



TIBCO ActiveMatrix BusinessWorks™

Concepts

*Version 6.8.1
May 2022*



Contents

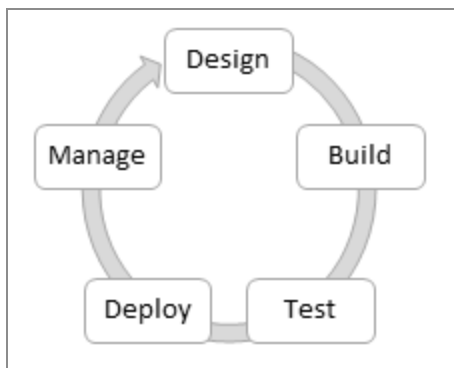
Contents	2
Overview	4
Key Concepts	5
Layout of the Concepts Guide	7
General Concepts	9
Applications	9
Modules	10
Application Modules	11
Shared Modules	13
Binary Shared Modules	14
Processes	15
Process Packages	20
Activities	20
Palettes	21
Transitions	22
Shared Resources	23
Mapping Concepts to a Sample: File Poller	25
Additional General Concepts	27
Groups	27
Properties	29
Shared Variables	31
Conversations	34
Event Handlers	35
Fault Handlers	36
Module Property	38

Components	39
Component Services	39
Component References	39
Services	40
Operations	41
SOAP Services	41
REST Services	42
Policies	42
Policy Definitions and Concepts	43
Mapping Concepts to a Sample: Mortgage Broker Service and Client	46
Mapping Concepts to a Sample: Managing Books for a Bookstore	48
Design-time Concepts	49
Runtime Concepts	53
Administration Concepts	55
Application Archives	57
Domains	58
AppSpaces	59
AppNodes	60
bwagent	63
TIBCO Documentation and Support Services	65
Legal and Third-Party Notices	67

Overview

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

With this software you can create services and integrate applications using a visual, model-driven development environment, and then deploy them in the ActiveMatrix BusinessWorks™ run time environment.



It 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. The deployable artifacts can be deployed and run in the run time environment, and managed using an administration interface such as TIBCO™ Enterprise Administrator.

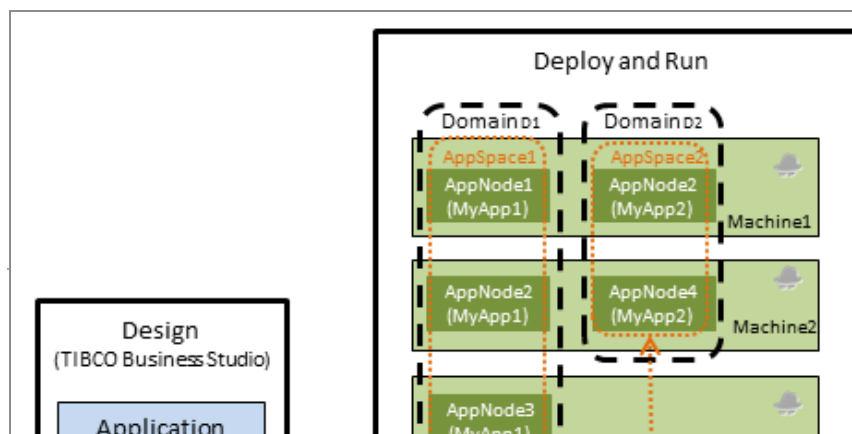
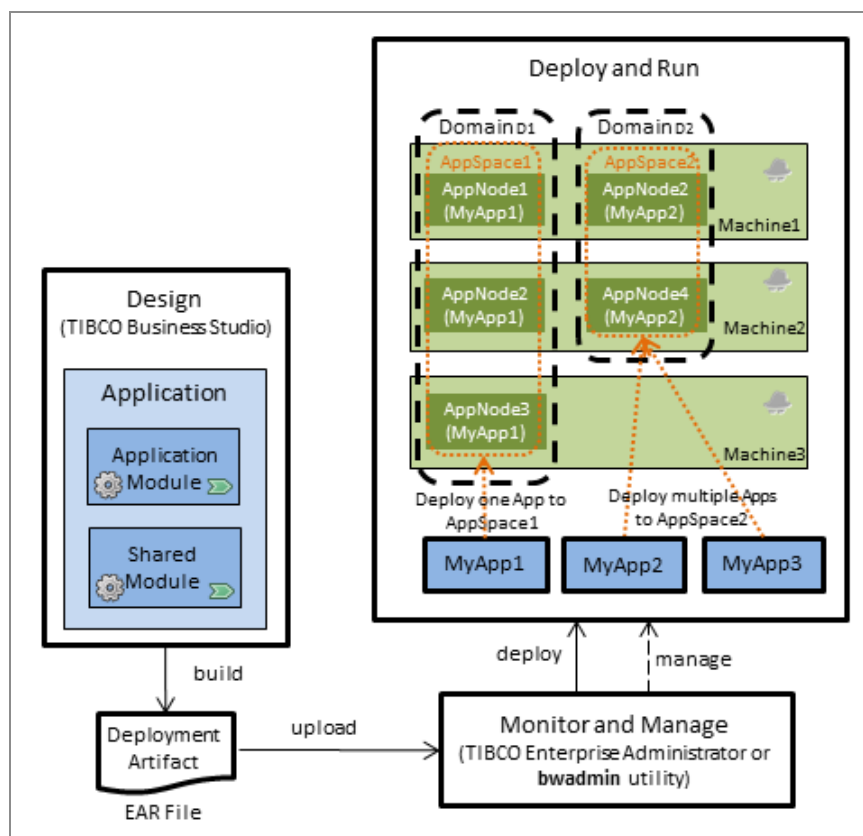
With TIBCO Business Studio for BusinessWorks, you can model integration processes of varying complexity using any of the following integration styles:

- **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 and administration perspective, while some are applicable to both perspectives. ActiveMatrix BusinessWorks™ consists of a design-time where you can develop applications that implement business logic. The runtime environment exists on the cloud.

ActiveMatrix BusinessWorks consists of a design-time where you can develop applications that implement business logic, the runtime where you execute the applications, and the administration component where you deploy and manage applications in the runtime. The runtime is an ecosystem of entities that can be co-located or distributed. You can deploy, monitor, and manage the applications by using the bwadmin utility and TIBCO® Enterprise Administrator.



ActiveMatrix BusinessWorks is based on open architecture, flexibility, modularity, and support for standards.

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.
- Ability to manage the process state that is maintained across invocations either by the run time container (process engine) or by the process implementation.
- 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.
- Extensible administration framework based on TIBCO® Enterprise Administrator.

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 and REST, WSDL, HTTP, HTTPS, JMS, JDBC
- Data representation and transformation: Native support for XML, XSD, XPath, JSON, XSLT
- TIBCO Products: TIBCO Rendezvous™, TIBCO Enterprise Message Service™ (EMS), TIBCO AE Schema
- TIBCO Products: TIBCO Enterprise Message Service™ (EMS), TIBCO AE Schema
- Others: FTP, JNDI, SMTP, TCP

Layout of the Concepts Guide

This guide presents the design-time, runtime, and administration concepts that are useful to developers and administrators. These concepts are described in the following sections:

- **General Concepts:** Explains the essential concepts such as applications, application modules, shared modules, processes, activities, transitions, and shared resources.
- **General Concepts:** Explains the essential concepts such as applications, application modules, processes, activities, transitions, and shared resources.
- **Additional General Concepts:** Explains additional concepts that can be used when developing applications such as groups, properties, services, components, and event handlers.
- **Design-time Concepts:** Introduces the design-time environment, TIBCO Business Studio for BusinessWorks.
- **Runtime Concepts:** Explains the runtime concepts such as AppNodes, process instances, and jobs.

- **Administration Concepts:** Explains the administration concepts such as domains, AppSpaces, and AppNodes, that are useful to run and monitor the ActiveMatrix BusinessWorks applications.

Sections that map concepts to bundled samples aim to enhance your understanding of the concepts by mapping them to ready samples that can be viewed and executed.

General Concepts

ActiveMatrix BusinessWorks 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 these general concepts is essential to both developers and administrators.

ActiveMatrix BusinessWorks 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. An application contains one application module which consists of one or more processes. 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 executed in the runtime. Applications are developed using TIBCO Business Studio for BusinessWorks.

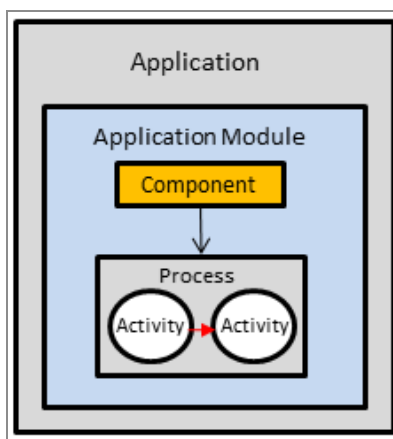
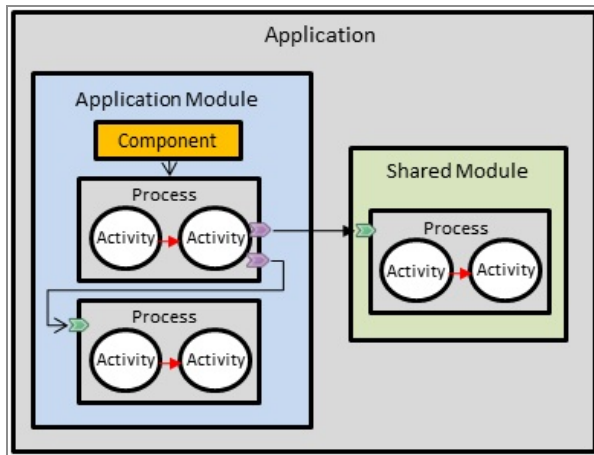
Applications are developed using features available in the product and can range from simple to very complex. An application contains one application module (see [Application Modules](#)), which in turn consists of one or more processes that define the business logic, and zero or more shared modules (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 OSGi bundle.

