**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 LICENSE.PDF) 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.

TIB, TIBCO, TIBCO ActiveMatrix, TIBCO Adapter, TIBCO Administrator, TIBCO AutoMeditate, TIBCO Enterprise Message Service, ActiveMatrix, AutoMeditate, Predictive Business, Information Bus, The Power of Now, and TIBCO Rendezvous are either registered trademarks or trademarks of TIBCO Software Inc. in the United States and/or other countries.

EJB, Java EE, J2EE, and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. 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. PLEASE SEE THE README.TXT 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 © 2005-2009 TIBCO Software Inc. ALL RIGHTS RESERVED.

TIBCO Software Inc. Confidential Information
## Contents

### Figures
- At Most Once QoS
- At Least Once QoS
- Best Effort QoS

### Tables
- MEP Support

### Preface
- Related Documentation
  - TIBCO ActiveMatrix Documentation
  - Other TIBCO Product Documentation
  - Third Party Documentation
- Typographical Conventions
- How to Contact TIBCO Support

### Chapter 1 Introduction
- Service-Oriented Architecture
- Challenges of Service Delivery
- TIBCO ActiveMatrix Services
  - Service Description
  - Service Life Cycle
- TIBCO ActiveMatrix Platform Overview
  - Product Family
  - TIBCO ActiveMatrix Service Life Cycle
  - Service Development
  - Service Execution
  - Service Administration

### Chapter 2 Messaging
- Overview
- Message Exchange Patterns
- Messaging Styles
  - MEP Support
- Messaging Quality of Service Options
  - Best Effort QoS
  - At Least Once QoS
  - At Most Once QoS
Chapter 3   Servers. ................................................................. 29
Messaging Server ................................................................. 30
Fault Tolerance .................................................................. 30
Optional Servers ................................................................. 31
   TIBCO ActiveMatrix Registry ........................................... 31
   TIBCO ActiveMatrix Runtime UDDI Server ...................... 31
   TIBCO ActiveMatrix Policy Manager and Policy Agent .... 31

Chapter 4   Development Tools .............................................. 33
Development Tools .............................................................. 34
   TIBCO Business Studio Overview ................................... 34
   ActiveMatrix Resource Wizard ....................................... 35
   Composite Element Editors .......................................... 37
   Mediation Flow Editor .................................................. 38
   Debugger ..................................................................... 40
   Service Assembly Editor .............................................. 42

Chapter 5   Administration Architecture and Tools. ................. 43
Administration Architecture and Tools ................................ 44
   TIBCO ActiveMatrix Administrator Architecture ............. 44
   Enterprise and Environment Administration .................. 46
   Service Administration ................................................ 47

Appendix A   WSDL Overview ............................................... 49
Overview ........................................................................... 50
   WSDL 1.1 and 2.0 Terminology .................................... 50

Glossary ............................................................................. 55

Index .................................................................................. 63
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1</td>
<td>TIBCO ActiveMatrix Product Family</td>
<td>8</td>
</tr>
<tr>
<td>Figure 2</td>
<td>Composite</td>
<td>11</td>
</tr>
<tr>
<td>Figure 3</td>
<td>Mediation Flow</td>
<td>13</td>
</tr>
<tr>
<td>Figure 4</td>
<td>Servicing a Request from an Internal Consumer</td>
<td>17</td>
</tr>
<tr>
<td>Figure 5</td>
<td>Servicing a Request from an External Consumer</td>
<td>17</td>
</tr>
<tr>
<td>Figure 6</td>
<td>ActiveMatrix Node</td>
<td>18</td>
</tr>
<tr>
<td>Figure 7</td>
<td>ActiveMatrix Environment</td>
<td>20</td>
</tr>
<tr>
<td>Figure 8</td>
<td>Deployed Service Units</td>
<td>21</td>
</tr>
<tr>
<td>Figure 9</td>
<td>Monitor &amp; Manage Perspective</td>
<td>22</td>
</tr>
<tr>
<td>Figure 10</td>
<td>TIBCO Business Studio</td>
<td>35</td>
</tr>
<tr>
<td>Figure 11</td>
<td>ActiveMatrix Resource Wizard</td>
<td>36</td>
</tr>
<tr>
<td>Figure 12</td>
<td>Component Property Sheet</td>
<td>37</td>
</tr>
<tr>
<td>Figure 13</td>
<td>Service Property Sheet</td>
<td>38</td>
</tr>
<tr>
<td>Figure 14</td>
<td>Mediation Flow Editor</td>
<td>39</td>
</tr>
<tr>
<td>Figure 15</td>
<td>Debugger</td>
<td>41</td>
</tr>
<tr>
<td>Figure 16</td>
<td>Service Assembly Editor</td>
<td>42</td>
</tr>
<tr>
<td>Figure 17</td>
<td>TIBCO ActiveMatrix Administration Architecture</td>
<td>45</td>
</tr>
<tr>
<td>Figure 18</td>
<td>ActiveMatrix Administrator Graphical Interface</td>
<td>46</td>
</tr>
</tbody>
</table>
Tables

Table 1  General Typographical Conventions .................................................. xii
Table 2  Syntax Typographical Conventions .................................................. xiv
Table 3  MEPs and Messaging Styles ............................................................... 26
Table 4  Structure of the View for a Mediation Flow ...................................... 39
Table 5  WSDL Terminology ........................................................................... 50
Table 6  WSDL Examples ................................................................................ 51
Preface

TIBCO ActiveMatrix® software is a scalable and extensible platform for developing, deploying, and managing applications that conform to a service-oriented architecture.

Topics

- Related Documentation, page x
- Typographical Conventions, page xii
- How to Contact TIBCO Support, page xv
Related Documentation

This section lists documentation resources you may find useful.

