TIBCO BusinessWorks™ Container Edition

Concepts

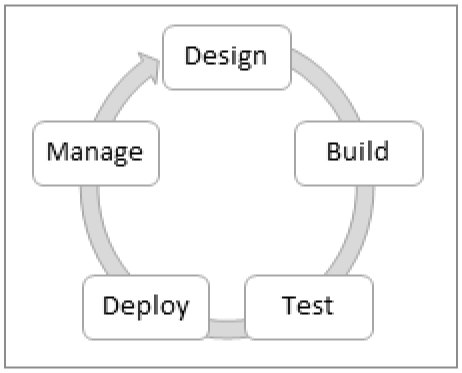
Version 2.10.0 | December 2024

Contents

Overview

TIBCO BusinessWorks™ Container Edition is an integration product suite for enterprise, web, and mobile applications.

You can use this software to create services and integrate applications using a visual, model-driven development environment, and then deploy them in the TIBCO BusinessWorks Container Edition runtime environment.



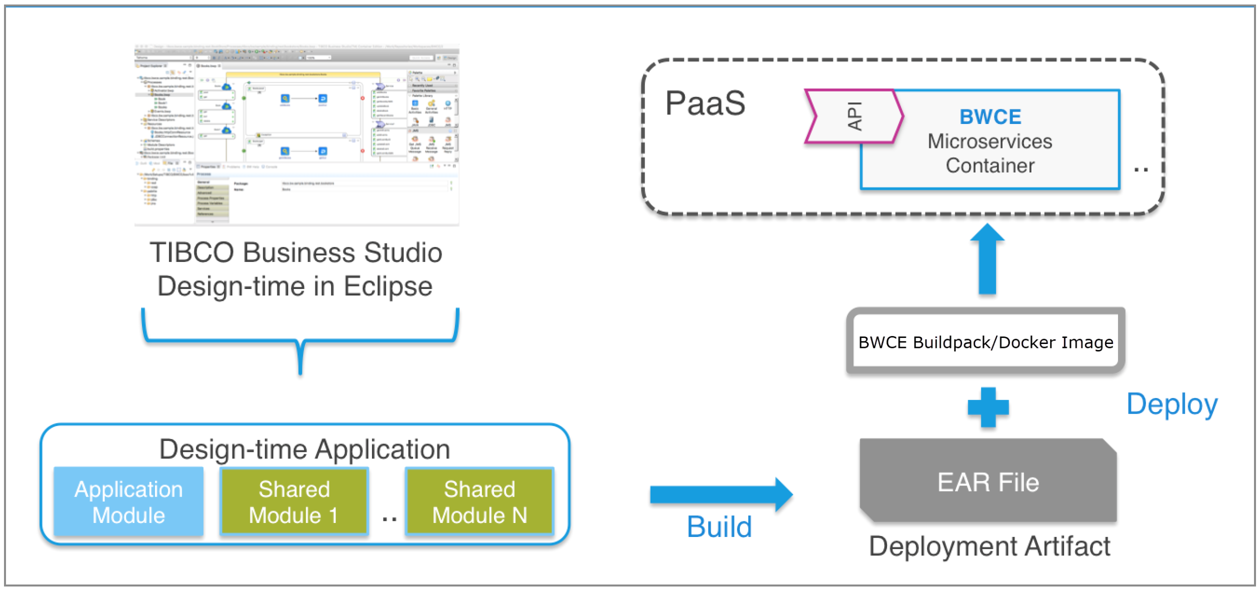
TIBCO BusinessWorks Container Edition uses the Eclipse graphical user interface (GUI) provided by TIBCO Business Studio™ for BusinessWorks™ to define business processes and generate deployable artifacts in the form of archive files.

You can use TIBCO Business Studio for BusinessWorks to model integration processes of varying complexity using any of the following integration styles:

- **Batch-oriented** - provides non-real-time integration for endpoints such as databases or files, and uses records for data abstraction.
- **Process-oriented** - provides real-time integration for endpoints such as application APIs and adapters, and uses APIs, objects, and messages for data abstraction.
- **Service-oriented** - provides real-time integration for endpoints such as web services and APIs, and uses services and messages for data abstraction.
- **Resource-oriented** - provides real-time integration for endpoints such as mobile or web applications and APIs, and uses resources for data abstraction.

Key Concepts

The following image provides an overview of the key concepts that you encounter when working with the product.



Some of these concepts are applicable exclusively to design perspective or runtime perspective, while some are applicable to both perspectives. TIBCO BusinessWorks Container Edition consists of a design time where you can develop applications that implement business logicand the runtime environment where you run the applications.

TIBCO BusinessWorks Container Edition is based on the following aspects:

- Flexibility
- Openness and Extensibility
- Modularity
- Standards-based

Flexibility

TIBCO Business Studio for BusinessWorks is designed to make adding, upgrading, and swapping of business components easy.

The flexible architecture is demonstrated by:

- A zero coding model with which you can select and drop activities onto the **Process**

Editor and configure the activities in the UI.

- Ability to build tightly coupled as well as loosely coupled services.
- Ability to build strongly typed as well as loosely typed service implementations.
- Ability to specify application configuration to be either hard-coded or late-bound.
- Encapsulation of configuration data, thus minimizing the configuration properties exposed by the application.

Openness and Extensibility

Openness and extensibility features include:

- Public APIs with which you can develop custom activities and XPath functions.
- Integration with standard Java classes and OSGi Java services to supplement the process or model-driven approach.
- Extensible Eclipse-based design-time.
- Extensible OSGi based run time.

Modularity

Modularity of the product supports:

- Large teams and distributed development through modular constructs.
- Increased visibility and traceability metadata, such as Name, Version, Exported Functionality, and Dependencies.
- Reusability with a consistent model across different technologies: Processes, Java Classes, XSDs, WSDLs, and shared resources.

Standards-based

Supported standards include:

- Protocols and API: SOAP, JSON, REST, WSDL, HTTP, HTTPS, JMS, JDBC
- Data representation and transformation: Native support for XML, XSD, XPath, JSON, XSLT
- Others: JNDI

General Concepts

TIBCO BusinessWorks Container Edition applications developed to solve business problems can range from simple to very complex solutions. These applications are packaged in deployable artifacts in the form of archive files. Understanding the general concepts is essential to both developers and administrators.

Applications

An application is a collection of one or more modules and can be run in the runtime. Applications are developed using TIBCO Business Studio for BusinessWorks.

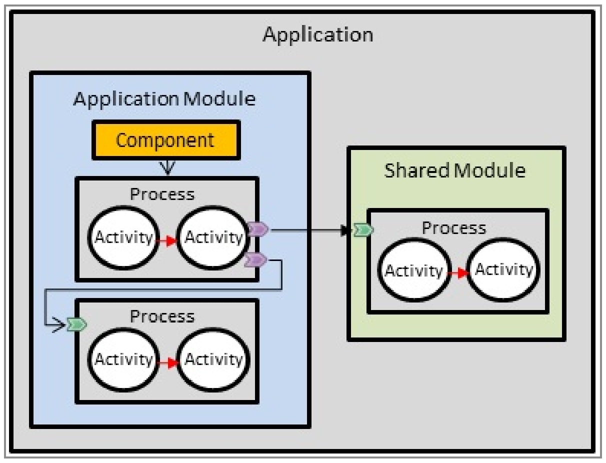
An application contains one application module, which in turn consists of one or more business processes. It can be run in the runtime. Applications are developed using TIBCO Business Studio for BusinessWorks

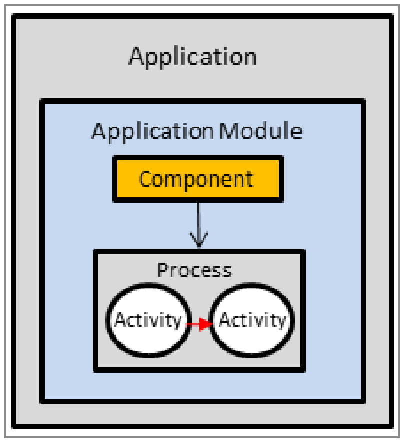
Applications are developed using features available in the product and can range from simple to very complex. An TIBCO BusinessWorks Container Edition application contains one application module (see Application Modules), which in turn consists of one or more processes that define the business logic, and zero or more shared modules (see Shared Modules). A process that is responsible for initiating the business logic at runtime is used to implement a *component* in an application module.