Elements of an application



Once 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 ActiveMatrix BusinessWorks.

Two types of modules are supported:

- **Application modules:** The smallest resource that is named, versioned, and packaged as part of an application and is executed in the ActiveMatrix BusinessWorks runtime. An application module cannot be deployed by itself in the ActiveMatrix BusinessWorks 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.

An **Application module** is the smallest unit of resources that is named, versioned, and packaged as part of an application and is executed in the ActiveMatrix BusinessWorks runtime. An application module cannot be deployed by itself in the ActiveMatrix BusinessWorks runtime; it must be packaged as part of an application.

Application Modules

The smallest resource that is named, versioned, and packaged as part of an application and is executed in the ActiveMatrix BusinessWorks runtime. An application module cannot be deployed by itself in the ActiveMatrix BusinessWorks 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, and so on.

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, interface, 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](#).
- **Resources:** Resources are reusable configuration data that can be shared within an application.
- **Schemas:** Schemas define elements and attributes which 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.



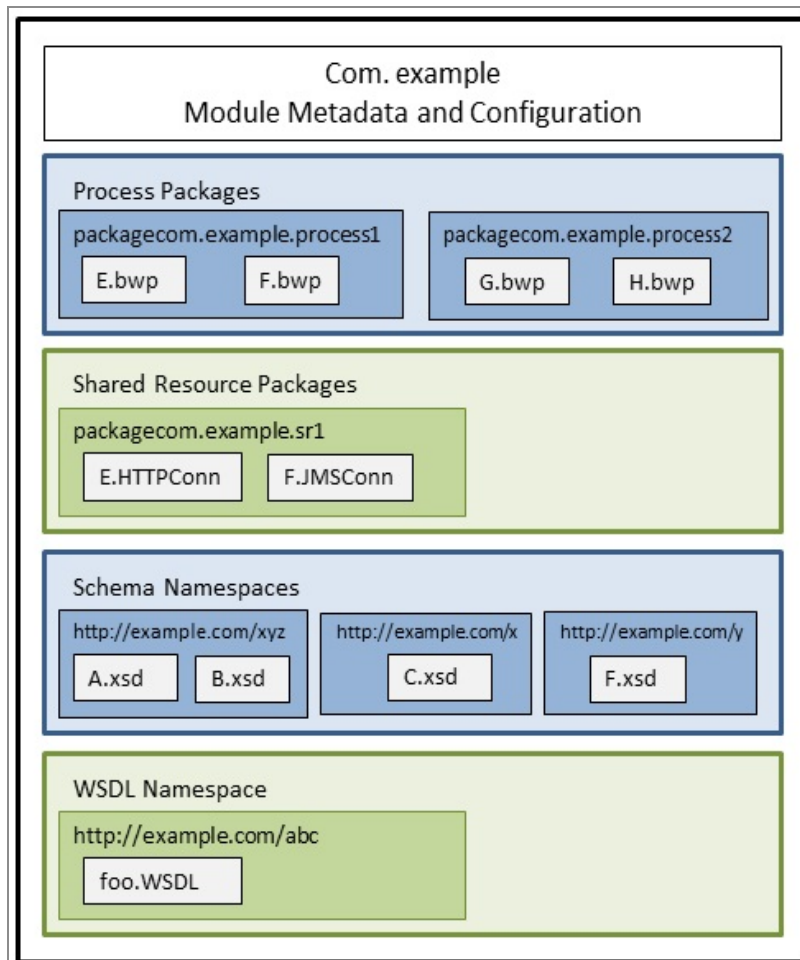
Note: Use the Component section under the **Module Descriptor** node to configure the components for this specific application module.

- **src:** Default source directory created when the project is Java enabled. A project can contain multiple source directories which 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

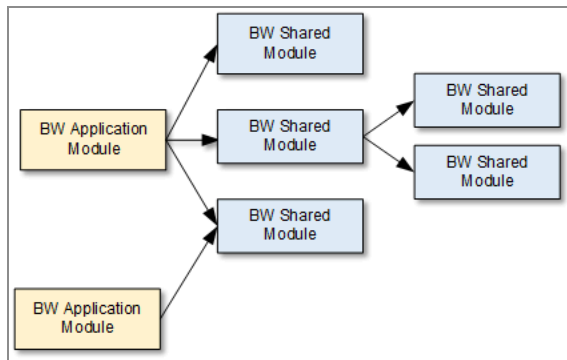


Shared Modules

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. A shared module cannot be deployed by itself; it must be included as part of an application module.

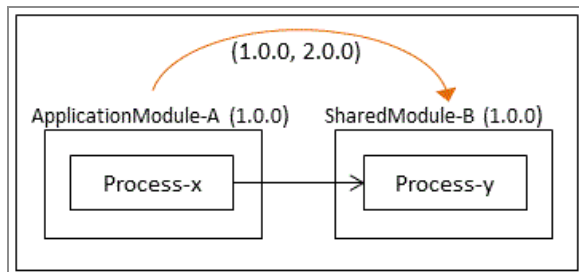
Shared modules export their functionality (processes, shared resources, and schema namespaces) to application modules or to other shared modules. This means there is a possibility that other modules in the system depend on a shared module for this information.

Relationship Between Application Modules and Shared Modules

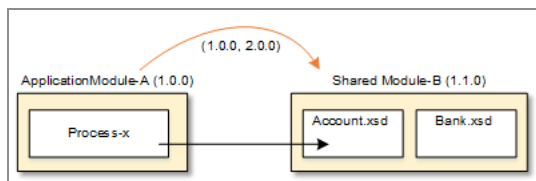


Shared modules can depend only on other shared modules and cannot depend on application modules.

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 that are intended to be exported from a shared module must be contained in the **Schemas** special folder.



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

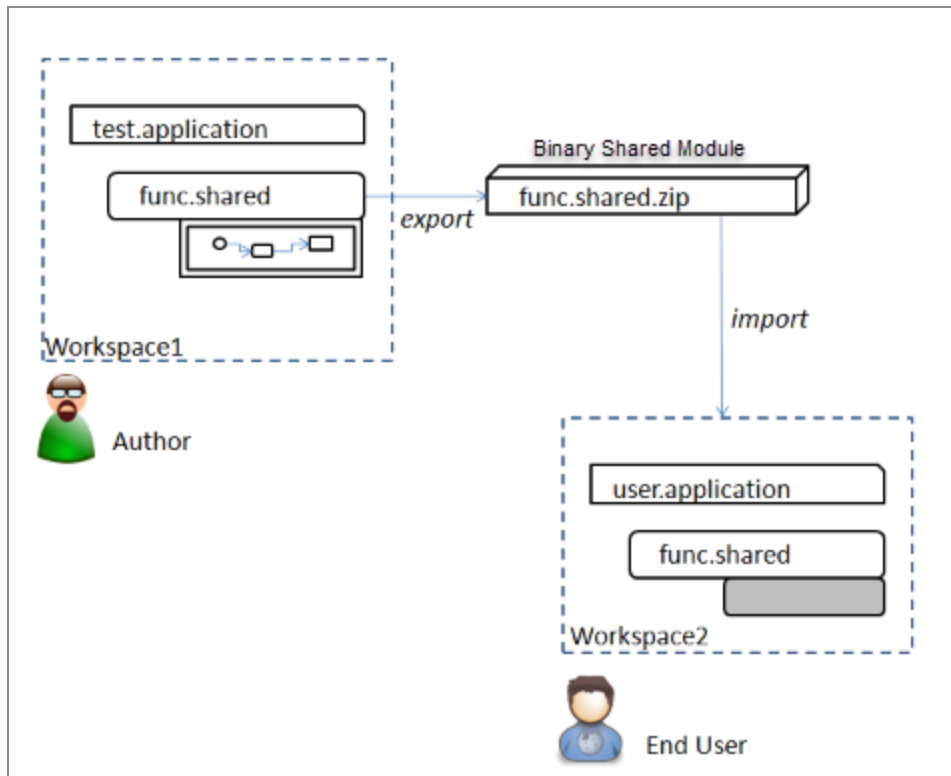
Binary Shared Modules

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 ActiveMatrix BusinessWorks™ Application Development*.

Processes

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

Processes comprise of 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 is making 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 is making the call is the *parent process*. The called process is referred to as a subprocess or a *child process*.

There are two types of subprocesses: direct or service.

- A direct subprocess is non-WSDL-based. This means you do not need to use a WSDL to define subprocess details. Instead, you can set input and output information on the **Start** and **End** activities in the subprocess interface.
- A service subprocess requires a WSDL to define subprocess details, and can be configured to use SOAP or REST binding.

There are two ways a subprocess can be called: inline or non-inline.

- An inline subprocess is part of the same job as the parent process, which means they share the same engine thread. A non-inline, or a spawned, subprocess is a separate job, which means it can use a separate engine thread. For example, when a service subprocess invokes a process using the **Invoke** activity, the process executes on the same thread if a reply is involved.

i Note: If the parent process of a direct inline subprocess is faulted, the subprocess is canceled as well. Inline direct subprocesses are sensitive to all state changes of the parent process, for example, suspend, resume, or cancel.

- Non-inline subprocesses execute on a separate, engine thread.

Note: If the parent process of an inline, or a non-inline, service subprocess is faulted, the subprocess is canceled as well. Service subprocesses, similar to inline direct subprocesses, are sensitive to any state changes the parent process undergoes.