TIBCO ActiveMatrix Documentation

The following documents form the core of the TIBCO ActiveMatrix documentation set:

- **TIBCO ActiveMatrix Concepts**: Read this manual before reading any other manual in the documentation set. This manual describes terminology and concepts of the ActiveMatrix platform. The other manuals in the documentation set assume you are familiar with the information in this manual.
- **TIBCO ActiveMatrix Composite Editor User’s Guide**: Read this manual to learn how to develop and package ActiveMatrix composites.
- **TIBCO ActiveMatrix Administration**: Read this manual to learn how to manage the ActiveMatrix runtime and deploy and manage ActiveMatrix services.

Other TIBCO Product Documentation

You may find it useful to read the documentation for the following TIBCO products:

- TIBCO ActiveMatrix® Service Bus
- TIBCO ActiveMatrix® Service Grid
- TIBCO ActiveMatrix® BusinessWorks
- TIBCO ActiveMatrix® Registry
- TIBCO ActiveMatrix® Policy Manager
- TIBCO Enterprise Message Service™
- TIBCO Adapter™
- TIBCO Administrator™
- TIBCO Hawk®
- TIBCO Rendezvous®
Third Party Documentation

TIBCO ActiveMatrix software supports the following standards:

- Service Component Architecture
  [http://www.osoa.org/display/Main/Service+Component+Architecture+Specifications](http://www.osoa.org/display/Main/Service+Component+Architecture+Specifications)

- World Wide Web Consortium web services activity
  [http://www.w3.org/2002/ws/](http://www.w3.org/2002/ws/)
  - Simple Object Access Protocol (SOAP) 1.1 W3C Note
  - WSDL 1.1 W3C Note
    [http://www.w3.org/TR/wsd1](http://www.w3.org/TR/wsd1)

- OASIS
  - UDDI Version 3 OASIS Standard
The following typographical conventions are used in this manual.

### Table 1  General Typographical Conventions

<table>
<thead>
<tr>
<th>Convention</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIBCO_HOME</td>
<td>Many TIBCO products are installed within the same home directory. This directory is referenced in documentation as TIBCO_HOME. The value of TIBCO_HOME depends on the operating system. For example, on Windows systems the default value is C:\tibco.</td>
</tr>
<tr>
<td>ENV_NAME</td>
<td>Other TIBCO products are installed into an installation environment. A product installed into an installation environment does not access components in other installation environments. Incompatible products and multiple instances of the same product must be installed into different installation environments. An installation environment consists of the following properties:</td>
</tr>
<tr>
<td>ENV_HOME</td>
<td>• <strong>Name</strong> Identifies the installation environment. The name is appended to the name of Windows services created by the installer and is used in the path to the product in the Windows Start &gt; All Programs menu. This directory is referenced in documentation as ENV_NAME.</td>
</tr>
<tr>
<td>AMX_HOME</td>
<td>• <strong>Description</strong> Provides information about what the environment contains or is used for.</td>
</tr>
<tr>
<td>AMX_ADMIN_HOME</td>
<td>• <strong>Path</strong> The directory into which the product is installed. This directory is referenced in documentation as ENV_HOME. The value of ENV_HOME depends on the operating system. For example, on Windows systems the default value is C:\tibco.</td>
</tr>
</tbody>
</table>

TIBCO ActiveMatrix installs into a directory inside ENV_HOME. This directory is referenced in documentation as AMX_HOME. The value of AMX_HOME depends on the operating system. For example, on Windows systems the default value is C:\tibco\amx.  

TIBCO ActiveMatrix Administrator installs into a directory inside ENV_HOME. This directory is referenced in documentation as AMX_ADMIN_HOME. The value of AMX_ADMIN_HOME depends on the operating system. For example, on Windows systems the default value is C:\tibco\amxadministrator. |

<table>
<thead>
<tr>
<th>code font</th>
<th>Code font identifies commands, code examples, filenames, pathnames, and output displayed in a command window. For example:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Use MyCommand to start the foo process.</td>
</tr>
<tr>
<td>Convention</td>
<td>Use</td>
</tr>
<tr>
<td>-------------------</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>bold code font</strong></td>
<td>Bold code font is used in the following ways:</td>
</tr>
<tr>
<td></td>
<td>• In procedures, to indicate what a user types. For example: Type <strong>admin</strong>.</td>
</tr>
<tr>
<td></td>
<td>• In large code samples, to indicate the parts of the sample that are of particular interest.</td>
</tr>
</tbody>
</table>
|                   | • In command syntax, to indicate the default parameter for a command. For example, if no parameter is specified, **MyCommand** is enabled:  
|                   | **MyCommand [enable | disable]**                                                             |
| italic font       | Italic font is used in the following ways:                           |
|                   | • To indicate a document title. For example: See *TIBCO ActiveMatrix BusinessWorks Concepts*. |
|                   | • To introduce new terms For example: A portal page may contain several portlets. *Portlets* are mini-applications that run in a portal. |
|                   | • To indicate a variable in a command or code syntax that you must replace. For example: **MyCommand pathname** |
| Key combinations  | Key name separated by a plus sign indicate keys pressed simultaneously. For example: Ctrl+C. |
|                   | Key names separated by a comma and space indicate keys pressed one after the other. For example: Esc, Ctrl+Q. |
| Note icon         | The note icon indicates information that is of special interest or importance, for example, an additional action required only in certain circumstances. |
| Tip icon          | The tip icon indicates an idea that could be useful, for example, a way to apply the information provided in the current section to achieve a specific result. |
| Warning icon      | The warning icon indicates the potential for a damaging situation, for example, data loss or corruption if certain steps are taken or not taken. |
Table 2  Syntax Typographical Conventions

<table>
<thead>
<tr>
<th>Convention</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ ]</td>
<td>An optional item in a command or code syntax. For example: MyCommand [optional_parameter] required_parameter</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>{ }</td>
<td>A logical group of items in a command. Other syntax notations may appear within each logical group. For example, the following command requires two parameters, which can be either the pair param1 and param2, or the pair param3 and param4. MyCommand {param1 param2}</td>
</tr>
<tr>
<td></td>
<td>In the next example, the command requires two parameters. The first parameter can be either param1 or param2 and the second can be either param3 or param4: MyCommand {param1</td>
</tr>
<tr>
<td></td>
<td>In the next example, the command can accept either two or three parameters. The first parameter must be param1. You can optionally include param2 as the second parameter. And the last parameter is either param3 or param4. MyCommand param1 [param2] {param3</td>
</tr>
</tbody>
</table>
How to Contact TIBCO Support

For comments or problems with this manual or the software it addresses, please 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.
Chapter 1  

Introduction

TIBCO ActiveMatrix® is an extensible platform that supports the development, deployment, and management of distributed applications that conform to a service-oriented architecture (SOA).