Applications can also contain OSGi bundles that do not contain application artifacts. For example, you can create an application that contains a Java OSGi bundle, which is also referred to as a *Java module*.

Note: The term module is used interchangeably with the OSGi bundle.

Elements of an application





After an application is developed, you can either run or debug directly in TIBCO Business Studio for BusinessWorks, or generate a deployable artifact (an archive file) that can be deployed later in the runtime environment. The deployment artifact is the only artifact that is handed over from the design-time to the runtime environment.

Modules

A module is an Eclipse project that is configured for TIBCO BusinessWorks Container Edition.

Two types of modules are supported:

- Application modules: The smallest resource that is named, versioned, and packaged as part of an application and is run in the TIBCO BusinessWorks Container Edition runtime. An application module cannot be deployed by itself in the TIBCO BusinessWorks Container Edition runtime; it must be packaged as part of an application.

- Shared modules: The smallest resource 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. A shared module cannot be deployed by itself; it must be included as part of an application module.

Application Modules

The smallest resource that is named, versioned, and packaged as part of an application and is run in the TIBCO BusinessWorks Container Edition runtime. An application module cannot be deployed by itself in the TIBCO BusinessWorks Container Edition runtime; it must be packaged as part of an application.

An application module typically contains one or more processes. An application module is configured and represented in TIBCO Business Studio for BusinessWorks, and can be used by multiple applications. Each application module contains metadata that is associated with it, such as name, version, dependencies.

An application module can include the following resources:

- **Processes:** Processes capture and represent the flow of business information between different data sources and destinations. Processes are contained within a process package. An application module can contain one or more process packages, and each of the process packages can contain one or more processes.
- **Service descriptors:** Service descriptors consist of Swagger files and WSDL files that provide the name of the service, the interface, the list of operations offered by the service, the parameters expected by the operations, and the return types.
- **Resources:** Resources are reusable configuration data that can be shared within an application. For example, Shared Resources.
- **Schemas:** Schemas define elements and attributes that can be used to define structured data.
- **Components:** The main process that is responsible for initiating the execution of the application logic is represented by a component. When the application logic is spread across multiple processes, there can be one or more components in the application

module.

- **Module Descriptors:** Module descriptors provide information about the application module such as module overview, configuration properties, dependencies, components, and shared variables.

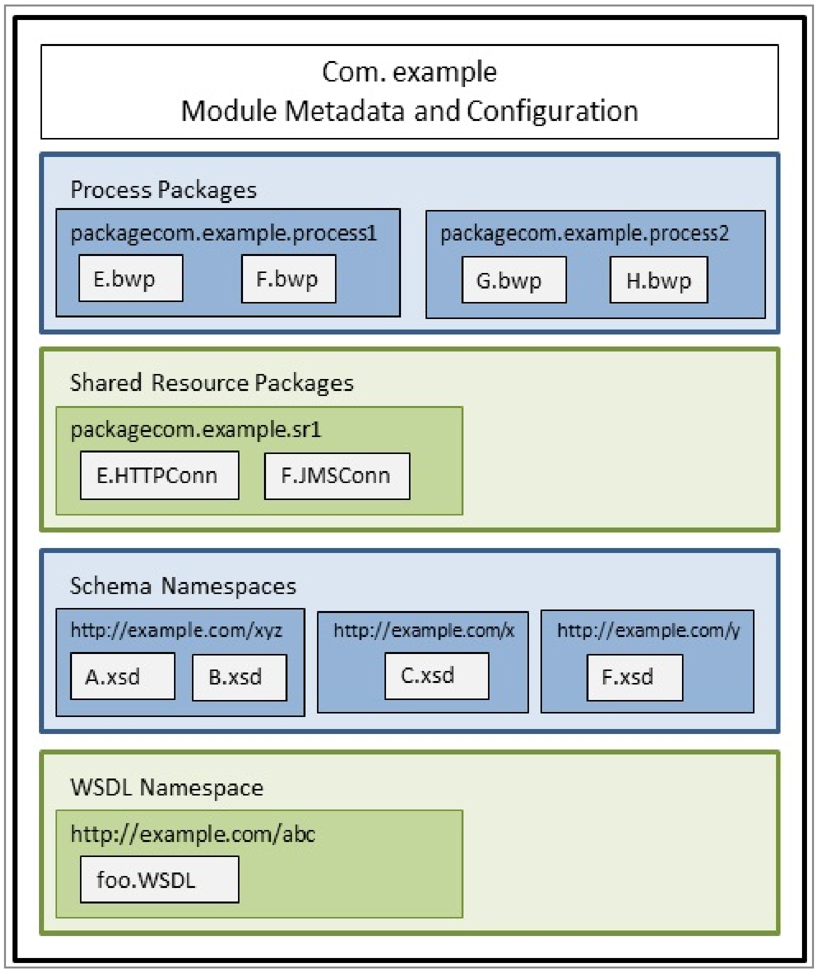
  > **Note:** Use the Component section under the **Module Descriptor** node to configure the components for this specific application module.

- **src:** Default source directory created when the project is Java enabled. A project can contain multiple source directories that are used to contain the Java classes and packages.
- **JRE System Library:** If your project is Java enabled, TIBCO Business Studio for BusinessWorks includes the required JAR files in this folder.

Application modules can depend on shared modules, which can contain processes, schemas, JSON, and WSDL files that can be used by a process in the application module.

The application modules cannot export their functionality to other modules.

Structure of an Application Module

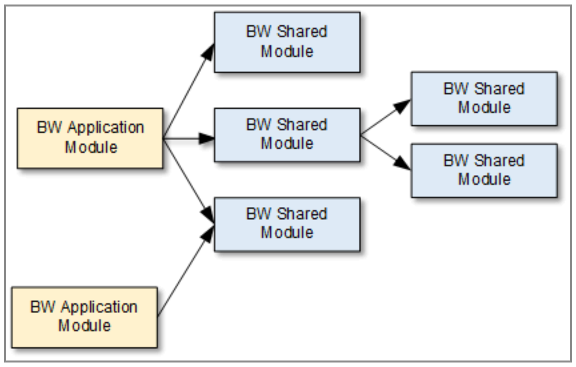


Shared Modules

The smallest resource 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.

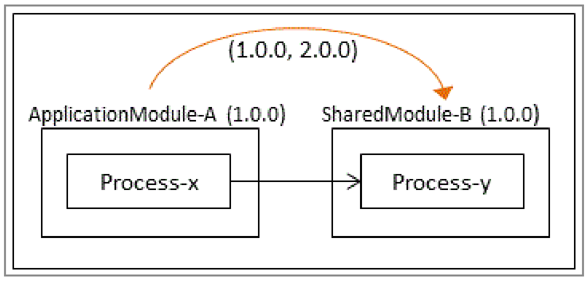
Shared modules export their functionality (processes, shared resources, and schema namespaces) to application modules or to other shared modules. This means that there is a possibility that other modules in the system depend on a shared module for this information.

Relationship Between Application Modules and Shared Modules

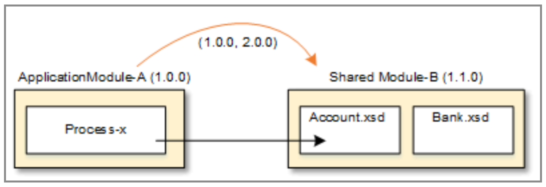


Shared modules can depend only on other shared modules and cannot depend on application modules.

At the module level, a process can reference another process in a different module.



A process can also reference a WSDL or a schema defined in a different shared module. Schemas intended to be exported from a shared module must be contained in the **Schemas** special folder.



Note: Two modules with the same package names cannot be used in the same application. Package names must remain unique across multiple shared modules and application modules within an application. If an application contains two packages with the same name, rename of the packages, or remove a package from an application.