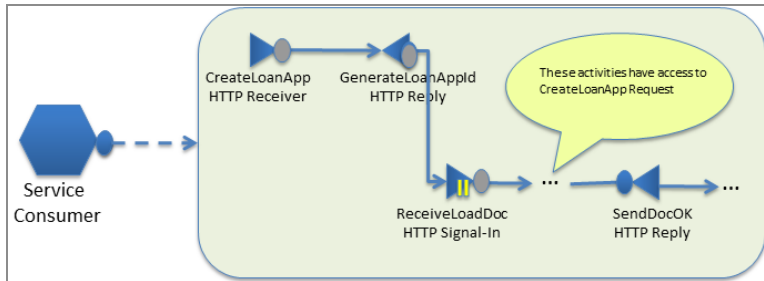
Component Process

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 [Process Starters](#), [Signal-Ins](#), and [Bindings](#).

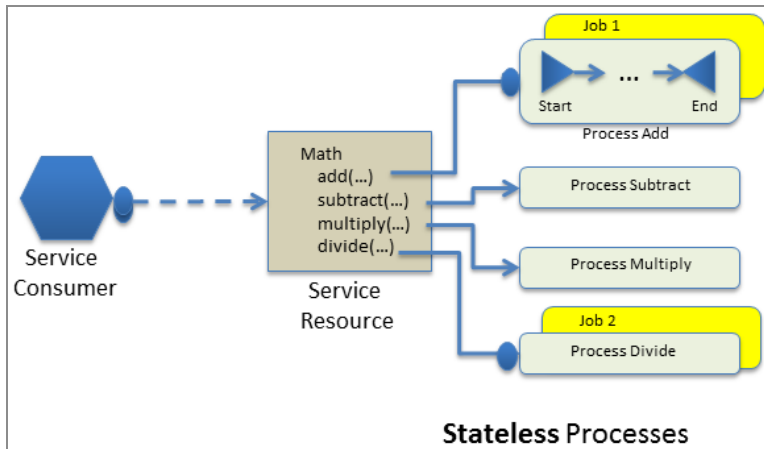
Stateful Process

A process that relies on the ActiveMatrix BusinessWorks engine to maintain its state across invocations is called a *stateful* process. As the engine maintains its state, a stateful process does not require an external persistence store.



Stateless Process

A process that does not require the ActiveMatrix BusinessWorks engine to maintain its state across invocations is called a stateless process. If needed, a developer can design a stateless process to manually maintain the process state by using an external persistence store.



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.

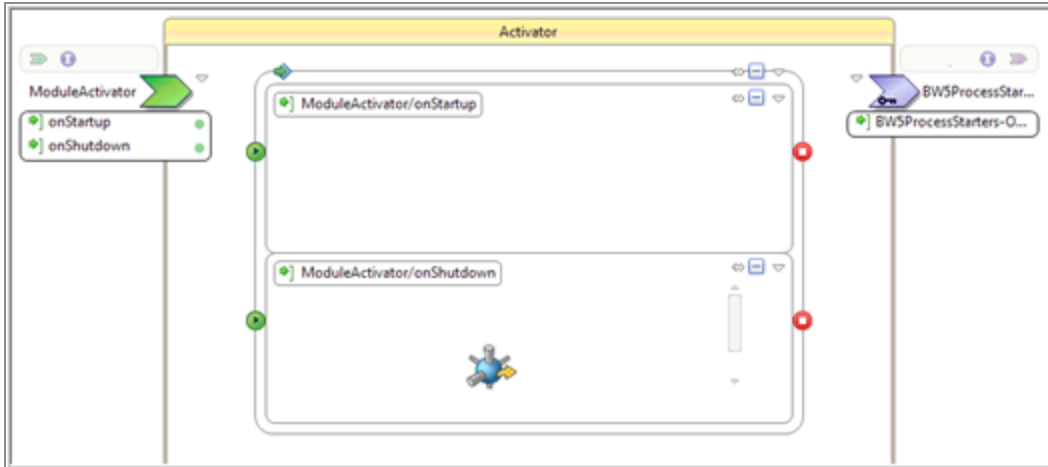
Process References

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 executing 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 sub-processes.

Note: The activator process in ActiveMatrix BusinessWorks 6.x provides the same functionality as that of the OnStart and OnShutDown activities in ActiveMatrix BusinessWorks 5.x.

For information on how to create an activator process, see "Creating an Activator Process" in *TIBCO ActiveMatrix BusinessWorks™ 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 executed on the job created by their calling process.

A process can define only private properties and private references.

i Note: For details about the TIBCO Business Studio for BusinessWorks development environment, see [Design-time Concepts](#).

Process Packages

Process packages are groups of related processes.

Process packages are similar to Java packages in their semantics and in the way they are represented in the file system.

Visibility of the processes outside of the package depends on whether the processes are declared as public or private:

- Private processes can be invoked only by processes that are part of the same package.
- Public processes can be invoked by processes that are defined either inside or outside the package.

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. 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, and so on. For example, the arrow sign in the indicates data is being sent by this activity.

i Note: Detailed descriptions of palettes are available in the TIBCO ActiveMatrix BusinessWorks™ 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 throw 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. Signal-in activities are always blocking.

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.
- **Signal-in Activities** wait for an asynchronous event in a process. They proceed with executing the process instance when an appropriate event is received. Signal-in activities require conversations to be configured. For more information, see [Conversations](#).

i Note: For details about the TIBCO Business Studio for BusinessWorks development environment, see the Concepts Guide. [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 displays the list of activities contained in a palette and allows 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.

i 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 executed 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, 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 execute 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 executed 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 executed.

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.

Mapping Concepts to a Sample: File Poller

The concepts introduced in previous sections enable you to understand and design a simple business process. The File Poller example demonstrates such a process.

Pre-requisites

The File Poller sample demonstrates the concepts introduced in the section [General Concepts](#):

- [Applications](#)
- [Application Modules](#)
- [Shared Modules](#)
- [Processes](#)
- [Activities](#)
- [Transitions](#)
- [Shared Resources](#)

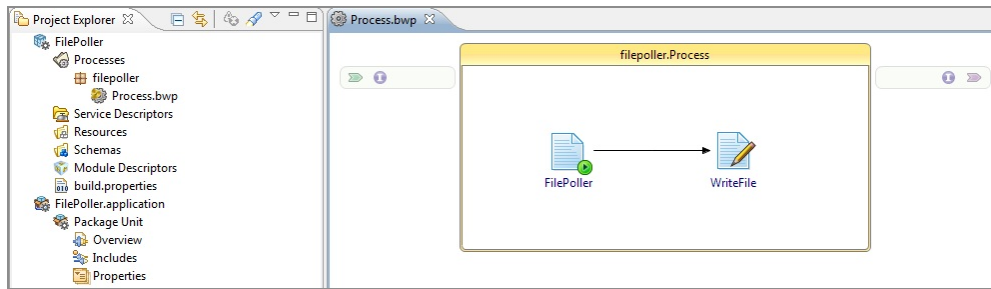
After going through these sections, you should be able to understand and execute a simple process such as the File Poller.

File Poller Sample

The File Poller sample project creates a simple process that polls a file at a given location, for example c:\tmp\fileread.txt, to periodically check if the file was changed and writes the content of the polled file to a specified output file. By default, the file is created if it does not exist in the specified location.

The activities, **File Poller** and **Write File**, from the File palette are used in this process. The data flows from **File Poller** activity to the **Write File** activity and is illustrated by the transition arrow in [File Poller Process Diagram](#).

File Poller Process Diagram



For step-by-step instructions to create and test the File Poller process, see *TIBCO ActiveMatrix BusinessWorks™ Getting Started*.

The Project Explorer in the [File Poller Process Diagram](#) also shows the application - FilePoller.application, application module - FilePoller, and the process - Process.bwp created when developing the File Poller sample.

Next Steps

After completing this section, you should be able to design a simple process with minimal assistance. You can further build on this sample to solve problems using batch-oriented and process-oriented styles by making use of TIBCO® Adapters and activities from other palettes such as JMS, JDBC, and FTP.



Additional General Concepts

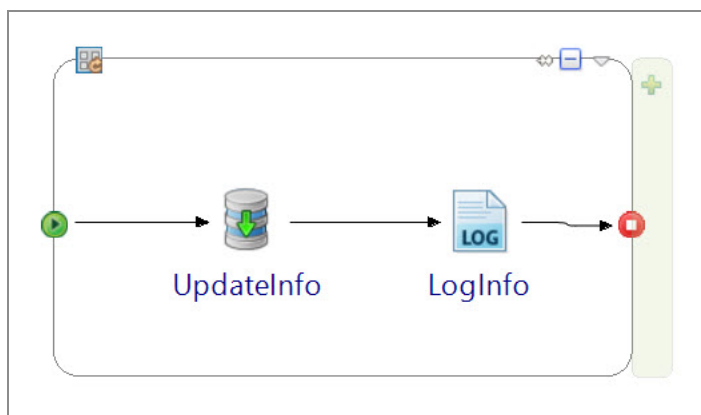
This section introduces the following additional concepts that help build complex business processes.

Groups

Groups consist of one or more activities that are assembled together and executed 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** : **Critical Section** groups are 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 executing the critical section completes. **Critical section** groups are 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.
- **Local Transaction** : Local Transaction group provides for activities such as database updates. It performs auto commit on a local transaction at the end of the group, making the use of transactional activities easier for users.

Groups With Conditions (Repetitive)

Loops are groups with conditions which 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 executes 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 executes 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 executed if the condition is initially false. The While loop and continues to execute as long as the condition holds true and stops when the condition evaluates to false.

i 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.