ActiveMatrix® provides tools for developing and packaging distributed applications, a distributed service execution environment, and tools for managing the runtime environment and the services.

This chapter discusses the motivation for and nature of SOA, the challenges of service delivery, the nature of TIBCO ActiveMatrix services, and the components of TIBCO ActiveMatrix.

Topics

- Service-Oriented Architecture, page 2
- Challenges of Service Delivery, page 3
- TIBCO ActiveMatrix Services, page 5
- TIBCO ActiveMatrix Platform Overview, page 8
Service-oriented architecture (SOA) is a software architecture in which applications and data are decomposed into discrete, independent services. Decomposing applications into services allows enterprise application components to be reused and integrated in flexible and efficient ways. Enterprises achieve agility by virtue of discrete, independent service deployment because they can plug in and out services as their needs dictate.

Traditionally, enterprise applications have been developed by two types of developers: integration developers and application developers.

Integration developers use enterprise application integration platforms. Such platforms provide visual tools for modelling business processes and data-centric approaches to tie systems together. These platforms have a low reliance on standards or APIs, which limits the developer’s ability to extend the platform.

Application developers use application development platforms—typically based on standards-based APIs and component frameworks such as the Java EE platform—to create applications that solve vertical business problems. Such platforms are flexible, but they usually offer weak support for integrating with applications created for platforms that support a different development model.

The adoption of SOA is accelerating due to the emergence of standards-based integration technologies such as web services and XML. In turn, SOA is precipitating the integration of the two types of enterprise application development. Mirroring the core competencies of the two different types of developers, building SOA-based applications involves two types of development activities:

- **Service composition** combines existing services to create new applications or services. Composition is often used to capture business processes and typically requires orchestration of services by a business process engine.
- **Service creation** exposes existing business functions as services or creates new services using code-centric development methods.
Challenges of Service Delivery

The primary challenge in providing business functions is to provide the services that consumers need, want, or are willing to pay for. Another challenge is to do so in as cost-effective a manner as possible.

These primary challenges bring with them a number of secondary challenges. As business requirements, models, and priorities change, businesses need to:

- Add new services, modify existing services, and replace or retire out-dated ones
- Provide legacy services to new consumers and new services to legacy consumers
- Expand the range of available services

To meet the business challenges of providing services, Information Technology departments face a series of challenges:

- **Accommodating heterogeneous software assets** A service platform must support multiple service development languages and deployment platforms.
- **Achieving service-consumer compatibility** In a heterogeneous service environment that spans departments within an enterprise, decades of software development, supply chain and partner companies, and individual consumers, services and consumers can have a number of incompatibilities:
  - **Transport protocols** In a web-services environment, consumers and services can use different transport protocols, for example HTTP and JMS.
  - **Interaction protocols** Interaction protocols like SOAP specify the message interaction needed to obtain a service.
  - **Data structure and formats** The structure and formats of data provided by a consumer must match those expected by a service, and vice versa.
  - **Data content** Sometimes there are expectations about the contents of specific data elements.
- **Reusing services and common features** Instead of repeating development of similar services for different consumers, services can be divided into building blocks and then combined as needed. Common features like logging and fault management can be developed once and used by all services.
- **Managing different versions of services** One size fits all does not work for services. Based on their locale, authorization, time of day, or contents of a request, to give several examples, consumers could be directed to different versions of services that provide different data. Consumers using different versions of client software can also be routed to different versions of services.
• **Minimizing the disruption of transitions** Replacing a service with a new one could render the service unavailable for older versions of consumer software. Even if compatibility remains, updating numerous clients with the location of the new service is difficult.

• **Easing of development and maintenance** To be cost effective and to reduce quality of service problems, the tools for development and maintenance of services must be easy to use. In the TIBCO ActiveMatrix platform, development and maintenance of certain types of services, known as mediation services, are performed with a graphical user interface, with no coding.

As described in the following sections, services developed and run using TIBCO ActiveMatrix software address all of these challenges.
**TIBCO ActiveMatrix Services**

TIBCO ActiveMatrix is a platform for building and deploying applications based on a service-oriented architecture. This section discusses how ActiveMatrix services are specified and introduces the life cycle of an ActiveMatrix service.

**Service Description**

ActiveMatrix software supports an application architecture based on SOA principles: applications are composed of services that interact by exchanging messages. ActiveMatrix services are described in documents expressed in Web Services Description Language (WSDL). The WSDL documents specify the messages that are required to access a service.

**Roles**

During any service interaction, each service will adopt one of two roles: provider or consumer. A service provider publishes a WSDL document that describes the services it offers. A service consumer uses the WSDL document to determine the available services and the messages required to access the services.

**Abstract WSDL Documents**

There are two types of WSDL documents: abstract and concrete. An abstract WSDL document defines an abstract messaging model without reference to protocols or encodings. This abstract model contains the following elements:

- **Definitions** is the root element. It enumerates the namespaces referenced in the WSDL document and contains all other elements.
- **Types** describe the data types of the objects that may be passed in messages.
- **Messages** consist of one or more logical parts. Each part is associated with a type from a type system using a message-typing attribute.
- **Parts** a mechanism for describing the logical abstract content of a message.
- **Operations** are composed of sequences of messages. The direction of the messages (input or output) is from the perspective of the service provider.
- **Port types** (also referred to as interfaces) are collections of operations.

