProvider.qin.EMSConnectAttemptCount=

# BW Engine Group Connection Provider EMS Connect Attempt Delay in ms. This
# property is optional.
#bw.engine.groupProvider.qin.EMSConnectAttemptDelay=

When setting the password property (bw.engine.groupProvider.qin.EMSPassword), the
default format is plain text. However, you can encrypt the password using the bwadmin
command and use the generated encrypted text as the password.

3. Use the config admin command to push the configuration to the AppSpace: bwadmin[admin] >
   config -d myDomain -a myAppSpace -cf <temporaryLocation>/config.ini.
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 ActiveMatrix BusinessWorks 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.

<table>
<thead>
<tr>
<th>Elements</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elements</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| 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. |
| 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 **TIBCO BW Predefined Module Properties**, such as Activity Name, Application Name, Domain Name, and so on. |
| 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.

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.
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**.

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 SOAPSrvceBinding, select **HTTP** from the **Transport Type** dropdown 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.

**Consuming SOAP Services**

The request message is generated by the SOAP reference binding for a service and response message is received by the reference binding from the service.

**Creating a Consumer for SOAP Service**

**Procedure**

1. Click on the process package, for example, "HelloWorld", and then click **Create a new Business Works Process** icon.
2. Specify the process name as **HelloWorldConsumer** and click **Finish**.
3. Drag and drop the **HelloWorldSOAP** portType to the right of the process editor.
4. Select and drop a **Timer** and a **Log** activity on the process and join it with the **Invoke** activity as shown in the image. Also, configure the **Log** activity with a message.
The SOAP reference binding is created.

5. Run the project.
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. ActiveMatrix BusinessWorks 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.

Schema Restrictions

- No wildcards or attribute wildcards.
- No foreign-namespace elements or attributes (single namespace).
- 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 allowed.

Restrictions on JSON

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

Not currently supported

- JSON nulls
- 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, 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.

   - 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.

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 and 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 can be used to develop and debug 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 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 check box Use Java configuration.
- For an existing application module, right-click on the project name in the Project Explorer view and select Configure > Convert to Java project.

Accessing Java Classes or Libraries from an ActiveMatrix BusinessWorks Application

An ActiveMatrix BusinessWorks 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 ActiveMatrix BusinessWorks 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 within the ActiveMatrix BusinessWorks application: When the Java code needs to 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 ActiveMatrix BusinessWorks 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.
- Pure Java application: When your application contains pure Java code and does not require any business logic to be implemented, create an application containing only the Java code in a module. This pure Java application can be deployed to the ActiveMatrix BusinessWorks runtime.

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 as the ActiveMatrix BusinessWorks process.

Prerequisites

The project must be configured with Java nature. See Adding Java Nature to a Project for details.

Procedure

1. In the Project Explorer view, expand the application module project and right-click on the Java source folder, src (default), and select New > Class.
2. In the New Java Class wizard, enter the package name and name of the Java class, and click Finish to create the Java class in the specified package. For example, enter `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 from ActiveMatrix BusinessWorks!" when invoked.

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

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

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

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

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

5. Open the ActiveMatrix BusinessWorks 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 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 "from ActiveMatrix BusinessWorks!" 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 ActiveMatrix BusinessWorks 6.x Module Properties in a user-defined Java code referenced in Java Global Instance, follow these steps:

### Procedure

1. In the ActiveMatrix BusinessWorks 6.x module, specify a dependency on the package "com.tibco.bw.palette.shared.java" using Import-Package.
   a) Double-click Dependencies located under ActiveMatrix BusinessWorks 6.x Module > Module Descriptors. This opens the BW Manifest Editor.
   b) In the Imported Packages section click the Add tab to add the dependency on a package com.tibco.bw.palette.shared.java.

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/value pair of ActiveMatrix BusinessWorks 6.x Module Properties.
Creating an Application

An application is a collection of one or more modules and can be executed in the runtime.

**Important:** An application created using TIBCO ActiveMatrix BusinessWorks Express can run in TIBCO ActiveMatrix BusinessWorks Enterprise. However, an application created using ActiveMatrix BusinessWorks Enterprise cannot run in ActiveMatrix BusinessWorks Express.

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 runtime as it contains all the bundles and metadata that is required to deploy and run the application.

Applications are developed using the features available in TIBCO Business Studio and can range from simple to very complex. An ActiveMatrix BusinessWorks application consists of an application module, which consists of one or more processes that define the business logic, and zero or more shared modules. See Application Modules and Shared Modules in the Concepts guide for details. ActiveMatrix BusinessWorks applications can also contain OSGi bundles that do not contain ActiveMatrix BusinessWorks artifacts.

An application archive contains one or more OSGi bundles, one each for all the modules referenced directly or indirectly by the application. It also contains application metadata which is used during deployment.