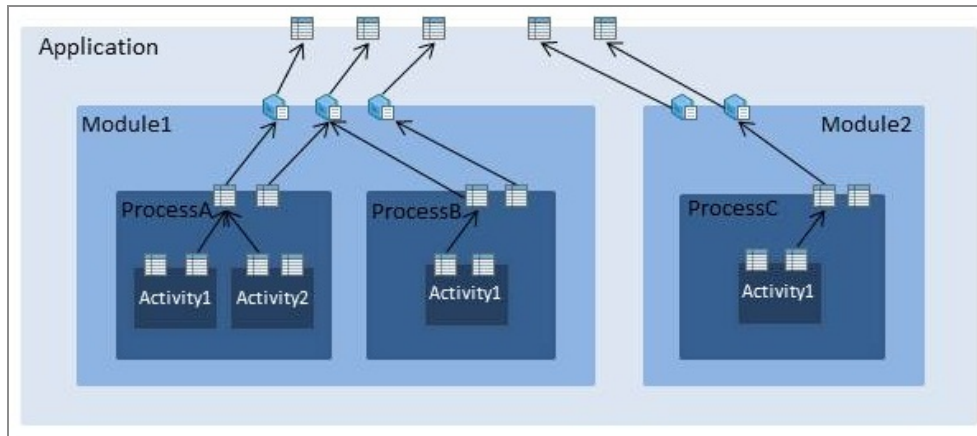
The three levels of properties are hierarchical: application properties are in the outer most scope, followed by module properties, followed by process properties.

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. Private properties are not visible to the encapsulating layers.

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	<ul style="list-style-type: none"> Visible within the module. Private module properties cannot be viewed from the Admin UI. Not visible or changeable from the Admin UI. 	<ul style="list-style-type: none"> Literal or a shared resource reference. Private module property values cannot be edited from the Admin UI. 	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	<ul style="list-style-type: none"> Visible within the module. Visible within projects that contain dependencies to the Shared 	<ul style="list-style-type: none"> Literal or a shared resource reference. Private module property 	<ul style="list-style-type: none"> Shared Module Properties are module properties that come from a Shared Module. Cannot be assigned to an activity directly.

Property	Scope or Visibility	Values	Additional Information
	<p>Module that the Shared Module Property came from.</p> <ul style="list-style-type: none"> Private module properties cannot be viewed from the Admin UI. Not visible or changeable from the Admin UI. 	<p>values cannot be edited from the Admin UI.</p>	<p>You need to reference a module property from a process property, and then reference the process property from the activity.</p> <ul style="list-style-type: none"> Can be used for activities, process properties, shared resources, and SOAP Bindings.
Application Properties	<ul style="list-style-type: none"> Only available for an application and visible within the application. These properties are visible from the Admin UI. 	<ul style="list-style-type: none"> Literal. Profiles can be used to specify a new set of values for the same application. 	<ul style="list-style-type: none"> Overrides module properties, thus enabling you to use different values for the same module. Cannot add new properties at application level.

i 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 ActiveMatrix BusinessWorks™ 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.

Persistent Module Shared Variable

The current state of a module shared variable is stored in memory by default and the state of the module shared variable is lost in case the ActiveMatrix BusinessWorks engine (or the AppNode) crashes. However, to preserve the current state in case of an engine failure, the module shared variable can be configured to be persistent.

For more information on configuring module shared variables with persistent option, see "Using Shared Variables" in *TIBCO ActiveMatrix BusinessWorks™ Application Development*.

When a module shared variable is configured as persistent, its current state is written to the engine database and on engine restart the module shared variable state is restored.

Sharing Module Shared Variable Across Multiple Engines (AppNodes)

A module shared variable state can be accessible across multiple engines if the engine persistent mode (`bw.engine.persistentMode`) is set to "group" and the module shared variable is configured to be persistent.

For more information on configuring engine persistence, see "Engine Persistence Modes" in *TIBCO ActiveMatrix BusinessWorks™ Administration* guide.

Using the persistence option, the same ActiveMatrix BusinessWorks applications running in multiple engines (AppNodes) that are part of the same engine group can read or update

the module shared variable state. Once the value is updated, the new value is available to all engines (AppNodes) which are running the same application.

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. They can also be set in shared modules, and accessed by application modules after setting up dependencies to the shared module.

Sharing Job Shared Variables Between Inline and Non-Inline Processes

Inline subprocesses are executed 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 executed on the same job as the caller process. Hence 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. Persistence option is not available for the Job Shared Variables.

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 execute 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 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. To synchronize multiple critical section groups, use a shared lock. The shared lock can be defined using a module or a job shared variable.

Conversations

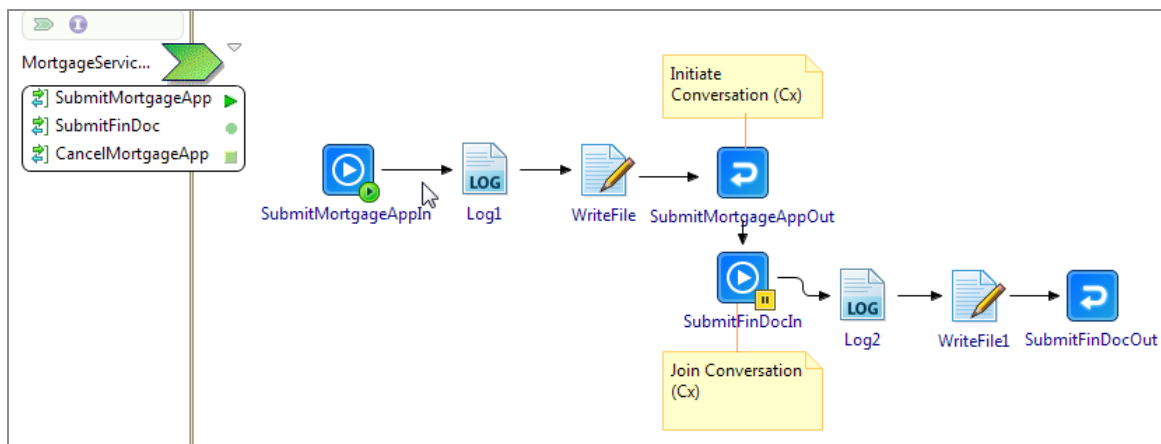
Conversations represent two or more related message exchanges in the same process which are correlated by the engine, for example, a conversation between a process and its clients, or between a process and its back-end service.

Conversations are used for stateful processes, which can consist of one or more operations. In a stateful process, the engine manages the state and helps correlate messages with the process; you need not correlate the state of the operations. In a stateless process, the state is managed by the job itself.

A conversation is defined within the scope of a process. Conversations are always initiated by one activity and joined by other activities. The activity that initiates a conversation generates a conversation ID. The activities that join this conversation use the generated conversation ID when exchanging messages.

Consider the Mortgage Service Provider process shown in the following figure. The Reply activity, `SubmitMortgageAppOut`, initiates a conversation, generates a conversation ID, then returns this ID in its reply message. The Receive activity, `SubmitFinDocIn`, joins the conversation initiated by the Reply activity, `SubmitMortgageAppOut`. When submitting the final documents, a client must use the conversation ID returned by the Reply activity. The engine uses the conversation ID to correlate messages with the process and ensures that the documents are associated with the right mortgage application.

Mortgage Service Provider Sample Using Conversations



When designing an application, better support for conversations can be achieved by defining the following:

- Number of conversations that a process participates in.
- Definition of the activities that are part of the same conversation.
- Sequence of messages for each conversation.

Event Handlers

Event handlers are used to overcome issues with interruptive and blocking activities.

Blocking activities contain a job that has to wait until a certain activity is executed. When using blocking activities, all events have to be handled in the order the process was designed. However, it is not possible to design a process without knowing when a message is received. A shopping cart is a good example of a process where adding and removing items is done in a random order by the shopper.

Event handlers allow asynchronous event processing. They are always attached to a scope and execute parallel to the main business logic of the process, so they are associated with an operation that is a part of the process.

Event handlers can be defined at two different levels:

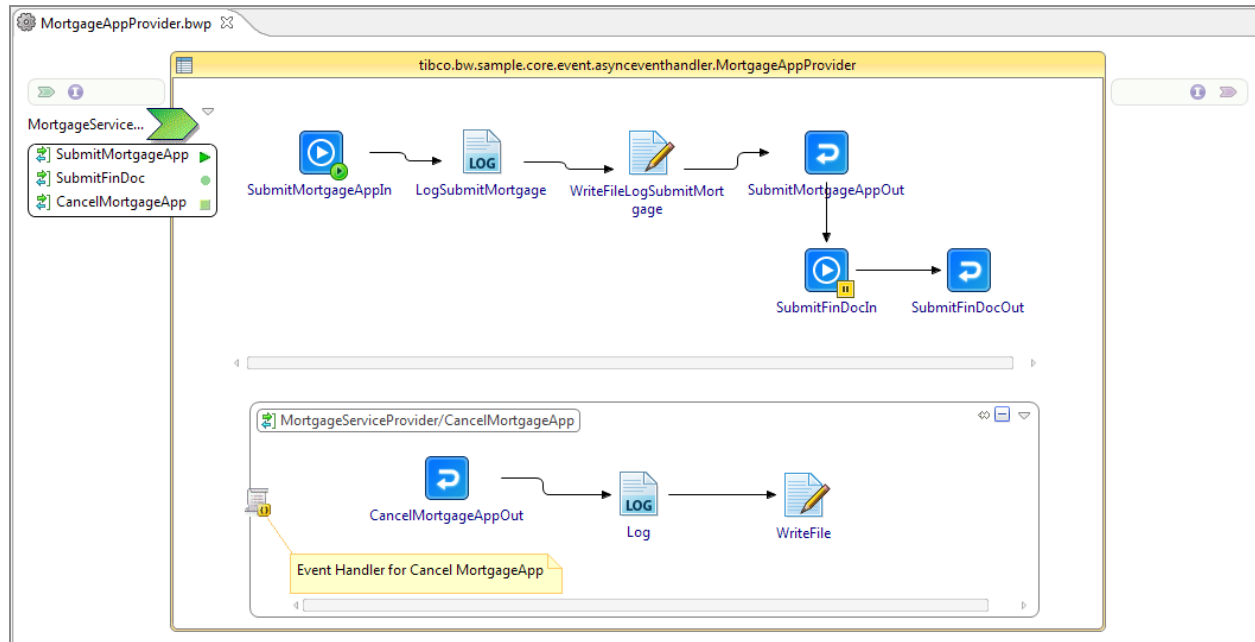
- Process level - When defined at the process level, allows you to cancel the job.
- Scope level - When defined at the scope level, allows you to cancel the scope.

