Important Information

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

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

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

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

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

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

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

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

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

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

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

TIBCO Software Inc. Confidential Information
# Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figures</td>
<td>5</td>
</tr>
<tr>
<td>TIBCO Documentation and Support Services</td>
<td>6</td>
</tr>
<tr>
<td>Overview</td>
<td>7</td>
</tr>
<tr>
<td>Key Concepts</td>
<td>7</td>
</tr>
<tr>
<td>Layout of the Concepts Guide</td>
<td>9</td>
</tr>
<tr>
<td>General Concepts</td>
<td>10</td>
</tr>
<tr>
<td>Applications</td>
<td>10</td>
</tr>
<tr>
<td>Modules</td>
<td>11</td>
</tr>
<tr>
<td>Application Modules</td>
<td>11</td>
</tr>
<tr>
<td>Shared Modules</td>
<td>12</td>
</tr>
<tr>
<td>Processes</td>
<td>13</td>
</tr>
<tr>
<td>Process Packages</td>
<td>15</td>
</tr>
<tr>
<td>Activities</td>
<td>15</td>
</tr>
<tr>
<td>Palettes</td>
<td>16</td>
</tr>
<tr>
<td>Transitions</td>
<td>16</td>
</tr>
<tr>
<td>Shared Resources</td>
<td>17</td>
</tr>
<tr>
<td>Mapping Concepts to a Sample: File Poller</td>
<td>18</td>
</tr>
<tr>
<td>Additional General Concepts</td>
<td>19</td>
</tr>
<tr>
<td>Groups</td>
<td>19</td>
</tr>
<tr>
<td>Properties</td>
<td>20</td>
</tr>
<tr>
<td>Conversations</td>
<td>22</td>
</tr>
<tr>
<td>Fault Handlers</td>
<td>22</td>
</tr>
<tr>
<td>Event Handlers</td>
<td>23</td>
</tr>
<tr>
<td>Components</td>
<td>24</td>
</tr>
<tr>
<td>Component Services</td>
<td>24</td>
</tr>
<tr>
<td>Component References</td>
<td>24</td>
</tr>
<tr>
<td>Services</td>
<td>25</td>
</tr>
<tr>
<td>Operations</td>
<td>25</td>
</tr>
<tr>
<td>SOAP Services</td>
<td>25</td>
</tr>
<tr>
<td>REST Services</td>
<td>26</td>
</tr>
<tr>
<td>Mapping Concepts to a Sample: Mortgage Broker Service and Client</td>
<td>27</td>
</tr>
<tr>
<td>Mapping Concepts to a Sample: Managing Books for a Bookstore</td>
<td>28</td>
</tr>
<tr>
<td>Design-time Concepts</td>
<td>29</td>
</tr>
<tr>
<td>Runtime Concepts</td>
<td>31</td>
</tr>
<tr>
<td>Administration Concepts</td>
<td>33</td>
</tr>
<tr>
<td>Application Archives</td>
<td>34</td>
</tr>
<tr>
<td>Domains</td>
<td>34</td>
</tr>
<tr>
<td>AppSpaces</td>
<td>35</td>
</tr>
</tbody>
</table>
Figures

Elements of TIBCO ActiveMatrix BusinessWorks Applications .......................................................... 10
Structure of an Application Module .................................................................................................. 12
Relationship Between Application Modules and Shared Modules .................................................. 12
File Poller Process Diagram ........................................................................................................... 18
Relationship Between Properties .................................................................................................. 21
Loan Application Provider Sample Using Conversations .................................................................. 22
Sample Fault Handlers .................................................................................................................... 23
TIBCO Business Studio Workbench ................................................................................................ 29
Application Node Architecture ........................................................................................................ 31
File System Manifestation of a Domain ........................................................................................... 35
File System Manifestation of an AppSpace ...................................................................................... 36
File System Manifestation of an AppNode ....................................................................................... 37
TIBCO Documentation and Support Services

All TIBCO documentation is available in the TIBCO Documentation Library, which can be found here:

https://docs.tibco.com

Product-Specific Documentation

The following documents for TIBCO ActiveMatrix BusinessWorks can be found in the TIBCO Documentation Library:

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

The following documents provide additional information and can be found in the TIBCO Documentation Library:

- TIBCO Enterprise Administrator User’s Guide

How to Contact TIBCO Support

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

- For an overview of TIBCO Support, and information about getting started with TIBCO Support, visit this site:
  http://www.tibco.com/services/support

- If you already have a valid maintenance or support contract, visit this site:
  https://support.tibco.com

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

How to Join TIBCOmmunity

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

http://www.tibcommunity.com
Overview

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

The software allows you to create services and integrate applications using a visual, model-driven development environment, and then deploy them in the ActiveMatrix BusinessWorks™ runtime.