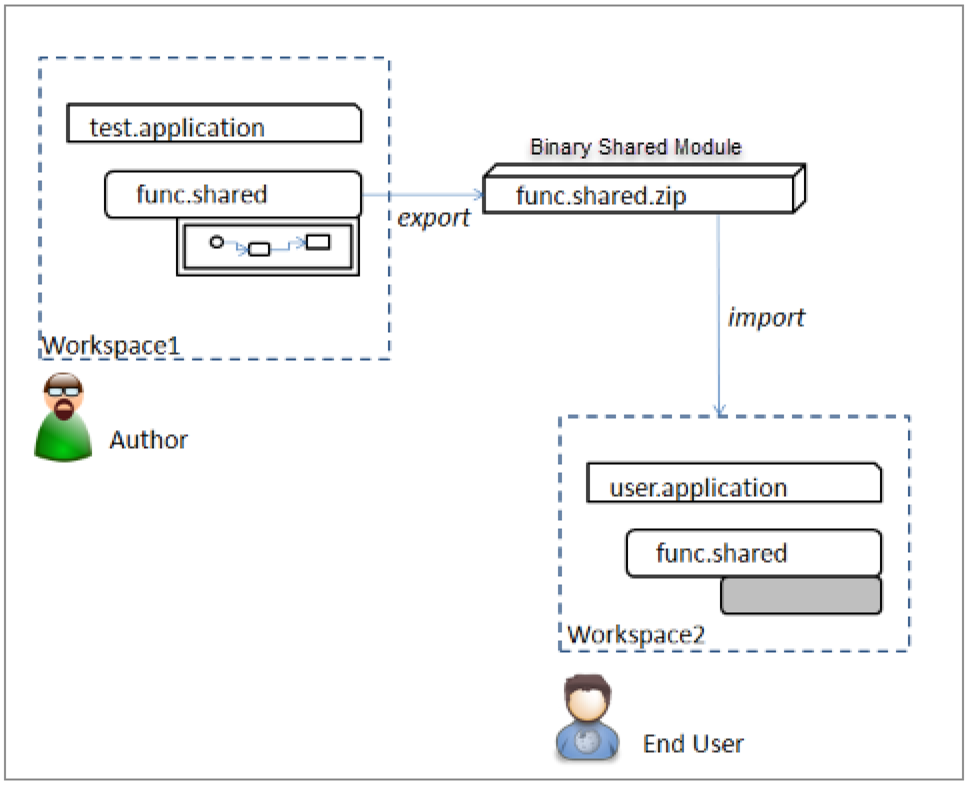
For information on shared modules that can be exported for the purpose of sharing them with other users, see "Binary Shared Modules" in *TIBCO BusinessWorks™ Container Edition Concepts*.

Binary Shared Modules

Binary shared modules are essentially shared modules that you create to hide the implementation details of a shared module from the consumers of the module. A binary shared module is compiled into a binary format for use by another application.

A binary shared module can be used like any other shared module, except its end user must use it as is without the ability to modify it in any way. When imported into a project, the end user cannot see its process diagram or the implementation details of other artifacts within the module.

Binary shared modules serve as a good vehicle when you have a standalone functionality to share without exposing its details.



For more information on creating and using a binary shared module, see "Creating a Binary Shared Module", and "Using a Binary Shared Module" in *TIBCO BusinessWorks™ Container Edition Application Development*.

Processes

Processes capture and describe the flow of business information in an enterprise between different data sources and destinations.

Processes comprise activities that accomplish tasks. The flow of data between activities in a process is represented using transitions, conditions, and mappings. TIBCO Business Studio for BusinessWorks provides design palettes containing activities and transitions that can be used to develop business processes.

Parent Process

A process can call another process, or a subprocess. The process that makes the call is referred to as a *caller process* or a *parent process*.

Subprocess

A subprocess can be called by a parent process, or another subprocess. In the case where a subprocess is calling another subprocess, the subprocess that makes the call is the *parent process*. The called process is referred to as a subprocess or a *child process*. At runtime, inline subprocesses are run on the same engine thread as the caller process while the non-inline subprocesses use different engine threads and are run on the new threads.

Component Process

The execution of a process is triggered by various events. Often the business logic that is designed to react to a particular event is spread across multiple processes. One of the processes is special and it reacts to the original event and triggers the execution of the other processes. This special process is referred to as the component process or main process. A component process is responsible for initiating the job at run time.

A component process is designed to react to various events and these events are triggered by Processes and Bindings.

Process Services

A process can provide services to other processes. A process service exposes the operations provided by the process and is implemented using a WSDL or a JSON file. When the process is implemented by a component, the process services are exposed as component services, which then need to be configured using bindings.

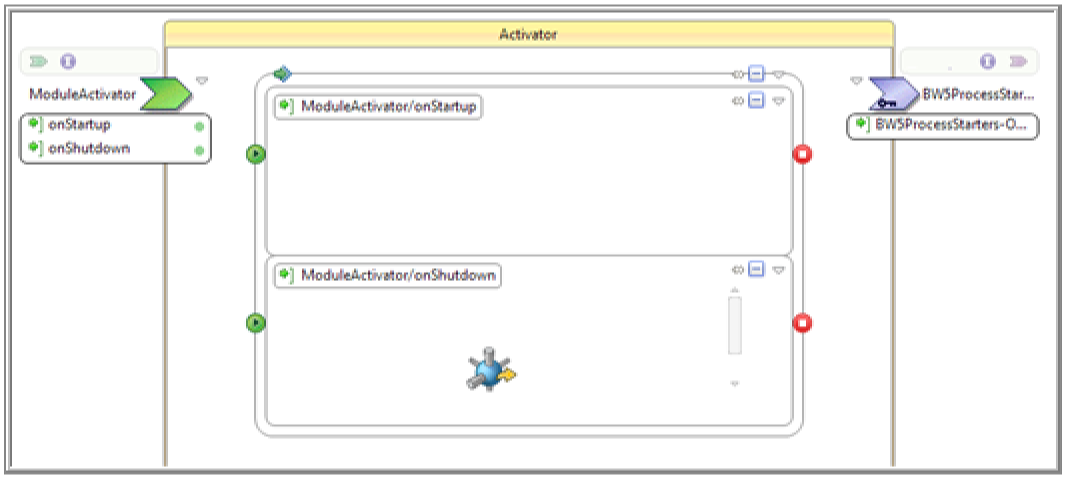
Process References

A process can consume services provided by other processes or by external service providers. A process reference exposes the operations consumed by the process and is implemented using a WSDL or a JSON file. A process reference can be configured to invoke a process or an external service.

When the process is implemented by a component, the process references that are not configured to call a process or an external service through a binding are exposed as component references, which then need to be configured using bindings.

Activator Process

An activator process is a special process that can be used to perform pre-processing and post-processing tasks when the application is started and stopped respectively. The activator process contains a process service with two operations: **OnStartup** and **OnShutDown**.



The **OnStartup** operation is called when an application is started, but before running any other processes in the application. The **OnStartup** operation can be used to implement any pre-processing tasks that must be performed for the application before the regular processing starts. For example, the **OnStartup** operation can be used to check if the database tables required by an application exist, and create them if they do not exist. If this process instance faults due to an unhandled exception, the application does not start.

The **OnShutDown** operation is called when an application is stopped, but after stopping and completing all other processes in the application. The **OnShutDown** operation can be used to implement any post-processing tasks that must be performed for the application after the regular processing is complete. For example, the **OnShutDown** operation can be used to send an email to administrators notifying them that the application is being stopped.

The activator process can only be configured for an application module. There can be only one activator process for an application module. However, the activator process can invoke one or more subprocesses.

For information on how to create an activator process, see "Creating an Activator Process" in *TIBCO BusinessWorks™ Container Edition Application Development*.

A simple business process can be developed by adding activities in sequence, and the connecting the activities using transitions with or without conditions. Developing a complex business process typically involves developing a component process and one or more subprocesses. Use of subprocesses makes the complex business process easier to understand and debug. At runtime, the in-line subprocesses do not create a new job, but are run on the job created by their calling process.

Note: For more information about the TIBCO Business Studio for BusinessWorks development environment, see Design-time Concepts.

Activities

Activities are the individual units of work in a process.

Activities generally interact with an external system and perform a task. Activities that perform similar tasks are grouped in an entity called a *palette*. TIBCO Business Studio for BusinessWorks provides various technology-specific palettes using which you can build a business process.