Each event handler is associated with one process and has access to that process. An event handler can be executed multiple times during the process execution and it is executed in parallel with the other tasks within the scope or process. The tasks within a scope or process can be terminated, prior to executing the tasks within the event handler, using the Cancel Scope Execution option.

Consider the Mortgage Service Provider process shown in the following figure. The Mortgage service contains three operations - SubmitMortgageApp, SubmitFinDoc, and CancelMortgageApp. The CancelMortgageApp operation is defined in the event handler container. Both the CancelMortgageApp and SubmitFinDoc operations have a correlation key obtained from the response to the SubmitMortgageApp operation. After submitting an application, if the consumer invokes the CancelMortgageApp operation with the correlation key before submitting the documents, the event handler containing the implementation for

the CancelMortgageApp operation is executed in parallel to the main process. The engine does not wait for the SubmitFinDoc activity to be executed.

Mortgage Service Provider Process With Event Handler



Fault Handlers

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

Fault handlers are the recommended way 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.

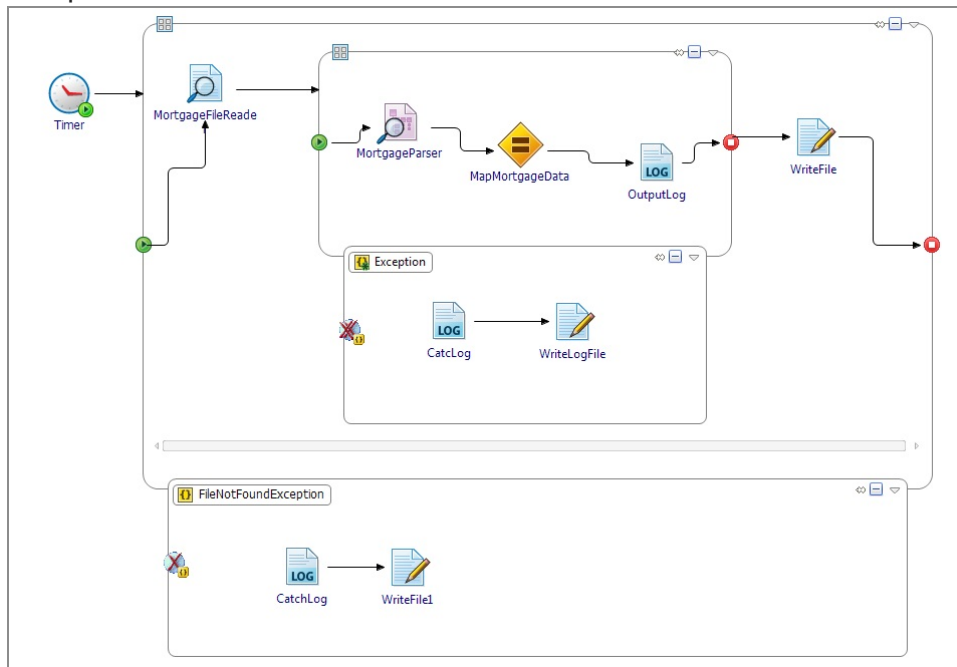
Fault handlers when defined at the scope level, allows you to catch faults or exceptions thrown by activities within a scope. To catch faults or exceptions specific to an individual

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 executed, the associated scope is not completed due to the error thrown. If a fault is not thrown in the fault handler, the process execution continues with the first activity that follows the scope. If a fault is thrown 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 executes it. Once the enclosing fault handler finishes its execution, the engine executes the next activity following the scope. If no fault handlers are found in the enclosing scopes, then the job terminates 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 transition to take in case of an error. For more information, see [Error Transitions](#).

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.

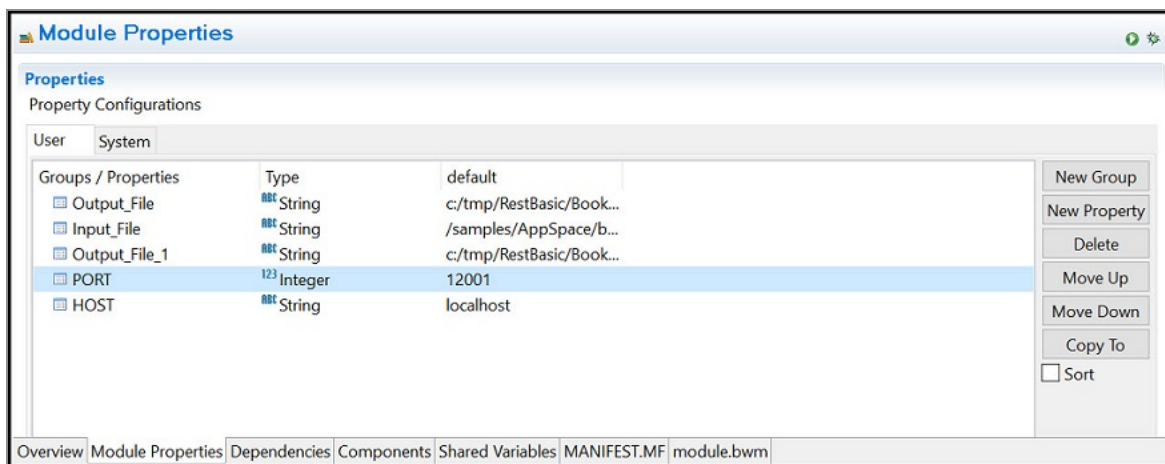
Module Property

Module property provides the default value for a module. Multiple process properties can source their value from a single module property.

Module properties are defined at the module level and can be referenced by various resources that are defined as a part of the module. Their values can also be sourced from application properties at deployment time.

Note: A module property cannot be assigned directly to an activity. It must be assigned to a process property in order to be used in an activity. The process property inherits the value of the module property that is assigned to it. The process property can then be used in the activity. Module properties can be used directly only when configuring shared resources and policy resources.

The Module Properties editor can be used to create and manage module properties. You can add a new property or logical groups which you can use to organize module properties.



Components

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

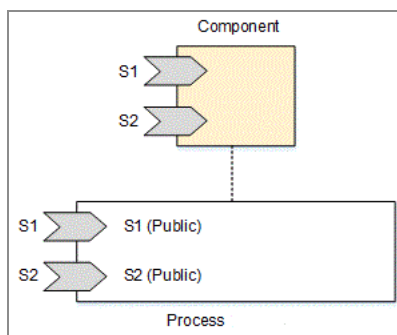
Components are generated only for main processes and each main process initialized by the engine must have a component associated with it. Components are required only by main processes that are responsible to initiate the business logic. Subprocesses do not require components as they are called by another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

Services

ActiveMatrix BusinessWorks 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 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 upon 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 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.

ActiveMatrix BusinessWorks 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 ActiveMatrix BusinessWorks 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 ActiveMatrix BusinessWorks:

Basic Authentication

Validates the user name 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 ActiveMatrix BusinessWorks™ 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 ActiveMatrix BusinessWorks™ 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	<ul style="list-style-type: none"> • LDAP Authentication • XML Authentication
Basic Credential Mapping	<ul style="list-style-type: none"> • Identity Provider
WSS Provider	<ul style="list-style-type: none"> • Subject Provider • Keystore Provider • Trust Provider • WSS Authentication
WSS Consumer	<ul style="list-style-type: none"> • 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 ActiveMatrix BusinessWorks run time component that dynamically enforces policies during runtime. A governance agent must be enabled on an appnode to enforce policies applied to ActiveMatrix BusinessWorks applications.

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

Mapping Concepts to a Sample: Mortgage Broker Service and Client

The concepts introduced in sections General Concepts, Groups, Conversations, and Services together enable you to understand and design a service-oriented solution such as the Mortgage Broker Service Client example.

Pre-requisites

The Mortgage Broker Service Client sample demonstrates the concepts introduced in the following sections:

- [General Concepts](#)
- [Groups](#)
- [Conversations](#)
- [Services](#)
- [SOAP Services](#)

After reading these sections, you should be able to understand and execute a service-oriented sample such as the Mortgage Broker Service Client.

Mortgage Broker Service Client Sample

In this sample, a service implements a simplified online mortgage broker application. The borrower requests a loan through a broker. The broker processes the loan request using one of the third-party partner services. The borrower can either specify the preferred third-party provider or allow the broker to default to one. The third-party partner services request credit rating of the borrower from a credit check service and in turn approves or rejects the loan application based on the credit rating.

The Mortgage Broker Service Client 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 service-oriented processes with minimal assistance.

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.