The following WSDL document fragment contains the abstract WSDL elements for a stock quote service.

```xml
<definitions name="StockQuote"
    targetNamespace="http://ns.tibco.com/StockQuote"
    xmlns:tns="http://ns.tibco.com/StockQuote"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns="http://schemas.xmlsoap.org/wsdl/">

    <types>
```

TIBCO ActiveMatrix Concepts
Concrete WSDL Documents

A concrete WSDL document contains the abstract definitions and the communication protocols and data formats by which the operations defined in the abstract WSDL document can be invoked. The concrete WSDL document adds the following elements to the abstract WSDL document:

- **Bindings** connect a port type to a protocol and data format
- **Ports** (also referred to as endpoints) are comprised of a binding and a network address
- **Services** are collections of ports

The following WSDL document fragment contains the concrete WSDL elements of the stock quote service.

```
<binding name="StockQuoteSoapBinding"
  type="tns:StockQuotePortType">
  <soap:binding style="document"
    transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="getQuote">
    <input message="tns:QuoteInput"/>
    <output message="tns:QuoteOutput"/>
  </operation>
</binding>

<service name="StockQuoteService">
  <port name="StockQuotePort"
```
For more information on WSDL, see Appendix A, WSDL Overview.

Within the ActiveMatrix platform, the abstract WSDL document is the basis of all interactions. Using an abstract service model ensures that consumers and providers are decoupled; they share only the abstract service definition, not endpoint information. Decoupling isolates the consumer from the particulars of the service provider's implementation and location.

**Service Life Cycle**

In common with other enterprise application platforms, ActiveMatrix services observe a multiphase life cycle. Architects design enterprise applications as collections of interacting services. Developers implement services based on the technologies provided by one or more runtime engines and assemble the services into packages. Administrators deploy the packages into the ActiveMatrix runtime engines, and manage and monitor service execution to ensure adherence to service level business agreements.
Product Family

The TIBCO ActiveMatrix product family is illustrated in Figure 1.

Figure 1  TIBCO ActiveMatrix Product Family

The ActiveMatrix service development and deployment products and their constituent parts are:

- **TIBCO ActiveMatrix Service Bus**  TIBCO Business Studio™ (including Composite Editor and Mediation Flow Editor), SOAP, Adapter, JMS, and EJB binding types and Mediation implementation type, TIBCO Enterprise Message Service, TIBCO ActiveMatrix Administrator, TIBCO Management Daemon, and TIBCO ActiveMatrix Policy Agent.
• **TIBCO ActiveMatrix Service Grid**  TIBCO ActiveMatrix Service Bus and Java, .NET, Ruby, and C++ implementation types.

The ActiveMatrix products providing enhanced service orchestration and governance are:

- **TIBCO ActiveMatrix BusinessWorks**  Service and process orchestration and service creation and integration.
- **TIBCO Adapter**  Interaction with third-party services
- **TIBCO ActiveMatrix Lifecycle Governance**  Service registration and inquiry
- **TIBCO ActiveMatrix Service Performance Manager**  Service and infrastructure monitoring and management
- **TIBCO ActiveMatrix Policy Manager**  Distribution of policies

**TIBCO ActiveMatrix Service Life Cycle**

The TIBCO ActiveMatrix platform addresses the entire SOA-based application life cycle: development, execution, and administration. This section describes the parts of ActiveMatrix that address the various phases of the service life cycle:

- **Service Development**  on page 9
- **Service Execution**  on page 15
- **Service Administration**  on page 19

**Service Development**

The phases of the ActiveMatrix service development process are covered in the following sections:

- **Design**  on page 10
- **Implementation**  on page 11
- **Configuration**  on page 14
- **Packaging**  on page 15

The ActiveMatrix development tools consist of TIBCO Business Studio and a set of ActiveMatrix plug-ins for TIBCO Business Studio and Microsoft Visual Studio. For an overview of the ActiveMatrix development tools, see Chapter 4, Development Tools, on page 33.
Design

In ActiveMatrix, the output of the design phase are composites. A *composite* exports a cohesive set of business functions as services. Composites support services implemented with a variety of technologies and the encapsulation of service implementations into a business solution. Composites and their exported services typically are constructed by service architects.

A composite contains one or more components, which implement the business functions provided by the composite. Components export their business functions as *component services*, which can either be used by other components within the same composite or which are made available for use outside the composite as *composite services*. Components are implemented by developers.

Components can rely on services provided by other components. These dependencies are called *component references*. References can either be connected to services provided by other components in the same composite or to services provided outside the composite. Services provided outside the composite are invoked via *composite references*.

The connections between references and services are represented by *wires*. In general, connections are point-to-point. However, topics, which represent publish-subscribe messaging channels between services and references, support one-to-many connections. Figure 2 on page 11 illustrates a composite and the relationship between the composite elements contained within it.
During the service implementation phase, developers produce component implementations. The environment for developing ActiveMatrix component implementations depends on the component implementation type. TIBCO ActiveMatrix supports the following component implementation types:

- Java, .NET C# and Visual Basic, and C++. These general purpose programming languages provide a great deal of power and flexibility in developing services, but require specialized skills and greater initial effort to create the services. ActiveMatrix plug-ins for TIBCO Business Studio provide extensive support for developing Java implementations. ActiveMatrix
extensions for Microsoft Visual Studio support the development of .NET C# and Visual Basic implementations.

- TIBCO ActiveMatrix mediation flows. Mediation flows can be developed quickly by reusing prebuilt tasks but are more constrained in the available functions. The Mediation Flow Editor supports the development of mediation flows.

Optional add-ons to ActiveMatrix support other component implementation types, such as ActiveMatrix BusinessWorks processes and TIBCO Adapters.

**Mediation Flows**

Mediation is the process of resolving differences between two parties by playing a role in communications. In software, this sense applies (for example, when bridging between different transport protocols). In addition, other tasks can be performed, for example logging and routing.