Each activity in a palette is represented by an icon. For example, the **JDBC Update** activity is represented by the icon. For example, the database update activity is represented by the icon. Often an activity icon is also decorated with an additional symbol such as a green or a yellow pause sign to indicate the activity waits for an event, an arrow to indicate the direction of the data flow. For example, the arrow sign in the JMS Send Message icon JMS Send Message indicates that data is being sent by this activity.

Note: For more information on palettes, see the *TIBCO BusinessWorks™ Container Edition Bindings and Palettes Reference* guide.

Activities can be classified into three types:

- **Regular Activities** perform a specific task. Regular activities can have input and output in addition to their configuration. Activities can also state the faults they can generate at runtime. This allows the process to be designed to handle these faults and perform the necessary actions. Regular activities can be further classified into synchronous and asynchronous activities.

  Synchronous activities are *blocking*. They block the execution of the process until the activity task completes.

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

- **Process Starter Activities** are configured to react to events. They trigger the execution of a process when the event occurs. Process starter activities can have only outputs in addition to their configuration. For example, the **HTTP Receiver** process starter activity starts a process when an HTTP request is received.

Note: For more information about the TIBCO Business Studio for BusinessWorks development environment, see Design-time Concepts.

Palettes

Palettes group activities that perform similar tasks. TIBCO Business Studio for BusinessWorks provides various technology-specific palettes that provide quick access to activities when building a process.

Palettes are typically located to the right of the **Process Editor** in TIBCO Business Studio for BusinessWorks. Depending on the process being designed and the stage of process development, you can focus on the activities available under appropriate palettes.

In TIBCO Business Studio for BusinessWorks, the **Palette** views display the list of activities contained in a palette and allow you to perform the following actions:

- Search for activities in palettes.

- Use multiple palettes and save them as grouped palette sets.
- Save palettes, or the grouped palette sets, as favorites.
- View recently used palettes.
- Create virtual palettes, which means that some activities can be taken from unrelated palettes. This activity is called a custom shortcut.

Note: For more information about the TIBCO Business Studio for BusinessWorks development environment, see Design-time Concepts.

Transitions

Transitions can be added to activities and groups in a process. They represent the flow of execution from one activity or group to another.

In TIBCO Business Studio for BusinessWorks, transitions are displayed as an arrow between two resources in a process. Transitions are unidirectional and cannot connect to a previously run activity or group. The control flow in a process must proceed sequentially, beginning with the starting activity or group and ending with the last activity or group in the process.

Transitions can have a one-to-many relationship with the activities. In a process, one activity can simultaneously transition to multiple activities or groups. For example, if the shipping schedule indicates a delay in shipping an order, you want to notify the customer and enter the information into the customer service system. However, if there is no delay, you want to enter the information into the customer service system without notifying the customer.

Transitions can fall into one of the following categories:

- **Transitions Without Conditions**: Control automatically flows from one activity or group to the next without any conditions.
- **Transitions With Conditions**: When an activity or group completes processing, the conditions specified on the transitions originating from that activity or group are evaluated to determine whether the transition to the next activity, or group should be taken or not. All transitions whose conditions are met are taken.
- **Error Transitions:** Special transitions that specify the activities or groups to run in

case of an error. When configuring an activity or group, you can select one transition to take from the specified activity or group, and the activities or groups to be run following the error transition.

Note: Error transitions take precedence over fault handlers. If an error is encountered in a scope and it has both a fault handler and an error transition, then the error transition is run.

Shared Resources

Shared resources are resources that contain common configuration data that can be referenced from multiple places.

You can define a shared resource and then reference it from multiple activities in the same or different process. For example, you can define a **JDBC Connection** resource and then use it in any of the **JDBC** activities in your process to connect to the database.

Shared resources such as JDBC Connection, JMS Connection, HTTP Connection, are available at design-time. At runtime, the referencing activities and event sources have full access to their instances and configuration.

Shared resources can be grouped in packages, similar to the way process packages and Java packages are presented in the file system.

When defined in an application module, shared resources are not visible to processes outside the application module. However, when defined in a shared module, they are visible to processes outside the shared module.

Shared Variables

Shared variables are used to define data for modules and jobs. There are two types of shared variables: job shared variables and module shared variables. They are stored separately.

Job Shared Variables

Job shared variables are used to share data within a job such as between a parent and child process instance. At runtime, the engine allocates a new variable for each job and the value of that variable is not visible outside the job to which it was allocated.

Module Shared Variables

Module shared variables are used to share data across all processes in a module. The module shared variable is visible to all process instances within the same module.

The key difference between a job shared and a module shared variable is that when jobs expand across module boundaries, a job shared variable is visible outside the module it was set in, while the module shared variable is visible only inside the module in which it was set.

Independent Start of a Component

The Independent start of a component feature allows the application to start even if there is component failure (due to shared resource failure) so that the user can use other working components in the application without any issue. For more information, see "Shared Resources" in the *TIBCO BusinessWorks Container Edition Bindings and Palettes Reference* guide.

Additional Concepts

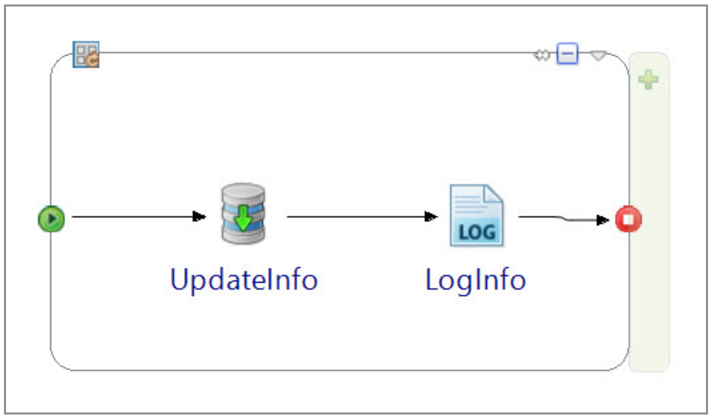
This section introduces the following additional concepts that help build complex business processes.

Groups

Groups consist of one or more activities that are assembled and run according to their type.

Groups enable you to put one or more activities together and configure the group as needed. For example, defining a single error condition for the group, or creating a group as a transaction that commits to a database only when all the activities in the group are completed.

Every group contains a **GroupStart** element on the left and a **GroupEnd** element on the right.



Groups can be classified into two categories: groups with conditions (repetitive groups) and groups without conditions (non-repetitive groups).

Groups Without Conditions (Non-repetitive)

The following types of groups do not require any conditions to be defined for their execution:

- **Scope** : A scope is a simple group that has no custom behavior. It can define local variables and can also contain fault handlers and event handlers. A scope with a single activity can be defined if you need to handle faults or catch exceptions specific to an individual activity.
- **Critical Section** : The Critical Section group is used to synchronize jobs so that only one job is acting on the group of activities at any given time. Any concurrently running job that contains a corresponding critical section waits until the job currently running the critical section completes. The Critical Section group is useful to control concurrent access to shared variables. While a critical section group can be used to synchronize jobs within a process, module shared variables help synchronize jobs for multiple processes.

Groups With Conditions (Repetitive)

Loops are groups with conditions that follow a pattern at runtime: initialize the loop, update the loop at each iteration, and test conditions for the loop to stop iterating. The following types of loops are available:

- **For Each** : For Each is used to loop for a specific number of iterations with a counter ranging from a start value to an end value.
- **Iterate** : This loop has a simple index variable that can be used to count each iteration and the loop runs for the number of iterations specified.
- **Repeat** : This loop has a simple index variable that can be used to count each iteration and has a conditional expression to determine when to stop. The loop runs at least once and a test for the specified condition is performed at the end of the loop. The Repeat loop continues to run until the condition evaluates to true.
- **Repeat on Error** : This loop involves a retry mechanism: if any activity in the loop displays a fault, the condition expression is evaluated to determine if the loop should be repeated. An index allows the condition to be based on the number of previous attempts, but any condition expression may be used.
- **While** : This loop has a simple index variable that can be used to count each iteration and has a conditional expression to determine when to stop. The condition for the While loop is tested at the beginning of each iteration and the loop may never be run if the condition is initially false. The While loop and continues to run as long as the condition holds true and stops when the condition evaluates to false.

Note: For more information about the TIBCO Business Studio for BusinessWorks development environment, see Design-time Concepts.