Pre-requisites

The Bookstore sample requires the concepts introduced in the following sections:

- [General Concepts](#)
- [Additional General Concepts](#)

After going through these sections, you should be able to understand and execute 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 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

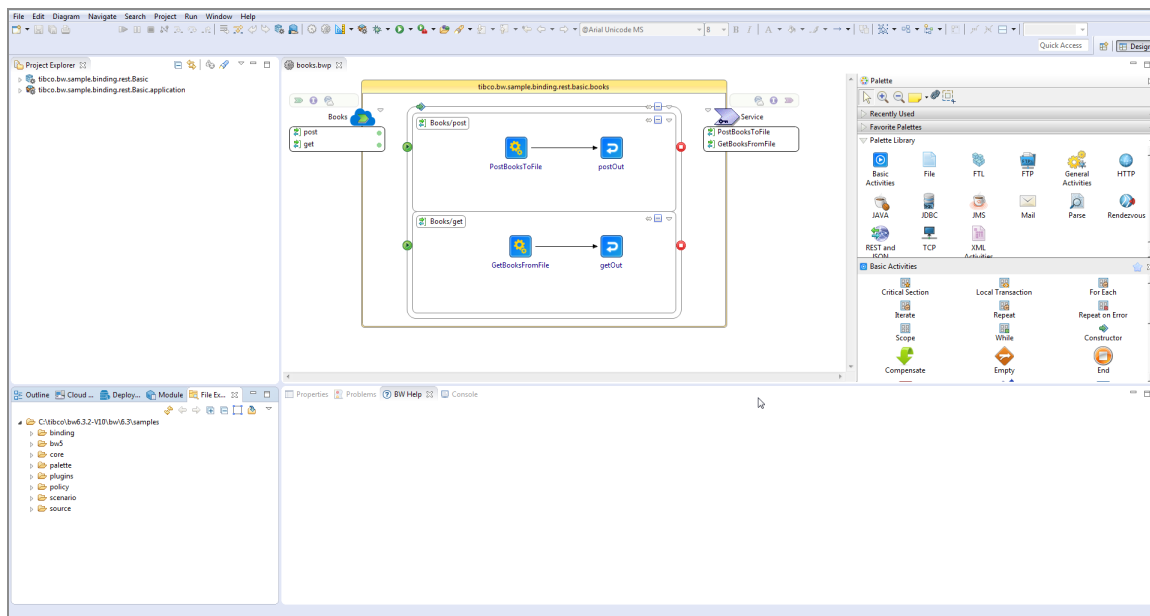
Design-time concepts introduces TIBCO Business Studio for BusinessWorks, an Eclipse-based integration development environment that is used to design, test, and deploy applications.

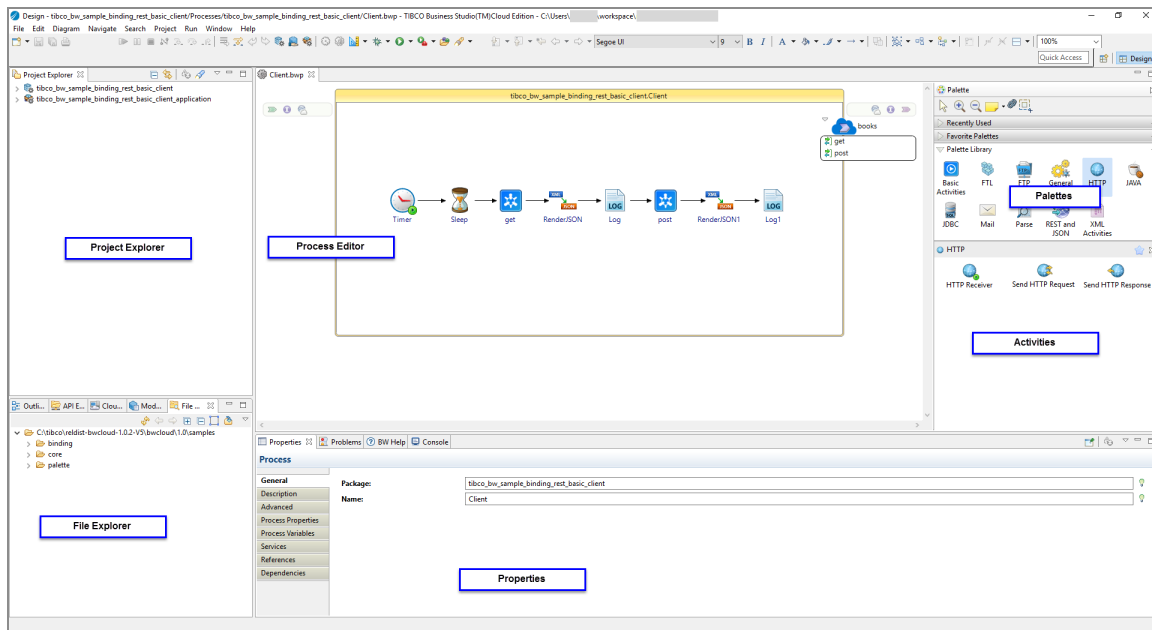
TIBCO Business Studio for BusinessWorks provides Eclipse extensions such as editors, palettes, and so on.

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





The workbench consists of:

- **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 launches 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 **Window > Show View > <view_name>**.
- **Editors:** Provide a canvas to configure, edit, or browse a resource. Double-click on 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 a selected artifacts in an editor
- **Module** tab: Displays the module properties and shared variables used in the module

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

Run configurations specify information such as:

- Bundles to be executed.
- Arguments such as the target operating system, target architecture, target web services, 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 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 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.

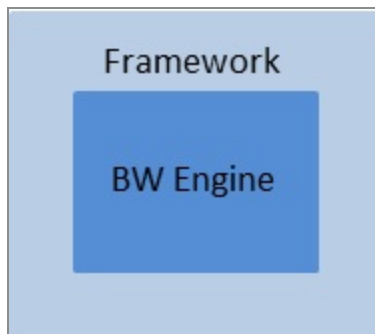
Runtime Concepts

Runtime refers to the AppNode and the ActiveMatrix BusinessWorks engine that host and execute .

AppNode

An AppNode (also called *bwappnode*) is an operating system process (JVM) that hosts and executes ActiveMatrix BusinessWorks applications. An AppNode consists of two key layers: the OSGI Framework and ActiveMatrix BusinessWorks 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, and interacts with the Administrator (TIBCO® Enterprise Administrator or bwadmin utility). The engine layer is responsible for executing the application. The engine is multi-threaded and can execute multiple jobs for the same or different applications concurrently.

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

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

Each AppNode is associated with an AppSpace. For more information about AppSpaces, see [Administration Concepts](#).

Process Instance

Execution of any process 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, and so on), or simply by messages sent by other processes.

The ActiveMatrix BusinessWorks 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 executes 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 executed in conjunction with a particular event. Even though these are separate process instances they work together and can be executed 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 executes 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 to be a part of the same job. Non in-line subprocesses spawn a new engine thread and are executed on a different job.

Administration Concepts

Applications are deployed into runtime environments and managed using the bwadmin utility. TIBCO® Enterprise Administrator can also be used to manage and monitor applications.

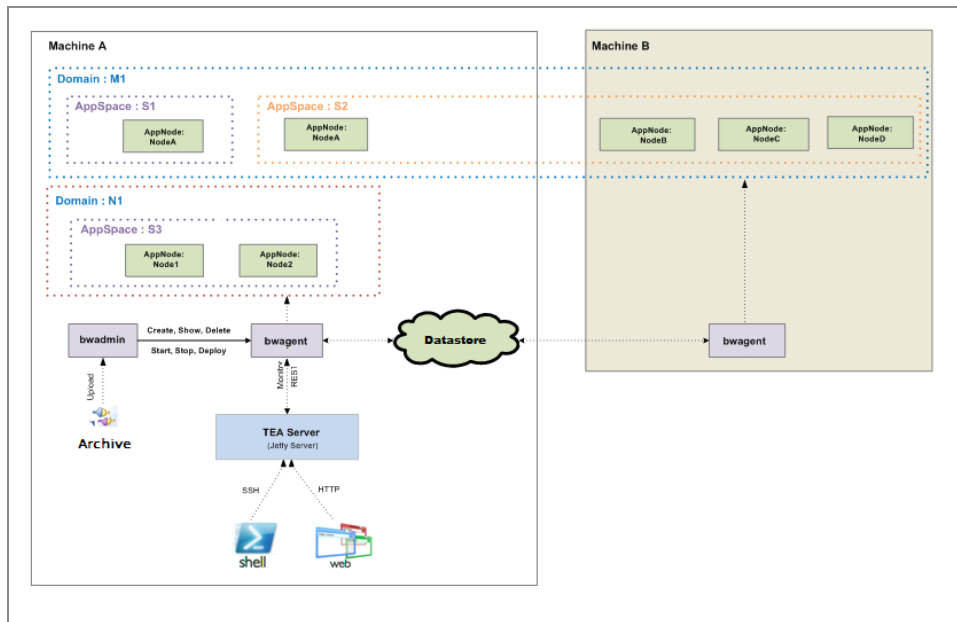
ActiveMatrix BusinessWorks provides a flexible framework that allows you to scale your runtime environment as needed. The runtime provides an option to execute the ActiveMatrix BusinessWorks engine so that the risk of a single point of failure when running an application is reduced. The engine is responsible for executing the applications.

Following are the key administrative components:

- An [Application Archive](#) is the deployment unit for an application that is generated in TIBCO Business Studio for BusinessWorks.
- A [Domain](#) is a logical group that provides an isolated environment for applications and their resources to reside.
- An AppSpace is a group of one or more AppNodes, which are runtime entities that host ActiveMatrix BusinessWorks applications. AppSpaces are contained within a domain. One or more application can be deployed to an AppSpace.
- An AppNode is a runtime entity that hosts applications. AppNodes are contained in an AppSpace.
- The [bwagent](#) is a daemon process that runs on every ActiveMatrix BusinessWorks installation. When multiple installations across machines are configured as a network, the bwagents interact with each other using a datastore. They also synchronize the data from the datastore with the local file system.

In the following Administration Architecture illustration, domain M1 spans two machines, Machine A and Machine B. Domain N1 is on Machine A. Domain M1 contains two AppSpaces, AppSpace S2 spans both the machines. The bwagent on Machine A is configured to interact with the bwagent on Machine B through the datastore.

Administration Architecture



The bwagent on Machine A is registered with the TIBCO® Enterprise Administrator server. If the registered bwagent becomes unavailable, the connection between the TIBCO Enterprise Administrator server and the agent network is automatically recovered. The bwagent on Machine B is automatically registered with the server.

If the TIBCO® Enterprise Administrator server becomes unavailable, running applications and AppSpaces are not impacted.

The runtime entities manifest as a hierarchical folder structure on the local file system. Every action performed on the runtime entities results in an update to the file system. The location of the default domains folder in the local file system can be changed by editing the *BW_HOME/domains/DomainHomes.properties* file.