It uses the Eclipse graphical user interface (GUI) provided by TIBCO Business Studio™ to define business processes and generate deployable artifacts in the form of archive files. The deployable artifacts can be deployed and run in the product runtime, and managed using an administration interface such as TIBCO® Enterprise Administrator.

ActiveMatrix BusinessWorks addresses business problems of varying complexity using 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 concepts map provides an overview of the key concepts that you may encounter when working with the product. Some of these concepts are applicable to design perspective or runtime and administration perspective alone, 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 where you execute the applications, and the administration component where you deploy and manage applications in the runtime. The ActiveMatrix BusinessWorks runtime is an ecosystem of entities that can be co-located or distributed. The bwadmin utility and TIBCO Enterprise Administrator allow you to deploy, monitor and manage the applications.
ActiveMatrix BusinessWorks is based on open architecture, flexibility, modularity, and support for standards.

**Flexibility**

ActiveMatrix BusinessWorks is designed to make adding, upgrading, and swapping of business components easy.

Flexible architecture is demonstrated through:

- A zero coding model that allows the users to 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 runtime 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 which allow you to 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 runtime.
• Extensible TIBCO Enterprise Administration based administration framework.

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

Layout of the Concepts Guide

The Concepts Guide presents the design-time, runtime, and administration concepts that are useful to developers and administrators. These concepts are described in one of more of the following sections:

• **General Concepts**: Explains the essential concepts such as applications, application modules, shared 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.
• **Runtime Concepts**: Explains the runtime concepts such as AppNodes, process instances, and jobs.
• **Administrative 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 are included after introducing the key concepts required to implement solutions using the different integration styles. These sections 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. An application typically contains one application module, which consists of one or more processes, and zero or more shared modules. 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.

Applications are developed using many features available in the product and can range from simple to very complex ones. An ActiveMatrix BusinessWorks application contains one 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). A process that is responsible for initiating the business logic at runtime is used to implement a component in an application module.

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

The term module is used interchangeably with OSGi bundle.

Elements of TIBCO ActiveMatrix BusinessWorks Applications

Once an application is developed, you can either run or debug directly in TIBCO Business Studio, 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 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.

- **Shared modules:** The smallest unit of resources that is named, versioned, and packaged as part of an application and can be used by other modules that are part of the same application. A shared module cannot be deployed by itself; it must be included as part of an application module.

Application Modules

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 typically contains one or more ActiveMatrix BusinessWorks processes. An application module is configured and represented in TIBCO Business Studio, 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 WSDL files that provide the service’s name, 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.

- **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.

  The component section in the Module Descriptor allows you 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 includes the required JAR files in this folder.

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

The application modules cannot export their functionality to other modules.
Shared Modules

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

Shared modules export their functionality (processes, shared resources, and schema namespaces) to application modules or to other shared modules. This means that other modules in the system may 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 a 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.

Two modules with the same namespace cannot be used in the same application.

Processes

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

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

**Parent Process**

A process can call another process. The calling process is referred to as a caller process or a parent process.

**Subprocesses**

A process can also be called by another process. The called process is referred to as a subprocess or a child process. A parent process can call the subprocesses into two ways: in-line and non in-line. At runtime, in-line subprocesses are executed on the same engine thread as the caller process while the non in-line subprocesses spawn new engine threads and are executed on the new threads.

**Component Process**

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

A component process is designed to react to various events and these events are triggered by Process Starters and Signal-Ins.

**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. 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. 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.
A simple business process can be developed by adding activities in sequence, and then 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, in-line subprocesses do not create a new job, but are executed on the job created by their calling process.

See Design-time Concepts for details about the TIBCO Business Studio development environment.

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 provides various technology specific palettes that allows you to build a business process.

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

Detailed descriptions of palettes are available in the 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. Furthermore, activities can also state the faults they can throw at runtime and this allows the ActiveMatrix BusinessWorks 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* and they block the execution of the process until the activity task is complete. Signal-in activities are always blocking.