Properties

Properties are used to define configuration. Depending on where and how they are defined and qualified, properties can be classified into application properties, module properties, shared module properties, and process properties. The values for all three kinds of properties can be of one of the six primitive types (Boolean, Integer, DateTime, Long, Password, or String) or one of the available default shared resource type. These values are static and cannot be changed once an application has started execution. These values can only be changed at design time or deployment time.

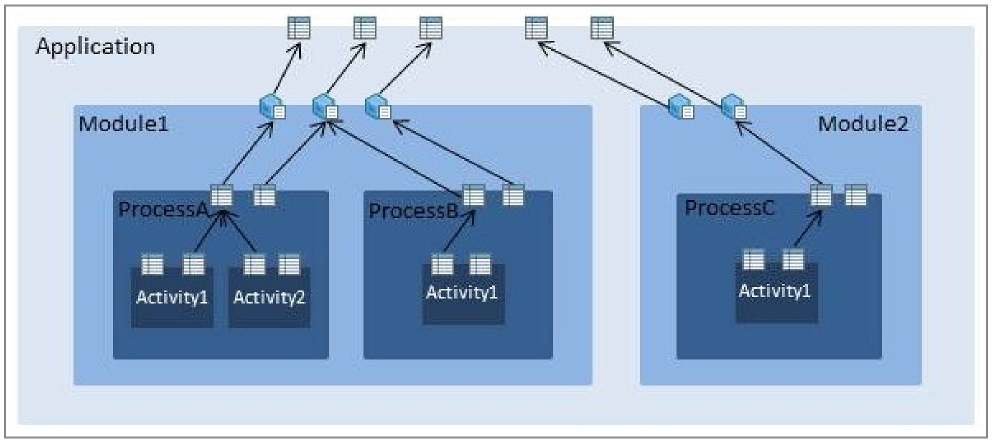
The three levels of properties are hierarchical: application properties are in the outer most scope, followed by module properties, followed by process properties.

Properties defined in the inner layer can reference a property defined at its parent layer. For example, a process property can reference a module property instead of providing a literal value. Similarly, a module property value can be defined by literal values or source from its parent scope application property.

Any process property or module property that you define is available both in the activity configuration page and is also available to use as an input to an activity (from the **Data Source** tab of the **Input** tab for the activity).

The following diagram illustrates the relationship between the different types of properties:

Relationship Between Properties



Features of Process, Module, Shared Module, and Application Properties

Property	Scope or Visibility	Values	Additional Information
Process Properties	Visible within a process.	Literal, module property reference, or a shared resource reference.	Literal values cannot be modified at the module or application level.
Module Properties	• Visible within the module.	• Literal or a shared resource reference.	Cannot be assigned to an activity directly. You need to reference a module property from a process property, and then reference the process property from the activity.
Shared Module Properties	• Visible within the module. • Visible within projects that contain dependencies to the Shared Module that the Shared Module Property came from. • Private module properties cannot be viewed from the Admin UI. • Not visible or changeable from the Admin UI.	• Literal or a shared resource reference. • Private module property values cannot be edited from the Admin UI.	• Shared Module Properties are module properties that come from a Shared Module. • Cannot be assigned to an activity directly. You need to reference a module property from a process property, and then reference the process property from the activity. • Can be used for activities, process properties, shared resources, and SOAP Bindings.
Application Properties	• Only available for an	• Literal.	• Overrides module properties, thus

Property	Scope or Visibility	Values	Additional Information
	application and visible within the application.	• Profiles can be used to specify a new set of values for the same application.	enabling you to use different values for the same module. • Cannot add new properties at application level.

Note: For more information about the TIBCO Business Studio for BusinessWorks development environment, see Design-time Concepts.

Shared Variables

Shared variables are used to save the state, either at the module level or for the duration of a job.

Using shared variables, you can share data across process instances associated with a module or a job. A process instance can read or update the data stored in a shared variable. The shared variable data updated by one process instance is accessible to other process instances of a Module or Job.

There are two types of shared variables: module shared variables and job shared variables. Both module and job shared variables are defined at the module level and can be accessed in a process using the activities **Set Shared Variable** and **Get Shared Variable**.

For more information on how to define and use shared variables, see "Using Shared Variables" in *TIBCO BusinessWorks™ Container Edition Application Development*.

Module Shared Variables

Module shared variables are used to share the state at a module level and are visible to all process instances created from the processes that are within a module. Module shared variables can be read and updated by the process instances during execution. Once the value is updated, the new value is available to all the process instances created from the processes that are within the module. Consider an example where the exchange rates are updated daily and the updated exchange rates should be accessible to all processes in a module. You can create a module shared variable to hold the exchange rate and use one process in the module for updating the exchange rate. All other processes in the module that require the exchange rate can retrieve the current value through the module shared variable.

Job Shared Variables

Job shared variables are used to share the state at the job level for the duration of a job. A copy of the job shared variable is created for each new job and it is accessible to all process instances associated with the job. Job shared variables can be used to share data between all process instances of a job without creating an input or output schema for the called process.

Sharing Job Shared Variables Between Inline and Non-Inline Processes

Inline subprocesses are run as part of the caller (parent) process jobs and therefore the current value of the job shared variable is passed from the caller process to the inline subprocess. Non-inline service subprocesses, and direct subprocesses that have been spawned, spawn a new thread, and are not run on the same job as the caller process. Hence the non-inline subprocess and direct subprocesses that have been spawned, obtain a copy of the job shared variable, and do not obtain the current value of the job shared variable from the caller process.

At runtime, the engine allocates a new job shared variable for each new job and the value of that variable is visible only to that job.

Shared Variable Synchronization

Multiple process instances can potentially access or update a shared variable at the same time. For example, a module shared variable can be accessed by multiple jobs concurrently. Without a synchronization mechanism, a process instance could update the value of a shared variable while another process instance is trying to read the value. This could result in an unpredictable value for the shared variable.

Critical Section groups can be used to synchronize access to shared variables. A **Critical Section** group allows only one process instance to run the **Critical Section** group and its contents at any given time. To synchronize shared variables, 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 running 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. To synchronize multiple critical section groups, use a shared lock. The shared lock can be defined using a module or a job shared variable.

Fault Handlers

Errors (or faults) can occur when running a process. You can use Fault handlers to catch faults or exceptions and create fault-handling procedures to deal with potential runtime errors in your process definitions.

It is a best practice to use Fault handlers to catch faults or exceptions in a process. Two types of fault handlers are available: **Catch Specific Fault** and **Catch All Faults**.

Fault handlers can be defined at two different levels:

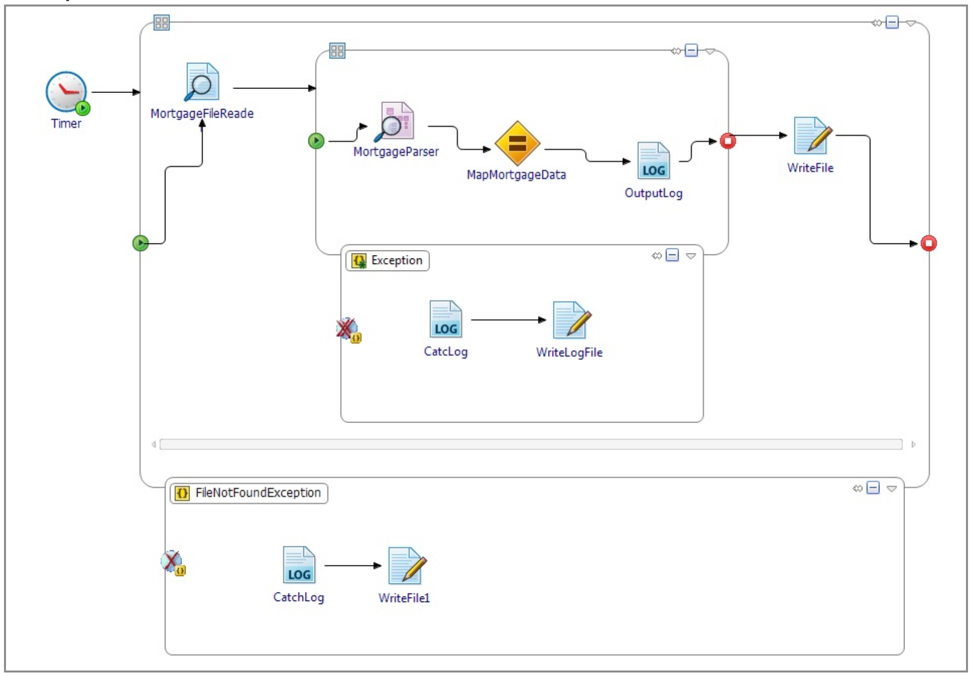
- Process level - When defined at the process level, allows you to catch fault in a process.
- Scope level - When defined at the scope level, allows you to catch fault within a scope.