When the runtime entities span across multiple machines, the bwagent synchronizes the data from the datastore with the local file system. The AppNodes that host and execute the applications read their configuration and data only from the local file system, making the file system the source of truth. The bwagents ensure that all AppNodes of an AppSpace access the exact same applications. Within an AppSpace all applications executed by all AppNodes are identical. This ensures that in case of a failure in the communication channel, the runtime is not affected as it refers to the data on the local file system.

i Note: ActiveMatrix BusinessWorks supports executing different versions of applications within the same AppSpace.

When the runtime entities span machines, the bwagent synchronizes the data from the datastore with the local file system. The AppNodes that host and execute the application reads their configuration and data only from the local file system, making the file system the source of truth. The BWAgents ensure that all AppNodes of an AppSpace access the exact same application. Within an AppSpace, the application executed by all AppNodes is identical. This ensures that in case of a failure in the communication channel, the runtime is not affected as it refers to the data on the local file system.

For more information, see *TIBCO ActiveMatrix BusinessWorks™ Administration*.

Application Archives

An application archive is a deployment unit for an application that is generated in TIBCO Business Studio for BusinessWorks. 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 for BusinessWorks and can range from simple to very complex. An ActiveMatrix BusinessWorks application consists of an application module ([Application Modules](#)), which in turn consists of one or more processes that define the business logic, and zero or more shared modules ([Shared Modules](#)). ActiveMatrix BusinessWorks 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.

At runtime, application names must be unique within an AppSpace. 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 runtime. If any further modifications are made to the application, the archive file must be regenerated with an updated version number and redeployed to the AppSpace. Any modifications to the application are then installed as hot fixes or service packs. Only a specific version of an application can be modified by a hotfix or service pack.

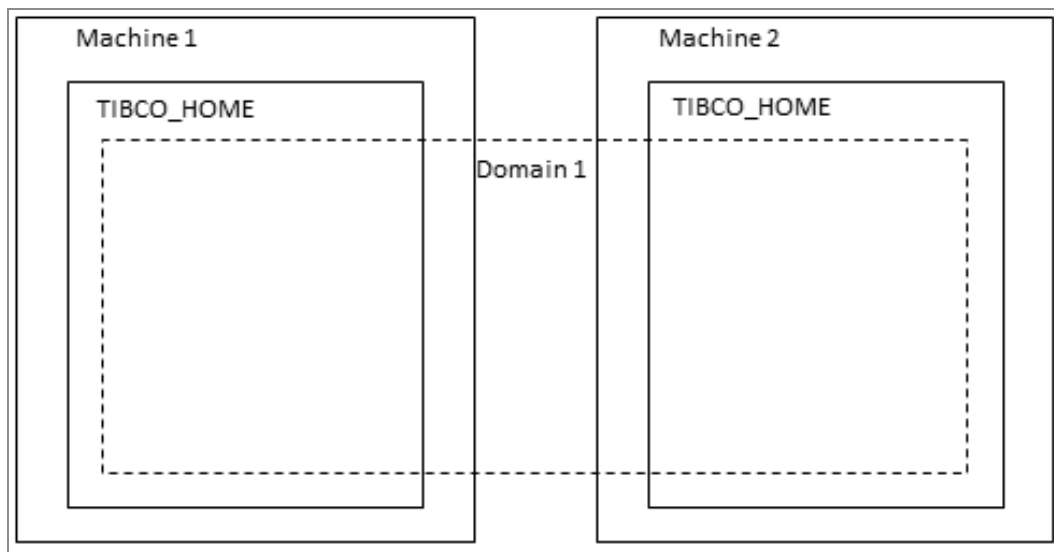
Domains

A domain is a logical group that provides an isolated environment for applications and their resources. Runtime entities such as AppSpaces and AppNodes are contained within a domain.

A domain can span more than one machine and can share a machine with other domains such that one machine can contain more than one domain. Applications in one domain are separated from applications in the other domains.

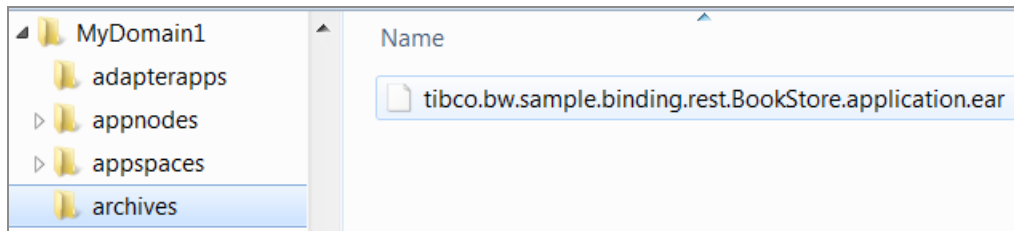
A domain is the first runtime entity you must create. Other runtime entities such as AppSpaces and AppNodes can only exist within a domain. An application archive is first uploaded to a domain. The application contained in the application archive can then be deployed into one or more AppSpaces for execution.

The following diagram shows a single domain that spans two machines. The artifacts installed, configured, or deployed into Domain1 are available on both machines.



A domain manifests as a folder, *domain_name*, in the file system and is located in the `<TIBCO_HOME>\bw\domains<TIBCO_HOME>\bwx\domains` directory. This folder contains sub-folders `appspaces` and `appnodes` to store data about the AppSpaces and AppNodes contained in the domain. It also stores the application archive files that are uploaded to the domain under the `archives` sub-folder.

File System Manifestation of a Domain



AppSpaces

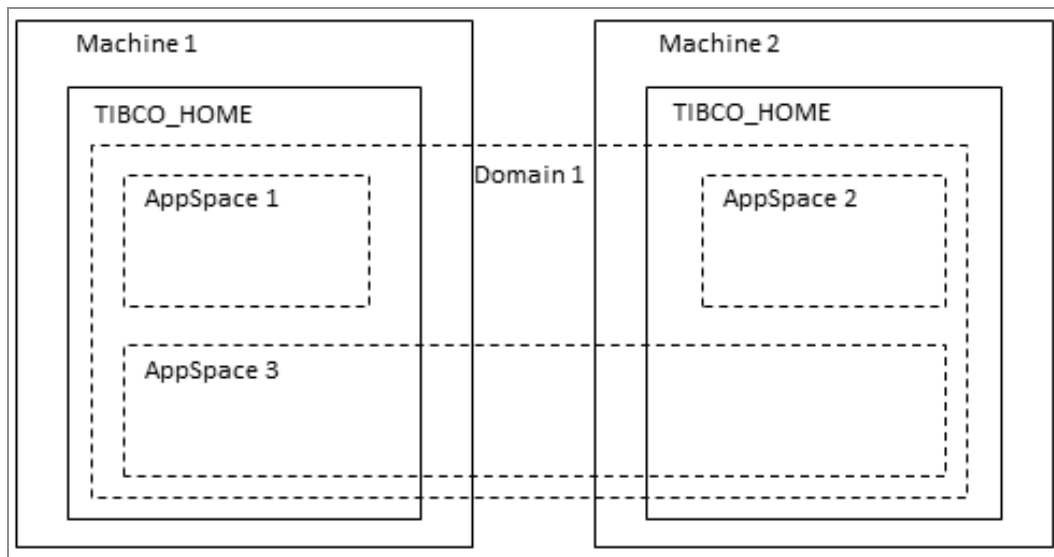
An AppSpace is a collection of one or more AppNodes.

A domain can contain one or more AppSpaces. AppSpaces can span multiple physical machines across networks. An AppSpace manifests on the physical machine as a predefined folder structure that contains information about the applications deployed in that domain. One or more than one application can be deployed to an AppSpace.

A domain can contain one or more AppSpaces. AppSpaces can span multiple physical machines across networks. An AppSpace manifests on the physical machine as a predefined folder structure that contains information about the applications deployed in that domain. One application can be deployed to an AppSpace.

Each AppSpace contains one or more execution runtimes called AppNodes which host the applications. When you deploy an application to an AppSpace, the application is deployed to all AppNodes that are part of the same AppSpace. An AppSpace is extensible, which allows AppNodes to be added dynamically to scale the load on an application, thereby providing load-balancing and fault-tolerance for applications. You can add and remove AppNodes to an AppSpace even after an application has been deployed.

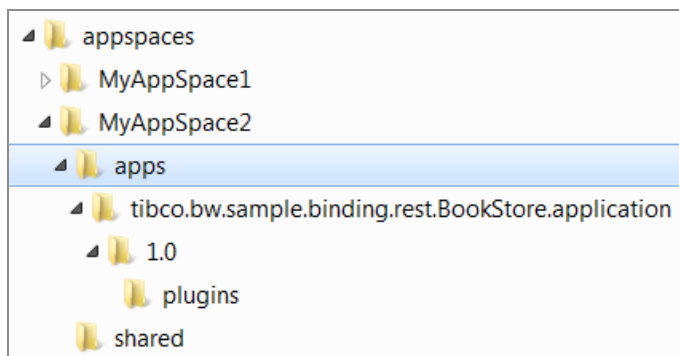
The following diagram shows AppSpace1 on Machine 1, AppSpace2 on Machine 2, and AppSpace3 spanning Machine 1 and Machine 2:



An AppSpace is a folder, *appspace_name*, in the file system and is located in the `<TIBCO_HOME>\bw\domains\domain_name\appspaces<TIBCO_HOME>\bw\domains\domain_name\appspaces` directory, where *domain_name* is the domain it belongs to. It contains a folder for each AppSpace in the domain, identified by the AppSpace name, which is unique for each domain.

The *appspace_name* folder contains two subfolders - *apps* and *shared*. The *apps* folder contains applications that are deployed in the AppSpace.