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

- **Process Starter Activities** are configured to react to events and 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 Receive process starter activity starts a process when an HTTP request is received.

- **Signal-in Activities** wait for an asynchronous event in a process and then proceed with executing the process instance when an appropriate event is received. Signal-in activities require conversations to be configured. See Conversations for details.
Palettes

Palettes group activities that perform similar tasks. TIBCO Business Studio 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. 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, the Palettes view 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.

Transitions

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

Transitions are represented by an arrow between two activities. The arrows are unidirectional and you cannot draw a transition to a previously executed activity; control flow in a process must proceed sequentially, beginning with the starting activity and ending with the last activity in the process.

You can draw transitions from one activity to many other activities. 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 be classified into transitions without conditions, transitions with conditions, and transitions with errors.

- **Transitions Without Conditions**: Control flows from one activity to the next automatically, without any conditions.
- **Transitions With Conditions**: When an activity completes processing, conditions specified on the transitions originating from that activity are evaluated to determine whether the transition to the next activity should be taken or not. All transitions whose conditions are met will be taken.
- **Error Transitions**: Special transitions that specify the activities to execute in case of an error. When configuring an activity, you can specify one transition out of the activity that is to be taken in case of an error, and the activities to be executed following the error transition.
Shared Resources

Shared resources are used to define a resource that contains 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, and so on 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 outside the application module. However, when defined in a shared module, they are visible 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 and 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 it was allocated to.

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 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 one module.
Mapping Concepts to a Sample: File Poller

The concepts introduced in sections Application Concepts and Basic Developer Concepts enable you to understand and design a simple business process such as the File Poller example.

Pre-requisites

The File Poller sample demonstrates the concepts introduced in 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

See the *Getting Started* guide for step-by-step instructions to create and test the File Poller process.

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 BusinessWorks Adapters and activities from other palettes such as JMS, JDBC, FTP.
Additional General Concepts

This section introduces 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 execute until the condition evaluates to true.

- **Repeat on Error**: This loop involves a retry mechanism: if any activity in the loop throws 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: it continues to execute as long as the condition holds true, and stops when the condition becomes false.

See Design-time Concepts for details about the TIBCO Business Studio development environment.

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, process properties, and activity configuration properties.

Properties defined in the inner layer can reference a property defined at the parent layer. For example, a process property can reference a module property instead of providing a literal value. Private properties are not visible to the encapsulating layers.

The following diagram illustrates the relationship between the different types of properties:
Features of Process, Module, and Application Properties

<table>
<thead>
<tr>
<th></th>
<th>Scope/Visibility</th>
<th>Values</th>
<th>Additional Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Process Properties</td>
<td>Visible within a process.</td>
<td>Literal, module property reference, or a shared resource reference.</td>
<td>None.</td>
</tr>
<tr>
<td>Module Properties</td>
<td>Visible within the module.</td>
<td>Literal or a shared resource reference.</td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>Not visible or changeable from TIBCO Enterprise Administrator.</td>
<td>Any value for a private module property defined in the profile is ignored.</td>
<td></td>
</tr>
<tr>
<td>Application Properties</td>
<td>Displays all the module properties in the application. These properties are visible in TIBCO Enterprise Administrator.</td>
<td>Literal.</td>
<td>Overrides module properties, thus enabling you to use different values for the same module.</td>
</tr>
<tr>
<td></td>
<td>Cannot add new properties at application level.</td>
<td>Profiles can be used to specify a new set of values for the same application.</td>
<td></td>
</tr>
</tbody>
</table>

See Design-time Concepts for details about the TIBCO Business Studio development environment.
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.

Loan Application 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.

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 are defined at the scope level, allowing 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 will not complete 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 WriteFile 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 will return 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. See **Error Transitions** for details.

**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 will be received. The implementation of 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 are always attached to a scope and executed in parallel with the process so that they are associated with an operation that is a part of the process. Each event handler is associated with one process and has access to that process. However, it can be repeated multiple times during the process execution.