TIBCO ActiveMatrix software directly supports the design of mediation flows, which are software programs that perform activities such as routing and transforming messages between consumers and providers of services. Mediation flows:

- Map requests for mediation services to target services, possibly routing requests based on the message content, the context of the messages, or both. Between receiving a request for a mediation service and readying it for delivery to a target service, the mediation flow can perform mediation tasks such as routing, data transformation, and logging.

- Convey messages received by target services to mediation services. Again, mediation tasks can be performed by the mediation flow between receipt by the target reference and delivery to the consumer by the mediation service.

- Manage faults. Mediation flows can throw faults, and can catch and send faults thrown by the mediation flow or by target services.

Figure 3 on page 13 shows how a service consumer invokes a mediation flow and how the mediation flow interacts with target services.
In this figure:

1. A user accesses a program that is the service consumer. The service consumer invokes the Query operation in the mediation service to request information, accessing the mediation service over HTTP.

2. Based on the contents of the message, a Route task within the mediation flow directs the message to one of three target operations, provided by web services in Asia, Europe, and the United States. Transport and interaction-protocol bridging allow communication with the target service providers to proceed.

3. For Asia and Europe, Transform tasks transform the message structure and contents provided by the service consumer to ones that the target service providers can accept.

In summary, mediation provides:

- **Service virtualization** A mediation service hides the location of service providers (including which providers are providing services) and details of
how the services are provided (for example, the transport protocol, message format, and schema) from service consumers. Virtualization enables:

— Location transparency The network locations that provide services are hidden from service consumers (and replaced with other network locations). The actual locations are transparent, that is, not visible to the service consumers.

— Transport and interaction-protocol bridging A composite application containing one or more mediation components can provide a bridge between service consumers and service providers that use different transport protocols, interaction protocols or both.

— Connections between mediation operations and target operations Mediation flows associate each mediation operation with one or more target operations.

• **Content and context-based routing** A routing task placed on the input path in a mediation flow can route service requests to alternative target services based on the message content, message context, mediation flow parameters or all of these. A routing task can also route service requests to Throw Fault tasks, as a means of rejecting requests.

• **Data transformation** When routing a service request to alternative service providers, it might be necessary to transform the message structure, data types, or contents used by the service consumer to the ones expected by the service provider, and vice versa. Transform tasks perform these transformations.

• **Fault management** Mediation flows provide the ability to map fault types reported by service providers to ones understood by service consumers. A mediation flow can also throw faults based on routing cases, rather than sending every message to a service provider. Finally, mediation flows handle runtime faults that occur in the mediation flow itself.

• **Logging** Log tasks can log elements of the message content, message context, mediation flow context or all of these elements.

• **Custom mediation tasks** To provide a mediation feature not present in pre-defined mediation tasks, you can write code that performs a custom mediation task, and incorporate the task in the Mediation Flow Editor using wizards.

**Configuration**

During the service configuration phase, developers configure components and bindings.
A component contains a configured implementation, where the implementation is the code that implements the business functions. The component configures the implementation with specific values for custom properties declared by the component and by wiring component references to services and composite references.

Bindings enable communication between ActiveMatrix and its environment. Service bindings enable consumers outside the ActiveMatrix environment to consume services provided by ActiveMatrix. Reference bindings enable consumers within ActiveMatrix to access external services. TIBCO ActiveMatrix supports the following binding types:

- SOAP
- Adapter
- JMS

Binding configuration involves wiring service bindings to components, the specification of transport properties, and performing binding type-specific configuration.

### Debugging

The ActiveMatrix debugger allows you to test services while you are developing them within TIBCO Business Studio. The debugger invokes a standalone runtime environment independent of the service deployment environment.

### Packaging

Before being deployed, services, components, and references are packaged using TIBCO Business Studio into service assemblies. Service assemblies contain service units, which group components, services, and references based on container type. The service units share the same life cycle as the service assembly. When a service assembly is deployed, the containers register the provided services with the ActiveMatrix runtime.

### Service Execution

The ActiveMatrix runtime supports the execution phase of the service life cycle. The ActiveMatrix runtime consists of containers, Messaging Bus, nodes, and TIBCO servers.
Containers

ActiveMatrix software supports a service execution environment that allows service implementation and access to services via a variety of technologies. In the ActiveMatrix architecture, the runtime engines supporting these various technologies are called containers.

Implementation containers host component implementations. For example:

- A Java container hosts components implemented as Java objects
- A .NET container hosts components implemented as Microsoft Common Language Runtime components

Binding containers host bindings. For example:

- A SOAP container hosts bindings that convert ActiveMatrix messages into SOAP messages and vice versa.
- A TIBCO Adapter container hosts bindings that convert ActiveMatrix messages into TIBCO Adapter messages and vice versa.

Messaging Bus

Messaging Bus is the communications backbone that mediates message exchange between service consumers and providers. When a consumer makes a service request, it is the responsibility of Messaging Bus to locate providers that offer the service and deliver the message to a single provider.

Messaging Bus is distributed, with message handling implemented using TIBCO Enterprise Message Service software. In addition to supporting distributed enterprise applications, Messaging Bus also enables enhanced features such as load balancing and fault tolerance. For further discussion of the ActiveMatrix messaging architecture and functions, see Chapter 2, Messaging, on page 23.

Figure 4 on page 17 shows the message flow that occurs when a consumer in Container B makes a request for a service exported by a provider running in Container A. Messaging Bus locates the service and passes the request from the consumer to the provider.
When ActiveMatrix consumers interact with external services, the service requests are routed by Messaging Bus to reference bindings, which forward the messages to the external providers. Similarly, external consumers interact with service bindings to invoke ActiveMatrix services. Figure 5 shows the message flow that occurs when an external consumer makes a request for a service running in ActiveMatrix. The request is serviced by a service binding running in Container B, which fulfills the request by invoking a component implementation in Container A.
Nodes

A *node* is a Java virtual machine running ActiveMatrix containers and the Messaging Bus. *Figure 6* illustrates a node in which four containers have been installed.