File System Manifestation of an AppSpace

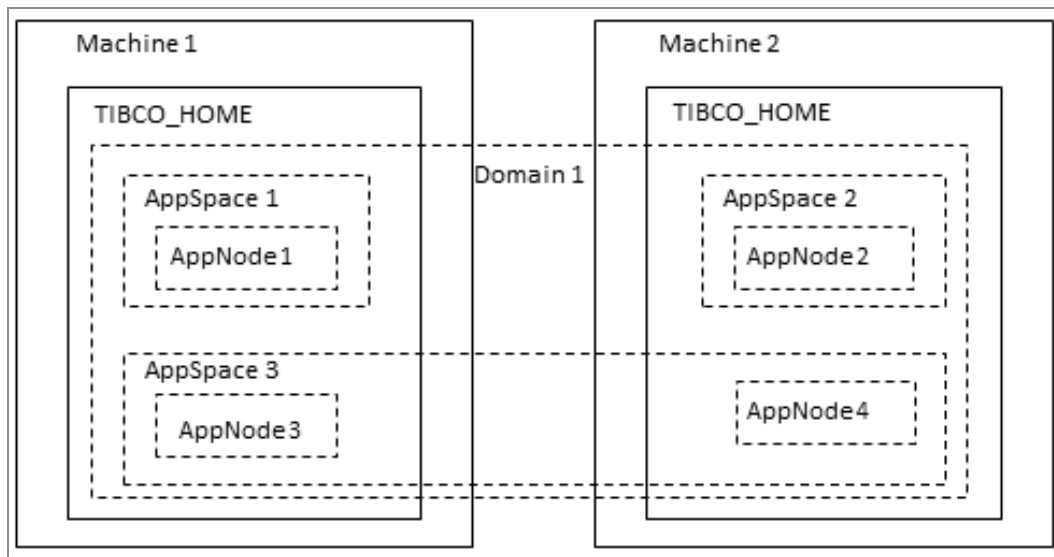


AppNodes

An AppNode is a JVM process that hosts applications created in TIBCO Business Studio for BusinessWorks. An AppNode can belong to only one AppSpace.

Each application that is deployed to an AppSpace runs on all of its AppNodes. AppNodes allow vertical and horizontal scaling. When AppNodes are added to an AppSpace, more processing capacity becomes available for the deployed application to handle a higher load of requests. AppNodes can be added to an AppSpace even after an application has been deployed, allowing the deployed application to scale dynamically across all the AppNodes.

The following diagram shows Domain1, with three AppSpaces and four AppNodes. AppSpace1 and AppSpace2 contain one AppNode each, while AppSpace3 contains two AppNodes. AppSpace3 spans two machines, with AppNodes on each machine.

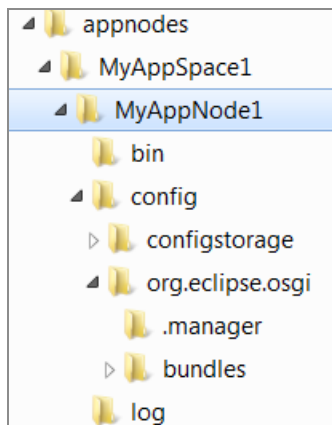


An AppNode manifests as a folder, *appnode_name*, in the file system and is located in the `<TIBCO_HOME>\bw\domains\domain_name\appnodes` directory, where *domain_name* is the domain it belongs to.

The *appnodes* folder contains a subfolder for each AppNode in the AppSpace in the domain, identified by the AppNode name (unique for each AppSpace).

The *appnode_name* folder contains the executable binaries and corresponding tra files, the AppNode's configuration file, and the log file.

File System Manifestation of an AppNode



Understanding the TRA File Structure

The AppNode and AppSpace tra files are read in the order that they are listed below, and can override each other.

- bwappnode-*AppNodeName*.tra
- bwappnode-*AppSpaceName*.tra
- bwappnode.tra

For more information, refer to the following table.

File	Description
bwappnode- <i>AppNodeName</i> .tra	<p>This file can be found in the bin folder at <i><TIBCO_HOME>\bw\domains\<DomainName>\appnodes\<AppSpaceName>\<AppNodeName>\bin\bwappnode-AppNodeName.tra</i>.</p> <p>The configurations in this file apply to an individual AppNode in an AppSpace.</p>
bwappnode- <i>AppSpaceName</i> .tra	<p>This file can be found in the appspace folder at <i><TIBCO_HOME>\bw\domains\<DomainName>\appspaces\<AppSpaceName>\bwappnode-AppSpaceName.tra</i>.</p> <p>The configurations in this file apply to all AppNodes in the given</p>

File	Description
	AppSpace.
bwappnode.tra	This file can be found in the bin folder at <code><TIBCO_HOME>\bw\bin\bwappnode.tra</code> The configurations in this file apply to all AppSpaces on a given machine.

bwagent

A bwagent is a daemon process that is responsible for provisioning AppNodes and applications, performing administration commands, and synchronizing data from the datastore with the local file system.

There is one bwagent for each installation. The bwagent enables communication between agents located on different machines. When multiple bwagents are configured to communicate with each other using a common datastore, they form a bwagent network. bwagents can communicate using TIBCO FTL® for communication transport and TIBCO Enterprise Message Service for communication transport, and by using an external database for data persistence.

For information about configuring the bwagent, see "Configuring bwagent" in *TIBCO ActiveMatrix BusinessWorks™ Administration*.

When multiple bwagents belong to a network and one of the system fails, the failed system can be restored after a restart by using the bwadmin restore command to force the file system to be synchronized with the datastore.

There are multiple ways to access the bwagent: bwadmin, the Admin UI, or the REST API.

- **bwadmin:** In enterprise mode, bwadmin sends commands to the bwagent. The bwagent dispatches the command to the targeted agent. For more information, see the "bwadmin Command Line" tasks under Administration Tasks and Reference section in *TIBCO ActiveMatrix BusinessWorks™ Administration*.
- **Admin UI:** When the bwagent is registered with the TEA server, the Admin UI can be used to create and manage runtime entities. For more information, see the "Admin UI" tasks under "Administration Tasks and Reference" section in *TIBCO ActiveMatrix BusinessWorks™ Administration*.

- REST API: View the bwagent REST API in the Swagger UI. For more information, see the section "Accessing the bwagent REST API with the Swagger UI" in *TIBCO ActiveMatrix BusinessWorks™ Administration*. .

bwagent supports its own set of commands. Commands are issued from the command line in the format: `bwagent [options] command <arguments>`

bwagent commands are listed below.

bwagent Commands

Command	Description
apiserver	Starts the apiserver that hosts the REST API in the Swagger UI. Open a browser and go to the following URL: <code>http://localhost:5555</code>
startagent	Starts the bwagent. This is the same as the default command when no command is given.
stop	Stops the bwagent gracefully.

The following options can be specified for bwagent:

bwagent Command Options

Option	Description	Example
-config	Applies the configuration in the specified file to the server instance.	<code>bwagent -config bwagent.ini</code>
-logconfig <file>	Uses the specified file for logback configuration.	<code>bwagent -logconfig mylogback.xml</code>
-x,--xtrace	Echoes the command to the terminal.	Given <code>bwagent -x</code> , the text <code>+startagent</code> is echoed to the console when the agent starts.

TIBCO Documentation and Support Services

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

How to Access TIBCO Documentation

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

The TIBCO Product Documentation website is updated frequently and is more current than any other documentation included with the product. To access the latest documentation, visit <https://docs.tibco.com>.

Product-Specific Documentation

Documentation for TIBCO ActiveMatrix BusinessWorks™ is available on the [TIBCO ActiveMatrix BusinessWorks™ Product Documentation](#) page.

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

`TIBCO_HOME/release_notes/TIB_BW_6.8.1_docinfo.html` where `TIBCO_HOME` is the top-level directory in which TIBCO products are installed. On Windows, the default `TIBCO_HOME` is `C:\tibco`. On UNIX systems, the default `TIBCO_HOME` is `/opt/tibco`.

The following documentation for this product is available on the [TIBCO ActiveMatrix BusinessWorks™](#) page:

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

- *TIBCO ActiveMatrix BusinessWorks™ Release Notes*
- *TIBCO ActiveMatrix BusinessWorks™ Installation*
- *TIBCO ActiveMatrix BusinessWorks™ Application Development*
- *TIBCO ActiveMatrix BusinessWorks™ Bindings and Palettes Reference*
- *TIBCO ActiveMatrix BusinessWorks™ Concepts*
- *TIBCO ActiveMatrix BusinessWorks™ Error Codes*

- *TIBCO ActiveMatrix BusinessWorks™ Getting Started*
- *TIBCO ActiveMatrix BusinessWorks™ Migration*
- *TIBCO ActiveMatrix BusinessWorks™ Performance Benchmarking and Tuning*
- *TIBCO ActiveMatrix BusinessWorks™ REST Implementation*
- *TIBCO ActiveMatrix BusinessWorks™ Refactoring Best Practices*
- *TIBCO ActiveMatrix BusinessWorks™ Samples*

How to Contact TIBCO Support

You can contact TIBCO Support in the following ways:

- For an overview of TIBCO Support, visit <http://www.tibco.com/services/support>.
- For accessing the Support Knowledge Base and getting personalized content about products you are interested in, visit the TIBCO Support portal at <https://support.tibco.com>.
- For creating a Support case, you must have a valid maintenance or support contract with TIBCO. You also need a user name and password to log in to <https://support.tibco.com>. If you do not have a user name, you can request one by clicking Register on the website.

How to Join TIBCO Community

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

Legal and Third-Party Notices

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

Java and all Java based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

This document includes fonts that are licensed under the SIL Open Font License, Version 1.1, which is available at: <https://scripts.sil.org/OFL>

Copyright (c) Paul D. Hunt, with Reserved Font Name Source Sans Pro and Source Code Pro.

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.

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

Copyright © 2001-2022. TIBCO Software Inc. All Rights Reserved.