Event handlers can be defined on two different levels:

- **Process level** - When defined on the process level, it cancels the job.
- **Scope level** - When defined on the scope level, it typically cancels the scope. Instead it can be configured to timeout.
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, then that process service is exposed as a component service. The component service then needs to be configured using bindings such as SOAP. The service-centric architecture in TIBCO ActiveMatrix BusinessWorks 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 still 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 then 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. The 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 sort of 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 the roles of 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 key features of REST architectural style supported are:

- **Client-server architecture**: Provides a separation of concerns and implementation details between clients and servers.
- **Stateless communication**: Ensures that each request contains all of the information required to understand it independent 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.
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 going through 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 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, 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 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, an Eclipse-based integration development environment that is used to design and test ActiveMatrix BusinessWorks applications.

TIBCO Business Studio provides a common modeling, implementation, and deployment environment for different types of applications. It provides Eclipse extensions such as editors, palettes, and so on. Business analysts can capture, design and model all aspects of a business process, including the underlying organization and data models. Solution designers can implement the process as an executable application, then deploy the application at runtime for execution.

TIBCO Business Studio Development Environment

TIBCO Business Studio 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 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 BusinessWorks Application Module, Create a new BusinessWorks 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 launches the Modeling 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 ActiveMatrix BusinessWorks 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 on an ActiveMatrix BusinessWorks 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. See Palettes for details.

Testing and Debugging

TIBCO Business Studio 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 ActiveMatrix BusinessWorks applications.

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

![Application Node Architecture Diagram]

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 the 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, ActiveMatrix BusinessWorks engine starts the underlying processes.

The binary file named bwappnode is packaged under the TIBCO_HOME/bw/6.1/bin directory.

Each AppNode is associated with an AppSpace. See Administration Concepts for more information about AppSpaces.

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 fire off 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 (HTTP, JMS, etc), 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 unique id and it is referred to as "JobId".
When the business logic is spread across multiple processes, multiple process instances are created and executed in conjunction to a particular event. Even though these are separate process instances they are all working 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 of the process instances that are part of the same job will have the same JobId. A component process instance and all of its in-line subprocess instances are also considered to be 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.

TIBCO ActiveMatrix BusinessWorks provides a flexible framework that allows you to scale your runtime environment as needed. The runtime also 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:

- **Application Archive** is the deployment unit for an application that is generated in TIBCO Business Studio.
- **Domain** is a logical group that provides an isolated environment for applications and their resources to reside.
- **AppSpace** is a group of one or more AppNodes, which are runtime entities that host ActiveMatrix BusinessWorks applications. AppSpaces are contained within a domain.
- **AppNode** is a runtime entity that hosts ActiveMatrix BusinessWorks applications. AppNodes are contained in an AppSpace.
- **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 *Administration Architecture* illustration below, domain M1 spans two machines, Machine A and Machine B. Domain N1 is on Machine A. Domain M1 contains two AppSpaces; one of the AppSpaces S2 spans both the machines. The bwagent on Machine A is configured to interact with the bwagent on Machine B through the datastore.

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 will autoreregister 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.

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 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.

ActiveMatrix BusinessWorks does not support executing different versions of applications within the same AppSpace.

See Administration guide for more information.

Application Archives

An application archive is a deployment unit for an application that is generated in TIBCO Business Studio. It is the only artifact that is handed from the design phase to the runtime as it contains all the bundles and metadata that is required to deploy and run the application.

Applications are developed using the features available in TIBCO Business Studio and can range from simple to very complex. An ActiveMatrix BusinessWorks application consists of an application module (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 applications can also contain OSGi bundles that do not contain ActiveMatrix BusinessWorks artifacts.

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

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 then deployed 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.

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` 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.

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 elastic, 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. For more information, see Managing AppSpaces and AppNodes.

Following diagram shows AppSpace1 on Machine 1, AppSpace2 on Machine 2, and AppSpace3 spanning Machine 1 and Machine 2:
An AppSpace manifests as a folder, `appspace_name`, in the file system and is located in the `<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**

![Diagram of AppSpace manifest]

**AppNodes**

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

Each application that is deployed into a 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. For more information, see Managing 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**

```
- appnodes
  - MyAppSpace1
    - MyAppNode1
      - bin
      - config
        - configstorage
      - org.eclipse.osgi
        - manager
      - bundles
      - log
```

**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.

When multiple bwagents belong to a network and one of the system fails due to some reason, the failed system can be restored after a restart. The bwagent restores the system by synchronizing data from the datastore with the local file system.

There are multiple ways to access the bwagent: the `bwadmin` command line console, the TIBCO Enterprise Administrator UI, or the REST API.