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 ActiveMatrix BusinessWorks processes in SVG format, which can be rendered in the Admin UI.

There are multiple ways to create a deployment artifact:

- From the Project Explorer view in TIBCO Business Studio, 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.

TIBCO ActiveMatrix BusinessWorks™ Application Development
Artifacts intended to be exported from a shared module must be contained in their respective special folders. For example, schemas must be contained in the Schemas folder, WSDL files in the Service Descriptors folder, and processes must be contained in a package under the Processes folder.

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 the AppSpace.

When you deploy and start an application, if the archive file contains the SVG format of business processes, you can view the process diagrams for the processes from the Admin UI. See the Administration guide for details.
Deploying an Application

You can deploy an application in TIBCO Business Studio.

Prerequisites

The bwagent must be running. For information on runtime entities, see the Administration guide. If no network exists, you can create one by entering values in the Add Network dialog box and creating runtime entities in the Deploy Application dialog box.

Procedure

1. Connect to a deployment server.
   a) In the Deploy pane, right-click Deployment Servers and select Add Network.

   ![Add Network dialog box]

   b) In the Add Network dialog, specify the HTTP interface and port for the network. The default HTTP interface is the name of the bwagent. The default port is 8079. (The defaults may have been changed by your administrator.)

   ![Add Network dialog box with filled fields]

   The selected network is displayed in the Deploy pane:
2. Deploy and start your application. There are several ways to deploy:

- Drag an archive file into an Archives folder in the Deploy pane and drop it.
- Drag an application project into an AppSpace in the Deploy pane and drop it.
- Drag an archive file from the Project Explorer, from Mac Finder, or from Windows File Explorer into an AppSpace in the Deploy pane.
- Drag an archive file from the Archives folder (in the Deploy pane) into an AppSpace and drop it.
- Right-click the application and choose **Deploy Application**.

a) In the Deploy Application dialog box, choose:

- **Network**: The network to deploy to.
- **Domain**: The domain to upload to. If there are no domains in the network, a default value is provided. Click **New** to create a new domain.
- **AppSpace**: The AppSpace to deploy to. The default AppSpace name is based on the application name. Click **New** to create a new AppSpace in the selected domain.

If an AppNode does not exist, an AppNode is created. The AppNode is computed from the AppSpace name and an HTTP management port value is assigned.

The application deploys and starts. Select it in the Deploy pane to view the status in the Properties pane:
3. Right-click the application in the Deploy window and choose **Stop** to stop it.
Using the Debugger

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

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. 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

Using TIBCO Business Studio 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 by starting the ActiveMatrix BusinessWorks engine, domain (BWEclipseDomain), AppSpace (BWEclipseAppSpace), and AppNode (BWEclipseAppNode) from within TIBCO Business Studio. 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 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.

Remote Debugging

You can debug an application running on a remote AppNode through TIBCO Business Studio.

Procedure

1. Enable the AppNode for debugging. (The AppNode must be running.)
   a) Open the network in the Deploy pane and choose the AppNode. The AppNode properties are displayed in the Properties pane.
   
   
   ![Deploy Pane Example]

   b) Click the Enable Debug icon in the Properties pane to enable remote debugging.
   c) Enter the interface and port for remote debugging on the selected AppNode in the Enable Remote Debugging dialog box.

   - **Debugger Interface**: The interface for the debugger. This value is auto-generated.
   - **Debugger Port**: The port to use for remote debugging. This is the same as the port number you entered for the remote debug configuration. This port cannot be in use. If the port is in use, the following message is displayed at the top of the dialog box: Internal server error.
The remote debugger can also be launched with the Debug icon in the Properties view. The connection parameters on the Enable Remote Debugging dialog box will automatically be entered based on the AppNode configuration.

2. In TIBCO Business Studio, create a Remote Debug launch configuration.
   a) Choose Run > Debug Configurations.
   b) In the Debug Configuration dialog box, choose Remote BusinessWorks Application > New_configuration. Enter the following information:
      - **Name**: The name of the configuration.
      - **Host**: The name of the host. This is the agent name. To find the agent name, right-click the network name in the Deployment Servers window and choose Edit. The agent name is displayed in the Agent HTTP Interface field of the Add Network dialog box.
      - **Port**: The remote debug port. The port cannot be in use.

3. Deploy the application you want to debug to a network. See Deploying an Application for information.

4. Launch the application using the Remote Debug launch configuration.
The application is launched in the debugger. Confirmation is displayed in the Debug window.
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.

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

**Procedure**

1. To use the bwdesign utility, open a terminal and navigate to `BW_HOME\bin`.
2. Type: `bwdesign`
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, 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 ActiveMatrix BusinessWorks projects open in the Eclipse workbench may adversely affect the UI performance.
Use Project Clean

Sometimes TIBCO Business Studio 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 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.