*Figure 6  ActiveMatrix Node*

![ActiveMatrix Node Diagram]

Servers

In addition to Enterprise Message Service server, ActiveMatrix relies on other TIBCO servers to support various aspects of service development and execution:

- **TIBCO ActiveMatrix Registry** is a UDDI-compliant registry service for publishing and discovering services.
- **TIBCO ActiveMatrix Policy Manager** stores policies and distributes them to agents in the messaging infrastructure.

For information on these servers, see *Chapter 3, Servers, on page 29.*
Service Administration

ActiveMatrix administration focuses on configuring the enterprise assets used by the ActiveMatrix runtime, defining the ActiveMatrix infrastructure (environments, nodes, containers, and shared resources) and deploying and monitoring the services running on the infrastructure.

For an overview of the ActiveMatrix administration architecture and tools, see Administration Architecture and Tools on page 43. Nodes and containers have already been introduced in Service Execution on page 15. This section covers configuration of enterprise assets, environments, and shared resources, service deployment and infrastructure and service monitoring.

Configuration

Enterprise Assets

Before setting up an ActiveMatrix runtime an administrator must allocate the enterprise assets within the organization that will be hosting or collaborating with the runtime. The administrator identifies the machines on which the ActiveMatrix runtime will be hosted, identifies other servers such as database and authentication servers, defines software services such as database and messaging connections, and configures the ActiveMatrix user base and assign permissions.

Environments

An ActiveMatrix environment is a collection of ActiveMatrix nodes administered as one entity. Services within an environment communicate via Messaging Bus. Services communicate with services deployed in other environments and external services via service and reference bindings.

Figure 7 on page 20 illustrates an ActiveMatrix environment containing two nodes with their installed containers and the ActiveMatrix tools and servers that support service administration and execution.
Shared Resources

ActiveMatrix supports resources that can be shared between all the containers in a node. Shared Resources are files that contain configuration information for resources that are to be configured in the ActiveMatrix runtime and will be available for sharing among composites and components. Examples of shared resources are JDBC connections, HTTP server configurations. *Shared resource definitions* are specified at the enterprise level and then installed into nodes as *shared resources*. Shared resource definition types include:

- HTTP connections
- Identities
- JDBC connections
- JMS connections
- JNDI configurations
- Rendezvous® connections
- SSL configurations

Composites and components reference the shared resources in *shared resource profiles*. You map shared resource profiles to a node’s shared resources when you deploy a service assembly.
Service Deployment

During deployment, the service units within a service assembly are mapped to and then deployed into their respective containers and the services provided by the service units are activated in the ActiveMatrix containers.

Figure 8 on page 21 shows a service assembly containing five service units. The components in SU1 are implemented with .NET technology. The services in SU2 are implemented with SOAP technology. The components in SU3 are implemented as ActiveMatrix mediation flows. The components in SU4 are implemented with Java technology. The services in SU5 are implemented with JMS technology. When the service assembly is deployed, the service units are deployed into their respective containers.

Figure 8  Deployed Service Units
Monitoring

The ActiveMatrix platform provides extensive support for monitoring the health and performance of infrastructure and deployed services. Figure 9 on page 22 shows the Monitor & Manage perspective in ActiveMatrix Administrator.

Figure 9  Monitor & Manage Perspective

This perspective provides the following subviews:

- **Dashboard** provides information about the overall health of the ActiveMatrix runtime. It presents a summary of the services and infrastructure.

- **Infrastructure** provides details concerning environments, machines, and nodes and the containers, shared resources, and service instances running on the nodes.

- **Service** displays deployed services along with details such as service performance, providers and consumers, UDDI publishing. You can search for a specific service and get details of single service instance.

- **Deployment** displays details about service assemblies and service units.

- **Log** displays the Log Viewer.
Chapter 2  Messaging

This chapter describes the ActiveMatrix architecture and messaging features.

Topics

- Overview, page 24
- Message Exchange Patterns, page 25
- Messaging Styles, page 26
- Messaging Quality of Service Options, page 27
Overview

Messaging Bus is the communications backbone that mediates message exchange between service consumers and providers. When a consumer makes a service request, it is the responsibility of Messaging Bus to locate providers that offer the service and deliver the message to a provider. This message processing model decouples service consumers from service providers.

Messaging Bus, consisting of a messaging layer on each ActiveMatrix node and TIBCO Enterprise Message Service, manages message exchanges:

- Within a node
- Between multiple nodes on the same machine
- Between multiple nodes on multiple machines

If a service is deployed on multiple nodes, Messaging Bus load balances requests among the service instances using a round robin algorithm.

When you deploy a consumer service, you can also specify that a collocated provider service be given precedence. In such configurations, the requests are not load balanced.

In addition to supporting distributed enterprise applications, the use of TIBCO Enterprise Message Service also enables enhanced features, such as fault tolerance. For information on how to configure enhanced features on the Enterprise Message Service server, see Messaging Server on page 30.
Message Exchange Patterns

A provider generates and responds to messages according to the operations defined in the WSDL port types it offers. The WSDL document is always written from the perspective of the provider. That is, if a WSDL document says that the messages are input and then output, the provider first receives a message and then sends a message.

A consumer uses a service, and interprets a WSDL in order to consume a service. The consumer handles messages in the opposite direction of the provider.

A *message exchange pattern* (MEP) defines the sequence and cardinality of messages sent between the provider and the consumer. MEPs contain both normal and fault messages. ActiveMatrix software supports following MEPs:

- **One-Way (In-Only)** A consumer sends a message to a provider.

- **Request-Response (In-Out)** A consumer sends a message to a provider, with expectation of response. The provider sends a response message. The provider may generate a fault if it fails to process the message.

- **Request-Response (Out-In)** A provider sends a message to a consumer, with expectation of response. The consumer sends a response message. The consumer may generate a fault if it fails to process the message.