Fault handlers when defined at the scope level, allows you to catch faults or exceptions generated by activities within a scope. To catch faults or exceptions specific to an individual activity, you need to define a new scope for that individual activity and attach a fault handler to the new scope.

At runtime, once a fault handler is run, the associated scope is not completed due to the error generated. If a fault is not generated in the fault handler, the process execution continues with the first activity that follows the scope. If a fault is generated in the fault handler, then the engine looks for an enclosing scope that is designed to handle the fault. If one is found, the engine runs it. Once the enclosing fault handler finishes its execution, the engine runs the next activity following the scope. If no fault handlers are found in the enclosing scopes, then the job ends with a fault.

Consider the fault handlers defined in the sample process.

Sample Fault Handlers



If an exception is caught in the inner scope, the exception is logged to a file, and the scope is completed. The process execution then continues to the **Write File** activity, which is the next activity in the process. If an exception is caught in the outer scope, the exception is logged and the scope is completed. The process execution completes successfully as there are no following activities to be processed. An **Exit** activity inside the fault handler returns the control out of the scope and the process.

Error Transitions can also be used to handle error conditions by using them to specify a transition to take in case of an error.

Note: Error transitions take precedence over fault handlers. If an error is encountered in a scope and it has both a fault handler and an error transition, then the error transition is executed.

Components

Components implement a process and provide information to the runtime on how to represent the process.

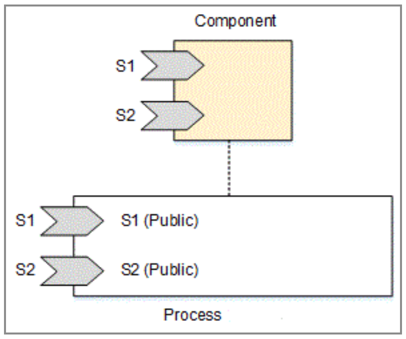
Components are generated only for the main processes and each main process initialized by the engine must have a component associated with it. Components are required only by the main processes that are responsible to initiate the business logic. Subprocesses do not require components as they are called by another parent process.

Component Services

Component services describe the binding information to receive an invocation from an external consumer.

When a component implements a process that has a service, the process service is exposed as a component service. The component service then must be configured using bindings such as SOAP and REST.

The service-centric architecture supports self-contained services. Each service is configured separately and can be deployed on a different machine. If one machine goes down, all other parts of the process can continue to run. This loosely coupled architecture makes it easy to change individual components as needed.



Component References

Component references describe the binding information required to invoke an external service.

When the component implements a process that has a reference, then the process reference is exposed as a component reference. When configuring to invoke an external service, the binding information that contains protocol details is not part of the process. The service consumer needs to create a component that is an implementation of that process and configure the binding along with protocol details. The **Invoke** operation activity or a reference can be used to invoke a service.

References have the following characteristics:

- They can be public or private. Public references are visible from outside of the process.
- They always reference one interface or port type.

Based on the availability of the target service name at design-time, you can use either static references or dynamic references. Static references can be used when the target service name is available at design-time and dynamic references are available when the target service name is not available at design-time. This applies to target services developed as a part of TIBCO BusinessWorks Container Edition as well as external target services.

Services

TIBCO BusinessWorks Container Edition can function both as a server and a client in a web services interaction. Services and references are defined at the process level while the bindings are created at the component level.

The supported service classes are:

- REST (Representational State Transfer)- compliant services, where the primary purpose of the service is to manipulate XML representations of web resources using a uniform set of stateless operations. When using a stateless operation, the state is managed by the job itself instead of by the engine.
- SOAP services, which are used for exchanging information in the implementation of web services relying on XML message format sent over HTTP and JMS.

Web services are typically associated with the following characteristics:

- **Interfaces** that describe the operations available within a service. An interface is analogous to a port type in a WSDL file. Each interface can contain multiple operations.
- **Operations** define an action that can be performed by the service and the way the message is encoded.
- **Transport** used for communication such as HTTP or JMS.
- **Schema** used for message exchanges such as XSD.

Operations

Operations define the action that can be performed by the process. Multiple operations are supported in a process with multiple inputs, outputs, and faults.

There are two types of message exchange operations: one-way operations and request-response operations.

SOAP Services

SOAP services are web services that use SOAP as the standard communication protocol for XML-based message exchanges.

The standard HTTP protocol makes it easier for the SOAP model to tunnel across firewalls and proxies without any modifications to the SOAP protocol.

- The Web Services Description Language (WSDL) contains and describes the common set of rules to define the messages, bindings, operations, and location of the Web service. A WSDL file is a formal contract to define the interface that the Web service offers.
- SOAP services require less coding than when designing REST services. For example, transactions, security, coordination, addressing, and trust are defined by the WSDL specification. Most real-world applications are not simple and support complex operations, which require conversational state and contextual information to be maintained. Application developers do not need to worry about writing this code into the application layer themselves.
- SOAP supports several technologies, including WSDL and XSD.

REST Services

Representational State Transfer (REST) is an architectural style of the World Wide Web that is used in building services for distributed systems and networked applications. RESTful APIs are increasingly preferred for enterprise, web, and mobile integration use cases.

The key abstraction of information in REST is a resource, with focus on components, the constraints on their interaction with other components, and their interpretation of significant data elements. REST ignores the details of component implementation and protocol syntax.

The supported features of the REST architectural style are:

- **Client-server architecture**: Provides a separation of implementation details between clients and servers.
- **Stateless communication**: Ensures that each request contains all of the information required to understand it independently of any stored context on the server.
- **Cacheability**: Provides an option to the client to cache response data and reuse it later for equivalent requests, thus partially eliminating some client-server interactions. This results in improved scalability and performance.

TIBCO BusinessWorks Container Edition currently allows the following HTTP operations 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, key-value parameters, and query parameters.

Policies

A policy is a set of constraints that you can define and apply in TIBCO Business Studio for BusinessWorks to manage and enforce cross-functional requirements within your TIBCO BusinessWorks Container Edition application such as security, monitoring, and compliance.

You can add policies to activities and bindings in a process to influence or alter actions in the process flow. For example, you can add a policy on an existing **HTTP Receiver** activity in your application to ensure that user credentials are authenticated, or verified as correct, before the message can continue moving through the process flow. Any request messages that cannot be authenticated are rejected, redirected, or handled in accordance to policy details.

The following policies are examples of policies provided in TIBCO BusinessWorks Container Edition:

Basic Authentication

Validates the username and password credentials stored in the HTTP header of REST, SOAP, or pure HTTP request messages.

Basic Credential Mapping

Automatically attaches appropriate credentials to request messages before they reach services.

Web Services Security Provider (WSS Provider)

Acts on the server side to ensure the security of a message by enforcing confidentiality, integrity, and time stamping.

Web Services Security Consumer (WSS Consumer)

Acts on the reference side to ensure the security of a message by enforcing confidentiality, integrity, and time stamping.

Policy Definitions and Concepts

The following definitions and concepts are used to describe policies and policy management.

Policy

A policy is set of constraints that you can define and apply in TIBCO Business Studio for BusinessWorks to manage and enforce cross-functional requirements within your application such as security, monitoring, and compliance. You can add policies to activities and bindings in a process to influence or alter actions in the process flow.

Policy Types

Policies that are related or perform similar functions are categorized under policy types. Policies that can be applied to the HTTP layer of SOAP, REST, and pure HTTP services are categorized under the HTTP Security policy type. Policies that can be applied to the SOAP layer are categorized under the SOAP Security policy type.

Activities

An activity is the individual unit of work in a process. You can add policies to activities to influence or alter actions in a process flow.