- **One-Way (Out-Only)** A provider sends message to a consumer.

Specific containers may support a subset of these MEPs. See *TIBCO ActiveMatrix Composite Editor User’s Guide* and *TIBCO ActiveMatrix Service Grid Component Developer’s Guide* for details.
Messaging Styles

ActiveMatrix supports two messaging styles: point-to-point and publish-subscribe.

*Point-to-point* messaging involves a single sender and receiver. It is built on the concept of message queues. Senders send messages addressed to a specific queue; receivers extract messages from the queues established to hold their messages.

*Publish-subscribe* messaging involves multiple publishers and subscribers. Publishers address messages to a specific node in a content hierarchy, called a topic. Publishers and subscribers are anonymous and can dynamically publish or subscribe to the content hierarchy. The system takes care of distributing the messages arriving from a node’s multiple publishers to its multiple subscribers. This type of messaging style is also known as broadcast messaging because messages are received by all interested subscribers.

MEP Support

*Table 3* summarizes ActiveMatrix support for MEPs and messaging styles. Publish-subscribe messaging is compatible only with in-only and out-only MEPs.

*Table 3  MEPs and Messaging Styles*

<table>
<thead>
<tr>
<th>Messaging Style</th>
<th>In-only MEP</th>
<th>In-out MEP</th>
<th>Out-in MEP</th>
<th>Out-only MEP</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Point-to-point</strong></td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td><strong>Publish-subscribe</strong></td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>Y</td>
</tr>
</tbody>
</table>
Messaging Quality of Service Options

Messaging quality of service options determine the guarantees Messaging Bus can make regarding the delivery of messages and the system resources that will be employed to ensure those guarantees. You specify quality of service options at the ActiveMatrix environment level. ActiveMatrix provides the following message quality of service options: best effort, at least once, and at most once.

Best Effort QoS

With best effort, messages
• May be dropped
• May be delivered more than once

In this option, system messages are not sent when messages are received. If the Enterprise Message Service server discards messages for any reason (for example queue limit or authorization failure) then the server does not throw an exception back to the message producer. Both these factors result in higher throughput.

At Least Once QoS

With at least once, messages
• May not be dropped
• May be delivered more than once

In this option, all messages are logged to persistent storage which causes degraded message throughput performance compared to the at most once option.

At Most Once QoS

With at most once, messages
• May be dropped
• May not be delivered more than once

If authorization is turned on, then any message discarded by the Enterprise Message Service server results in an exception being sent back to the message producer. This is the key difference between best effort and at most once quality of service.
Chapter 3  Servers

This chapter provides an overview of the servers that support various aspects of ActiveMatrix service development and execution.

Topics

- *Messaging Server, page 30*
- *Optional Servers, page 31*
Messaging Server

ActiveMatrix Messaging Bus uses TIBCO Enterprise Message Service software as the messaging backbone. Enterprise Message Service server configuration is independent of ActiveMatrix environment configuration; multiple environments may use the same Enterprise Message Service server.

ActiveMatrix software leverages features in Enterprise Message Service software that enable greater robustness while handling service requests. The following section summarizes how to configure fault tolerant Enterprise Message Service servers.

Fault Tolerance

Fault tolerance allows messaging between ActiveMatrix nodes and an Enterprise Message Service server to continue in the event of Enterprise Message Service server failure. To obtain fault tolerance, you configure two Enterprise Message Service servers as primary and backup servers. The primary and backup servers act as a pair, with the primary server accepting client connections and performing the work of handling messages, and the secondary server acting as a backup in case of failure. When the active server fails, the backup server assumes operation and becomes the primary active server.

For fault tolerance, install a pair of EMS servers on two different machines. For example, a primary Enterprise Message Service server on machine M1 and a backup server on machine M2. To configure a pair of Enterprise Message Service servers as a fault tolerant, you must

- Set up shared storage

  The shared storage contains information about client connections and persistent messages. This information enables the backup server to properly assume responsibility for those connections and messages. The shared storage must satisfy certain criteria.

- Set parameters in the server configuration files

For information on the combination of hardware and software that can be used to satisfy the shared storage criteria and the parameters that must be set in configuration files, see TIBCO Enterprise Message Service User’s Guide.
Optional Servers

ActiveMatrix optional servers complement ActiveMatrix platform software.

TIBCO ActiveMatrix Registry

TIBCO ActiveMatrix Registry is a full-featured UDDI registry for discovering and registering services during service development and deployment. TIBCO ActiveMatrix Registry enables service providers to publish service listings and consumers to discover service providers. TIBCO ActiveMatrix Registry provides a full set of tools for browsing and administering the registry.

In addition, you can browse the registry from TIBCO Business Studio and import WSDL files for use in developing services. TIBCO ActiveMatrix Administrator is integrated with TIBCO ActiveMatrix Registry. Once you have deployed a service in you can publish the service to TIBCO ActiveMatrix Registry.

For information on TIBCO ActiveMatrix Registry, see the TIBCO ActiveMatrix Registry documentation set.

TIBCO ActiveMatrix Runtime UDDI Server

TIBCO ActiveMatrix Runtime UDDI Server is a lightweight UDDI server for discovering and registering services during service execution. It provides Ant-based administration scripts.

For information on TIBCO ActiveMatrix Runtime UDDI Server, see the TIBCO ActiveMatrix Runtime UDDI Server documentation set.

TIBCO ActiveMatrix Policy Manager and Policy Agent

A policy is a rule or property that dynamically affects the behavior of a service. Changing policies while a service is running does not stop the service but changes its behavior.

TIBCO ActiveMatrix Policy Manager manages the distribution of policies to ActiveMatrix nodes. It supports the establishment and enforcement of following categories of policies such as:

- Cryptographic
- Security: authentication, authorization, data integrity, and confidentiality
- Logging
TIBCO ActiveMatrix Policy Agent enforces policies at selected endpoints. You can configure which endpoints (internal, external, or none) will be exposed to Policy Agent when you configure service assemblies for deployment.