For more information about activities, see "Application Development" in *TIBCO BusinessWorks™ Container Edition Getting Started*.

Bindings

A binding is used to establish a connection between SOA Services and their consumers. There are two types of binding components:

- Service Binding, which is used to create and expose a service to the external world. The service can contain one or more operations. Once exposed, the service can be consumed by its clients.
- Reference Binding, which is used to create a client that can connect and communicate to an external service.

You can add policies to bindings to manage, modify, and secure message exchanges on the consumer side and provider side.

For more information about the types of bindings offered in the workspace, see "Binding" in *TIBCO BusinessWorks™ Container Edition Concepts*.

Policy Association

When you add a policy on an activity or a binding, the relationship you create between the resources is called a policy association. At runtime, policies are enforced on the activities and their associated bindings.

Shared Resources

Policies reference shared resources. You can manage and configure shared resources in your workspace. The following table describes shared resources that each policy might reference.

Policy	Shared Resource
Basic Authentication	• XML Authentication
Basic Credential Mapping	• Identity Provider
WSS Provider	• Subject Provider • Keystore Provider

Policy	Shared Resource
	• Trust Provider • WSS Authentication
WSS Consumer	• Identity Provider • Keystore Provider • Trust Provider • Subject Provider • WSS Authentication

Note: You can define a shared resource and then reference it from a single policy or multiple policies. For example, you could use a single **Keystore** resource in the WSS Provider policy and the WSS Consumer policy.

Governance Agent

The governance agent is a TIBCO BusinessWorks Container Edition run time component that dynamically enforces policies during runtime. A governance agent must be enabled on an AppNode to enforce policies applied to TIBCO BusinessWorks Container Edition applications.

For instructions on enabling the governance agent, see "Enabling the Governance Agent" in *TIBCO BusinessWorks™ Container Edition Administration*.

Mapping Concepts to a Sample: Managing Books for a Bookstore

The concepts introduced in sections General Concepts and Additional General Concepts enable you to understand and design a resource-oriented solution such as the Bookstore example.

Mapped Concepts of the Bookstore Sample

The Bookstore sample requires the following concepts:

- General Concepts
- Additional General Concepts

After going through these sections, you should be able to understand and run a resource-oriented solution such as the sample to manage books for a bookstore.

Bookstore Sample

The bookstore sample uses a RESTful service to add, delete, update, and retrieve books from the bookstore. The following REST methods are used:

- POST - Posts books to the bookstore
- GET - Get books from the bookstore
- PUT - Updates books to the bookstore
- DELETE - Deletes books from the bookstore

The Bookstore sample project is shipped with the product and can be accessed in TIBCO Business Studio for BusinessWorks from **Help > BusinessWorks Samples**.

Next Steps

After completing this section, you should be able to design resource-oriented processes with minimal assistance.

Design Time Concepts

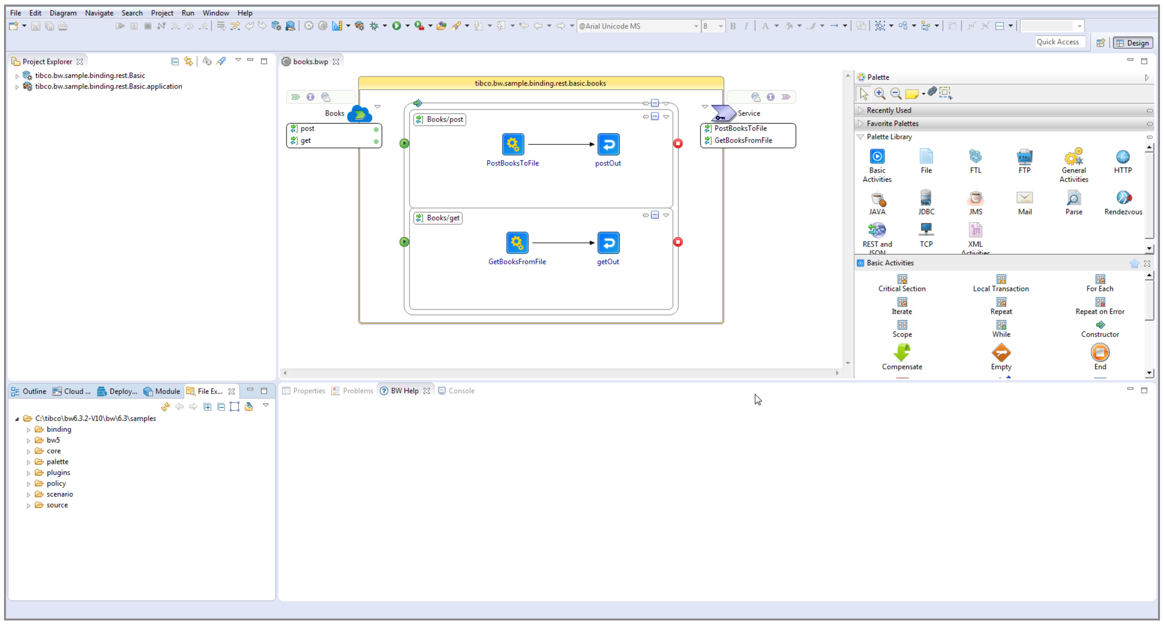
Design time concepts introduce TIBCO Business Studio for BusinessWorks, an Eclipse-based integration development environment that is used to design, test, and deploy applications.

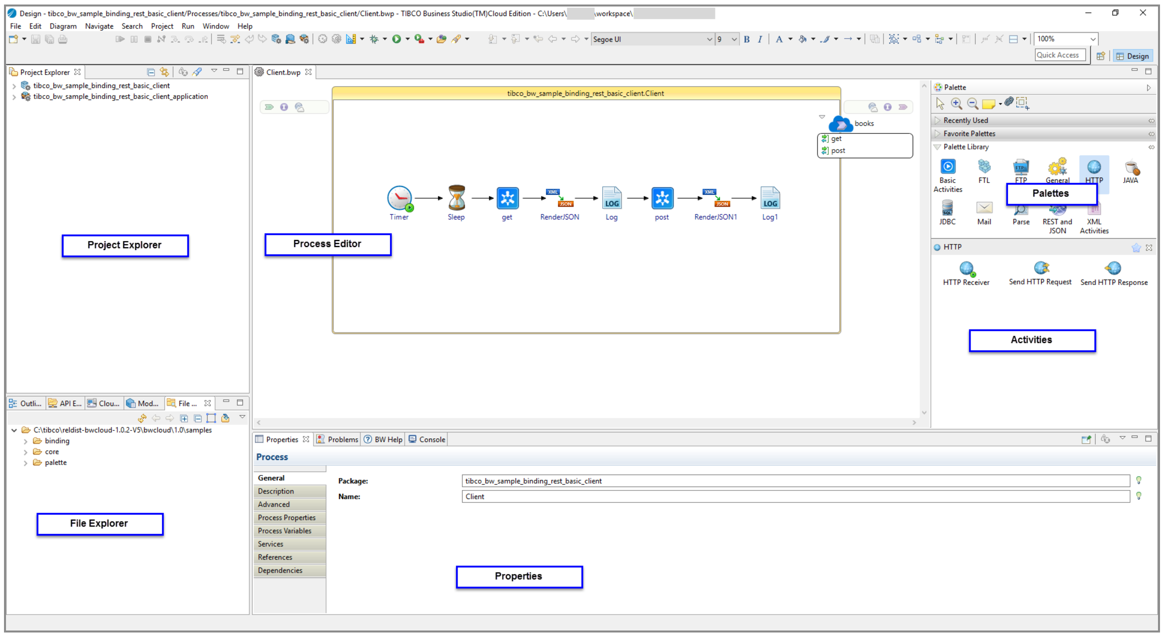
TIBCO Business Studio for BusinessWorks provides Eclipse extensions such as editors, palettes.

Development Environment

TIBCO Business Studio for BusinessWorks provides a workbench that can be used to create, manage, and navigate resources in your workspace. A *workspace* is the central location on your machine where all the data files are stored.

TIBCO Business Studio for BusinessWorks Workbench





Process

Package: tibco_bw_sample_binding_rest_basic_client

Name: Client

The workbench consists of the following attributes:

Attribute	Description
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 Application Module**, **Create a new Shared Module**, **Debug**, **Run**, and so on.
Perspectives	Contain an initial set and layout of views that are needed to perform a certain task. TIBCO Business Studio for BusinessWorks opens the **Design** perspective by default. You can change the perspective from the menu **Window > Open Perspective >** *<perspective_name>*.
Views	Display resources and allow for navigation in the workbench. For example, the **Project Explorer** view displays the applications, modules, and other resources in your workspace, and the **Properties** view displays the properties for the selected resource. You can open a view from the menu

Attribute	Description
	Window > Show View > ***<view_name>***.
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 a process (`MortgageAppConsumer.bwp`) in the **Project Explorer** view to open the process in the editor.
Palettes	Palettes group activities that perform similar tasks and provide quick access to activities when building a process. For more information, see Palettes.

Explorers

TIBCO Business Studio for BusinessWorks consists of the following tabs in its left pane:

- **Project Explorer**: Displays the logical view of your entire workspace with all the projects and the processes, service descriptors, resources, schemas, and module descriptors for each project.
- **API Explorer**: You can also view the APIs residing locally on your machine from the API Explorer. Use the Settings dialog in the API Explorer to filter the APIs you want to access.
- **File Explorer**: Displays a view of selected folders in your local file system.
- **Outline** tab: Displays a tree structure of the details of selected artifacts in an editor.
- **Module** tab: Displays the module properties and shared variables used in the module.

Application Testing and Debugging

TIBCO Business Studio for BusinessWorks bundles some of the runtime components so that you can run and debug an application in the design-time environment.

The menu option **Run > Debug** or the icon on the tool bar enables you to debug an application. The menu option **Run > Run** or the icon on the tool bar enables you to run an application.

The Run configurations specify information such as:

- Bundles to be run.
- Arguments such as the target operating system, target architecture, target web services.
- 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 choosing to save the results either as local files or as shared files, and also to display them in the menus (**Debug** and/or **Run**). It also allows you to define encoding for the result files.

Once created, an application can be run using a specific configuration. If a run configuration is not specified, the project displayed in the editor area is launched by default.

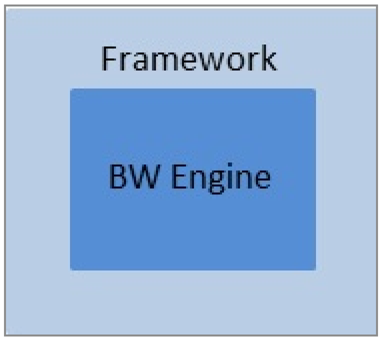
Runtime Concepts

Runtime refers to the AppNode and the TIBCO BusinessWorks Container Edition engine that host and run TIBCO BusinessWorks Container Edition applications.

AppNode

An AppNode (also called *BWAppNode*) is an operating system process (JVM) that hosts and runs TIBCO BusinessWorks Container Edition applications. An AppNode consists of two key layers: the OSGI Framework and TIBCO BusinessWorks Container Edition Engine. The high-level architecture of an AppNode is shown in the following figure:

Application Node Architecture



The framework layer performs application life-cycle operations, ensures that dependencies required by the application are satisfied. The engine layer is responsible for running the application. The engine is multi-threaded and can run multiple jobs for the same or different applications concurrently.

At runtime, an AppNode opens the framework to validate and identify dependencies. After the framework validates the modules and the application is deployed, the TIBCO BusinessWorks Container Edition engine starts the underlying processes.

The binary file named `bwappnode` is packaged under the *`TIBCO_HOME`*`/bw/`*`version`*`/bin` directory.

AppNode-Level Engine Properties

List of properties

Property	Use case
`bw.engine.enable.loop.reset=true`	To enable the reset variables used in the loops.
`bw.independent.component.startup=true`	The property allows the application to run even when one of the components fails due to an incorrect configuration of the shared resource. The rest of the components can be started after using this property.
`bw.process.deserialization.in.parallel=true`	This property when set to true helps to reduce the deserialized time of TIBCO BusinessWorks Container Edition applications. It has a major impact on applications having a large number of processes.
`bw.frwk.version.format`	This property enables you to print the `major.minor.patch` application versions in logs. The version format can be set using a system property in the following formats: • `bw.frwk.version.format=major.minor` • `bw.frwk.version.format=major.minor.micro` • `bw.frwk.version.format=major.minor.micro.qualifier`

Process Instance

When you run any process, it creates an execution scope for the activities that are a part of the process and this scope is called a *process instance*. Each process instance has a unique ID, which is referred to as "ProcessInstanceId".

The execution of a process is triggered by various events. For example, events can be generated by a Timer that is scheduled to trigger at specific time intervals, or by changes that occur in the file system, or by messages that are sent by a client over a specific protocol (for example, HTTP, JMS), or simply by messages sent by other processes.

The TIBCO BusinessWorks Container Edition engine is a multi-threaded engine capable of triggering the execution of the same process multiple times, concurrently, once for each event. When the events that trigger the execution of a process occur concurrently, the engine runs the same process multiple times, concurrently, once for each event. And for each execution, the engine creates a process instance that provides an execution scope for the activities that are a part of the process.

Job

Execution of a component process is called a *job*. Each job has a unique id referred to as `JobId`.

When the business logic is spread across multiple processes, multiple process instances are created and run with a particular event. Even though these are separate process instances that work together and can be run as part of the same job. A job can spawn multiple process instances and can provide the execution context for activities that are part of multiple processes. The engine always runs a job in one engine thread.

All the process instances that are part of the same job have the same JobId. A component process instance and all of its in-line subprocess instances are also considered being a part of the same job. Non in-line subprocesses spawn a new engine thread and are run on a different job.

Enabling and Disabling Auditing Events

The auditing is used to build metrics and statistics for an application.

Configure the following BWEngine property in the `config.ini` file to enable and disable auditing at AppSpace and AppNode level.

- `bw.engine.disable.auditevent`: By default, the value of the property is false.

  Note: Restart the AppNode after configuring the property.

Enabling and Disabling Input and Output Data for Audit Events

Configure the following BWEngine property in the `config.ini` file to enable and disable input and output data for audit events at an AppSpace and an AppNode level:

- `bw.engine.enable.activity.input.output.data.for.audit.events`: By default, the value of the property is false.

  > **Note:** Restart the AppNode after configuring these properties.

When using Statistics collection, the input and output data storage can also be disabled or enabled for audit events using the following BWEngine REST APIs:

- To disable:

  ```
  http://<host>:<appnode
  port>/bwm/monitor.json/disableinputoutputdataforauditevents
  ```

- To enable:

  ```
  http://<host>:<appnode
  port>/bwm/monitor.json/enableinputoutputdataforauditevents
  ```

TIBCO Documentation and Support Services

For information about this product, you can read the documentation, contact TIBCO Support, and join TIBCO Community.

How to Access TIBCO Documentation

Documentation for TIBCO products is available on the Product Documentation website, mainly in HTML and PDF formats.

The Product Documentation website is updated frequently and is more current than any other documentation included with the product.

Product-Specific Documentation

The following documentation for this product is available on the TIBCO BusinessWorks™ Container Edition page:

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

- *TIBCO BusinessWorks™ Container Edition Samples*

How to Contact Support for TIBCO Products

You can contact the Support team in the following ways:

- To access the Support Knowledge Base and getting personalized content about products you are interested in, visit our product Support website.
- To create a Support case, you must have a valid maintenance or support contract with a Cloud Software Group entity. You also need a username and password to log in to the product Support website. If you do not have a username, you can request one by clicking **Register** on the website.

How to Join TIBCO Community

TIBCO Community is the official channel for TIBCO customers, partners, and employee subject matter experts to share and access their collective experience. TIBCO Community offers access to Q&A forums, product wikis, and best practices. It also offers access to extensions, adapters, solution accelerators, and tools that extend and enable customers to gain full value from TIBCO products. In addition, users can submit and vote on feature requests from within the TIBCO Ideas Portal. For a free registration, go to TIBCO Community.

Legal and Third-Party Notices

Cloud SG software may be available on multiple operating systems. However, not all operating system platforms for a specific software version are released at the same time. See the “readme” file for the availability of a specific version of Cloud SG software on a specific operating system platform.