For information on TIBCO ActiveMatrix Policy Manager and Policy Agent, see the TIBCO ActiveMatrix Policy Manager or Policy Agent documentation set.
Chapter 4  Development Tools

This chapter provides an overview of the ActiveMatrix development tools.

Topics

- Development Tools, page 34
Development Tools

The ActiveMatrix development tools consist of TIBCO Business Studio and a set of ActiveMatrix plug-ins for TIBCO Business Studio and Microsoft Visual Studio. The plug-ins support composite development, debugging, and packaging and mediation flow development.

The following sections provide an overview of the ActiveMatrix development tools using the examples developed in TIBCO ActiveMatrix Service Grid Getting Started. For a step-by-step instructions on how to develop the examples, consult that manual. For detailed task and reference information about the development tools for each type of container, see TIBCO ActiveMatrix Composite Editor User’s Guide, TIBCO ActiveMatrix Service Grid Mediation Design Guide, and TIBCO ActiveMatrix Service Grid Component Developer’s Guide.

TIBCO Business Studio Overview

Figure 10 on page 35 shows TIBCO Business Studio opened to the sample projects and HelloWorld composite from TIBCO ActiveMatrix Service Grid Getting Started. The Project Explorer on the left shows the projects, subfolders, composite file, WSDL files, and service assembly file. The composite editor on the right is open to the HelloWorld composite and composite property sheets.
ActiveMatrix Resource Wizard

The starting point for creating all types of ActiveMatrix projects and assets is the ActiveMatrix Resource Wizard. The resource wizard, shown in Figure 11 on page 36, allows you to select wizards to create:

- ActiveMatrix sample projects
- ActiveMatrix SOA projects
- Composites
• Mediation flows
• Service assemblies

*Figure 11  ActiveMatrix Resource Wizard*
Composite Element Editors

Composite elements are configured in property sheets accessed through the Properties view.

Figure 12 shows the Properties view for the JavaHelloComp Java component.

Figure 12  Component Property Sheet

Figure 13 on page 38 shows the property sheets for the JavaSOAPService service.
Figure 13  Service Property Sheet

Mediation Flow Editor

When you create a mediation flow or open an existing one, TIBCO Business Studio displays the Mediation Flow Editor, which is used to create or modify the mediation flow.

Figure 14 on page 39 shows the Mediation Flow Editor.
Figure 14  Mediation Flow Editor

Table 4 describes the parts of the Mediation Flow Editor, which are displayed in four columns from the left to right.

Table 4  Structure of the View for a Mediation Flow

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mediation Interfaces</td>
<td>Column that contains mediation interfaces that are in the mediation flow. Interfaces and the operations that they contain are displayed. In the Composite Editor, mediation services are created from operations in the mediation interfaces.</td>
</tr>
<tr>
<td>Input, Output, and Fault buttons</td>
<td>Input, Output, and Fault buttons at the top of this column allow you to select between input, output, and fault message types for an operation.</td>
</tr>
</tbody>
</table>
Debugger

The TIBCO Business Studio debugger provides a testing environment for stepping through composite elements and determining the sources of errors. Figure 15 on page 41 shows the debugger in the process of debugging the HelloWorld composite from TIBCO ActiveMatrix Service Grid Getting Started. Breakpoints have been set before and after the Java component executes, and the debugger is stopped at the before breakpoint. In the Variables view on the top-right, the value of the request is being examined.
Figure 15  Debugger
Service Assembly Editor

In order to be deployed, composites must be transformed into service units and service assemblies. Figure 16, shows the service assembly editor opened to the HelloWorld service assembly from TIBCO ActiveMatrix Service Grid Getting Started.

Figure 16  Service Assembly Editor
Chapter 5  **Administration Architecture and Tools**

This chapter provides an overview of the ActiveMatrix administration architecture and tools.

**Topics**

- *Administration Architecture and Tools, page 44*
Administration Architecture and Tools

ActiveMatrix provides TIBCO ActiveMatrix Administrator for enterprise, environment, and service management. ActiveMatrix Administrator supports both graphical and command-line interfaces.

For the servers discussed in Chapter 3, Servers, on page 29, you use server-specific GUI and command-line tools for administration. For information on these tools, see the documentation set for each server.

The following sections provide an overview of TIBCO ActiveMatrix Administrator. To get a quick introduction to the administration tools in practice, see TIBCO ActiveMatrix Service Grid Getting Started. For detailed information about the administration tools, see TIBCO ActiveMatrix Administration.

TIBCO ActiveMatrix Administrator Architecture

Figure 17 on page 45 shows ActiveMatrix Administrator components, and the relationship between ActiveMatrix Administrator, other servers, and ActiveMatrix machines and nodes.
The TIBCO ActiveMatrix Administrator administration architecture consists of the following components:

- **TIBCO ActiveMatrix Administrator Server** Gathers management data from nodes, responds to requests from the ActiveMatrix Administrator graphical and command-line UIs, interacts with the authentication realm server to authenticate users, and interacts with TIBCO Management Daemon to manage nodes.

- **TIBCO ActiveMatrix Administrator Cluster** Groups one or more ActiveMatrix Administrator servers. ActiveMatrix Administrator servers within a cluster share a database and authentication realm and are kept synchronized.

- **ActiveMatrix Database** Stores ActiveMatrix administration data.
• **Authentication Realm**  Manages user authentication data. The authentication realm can be provided either by TIBCO Administrator or by another server or a file.

• **ActiveMatrix Administrator Graphical UI**  Displays the ActiveMatrix Administrator user interface. Figure 18 on page 46 shows the ActiveMatrix Administrator graphical UI welcome page. In ActiveMatrix Administrator, functionality is divided into perspectives. A **perspective** is a set of controls used to carry out a category of administration tasks.

• **ActiveMatrix Administrator Command-Line Interface**  Provides a script-based interface to ActiveMatrix Administrator functions.

• **Management Daemon**  Gathers installation information and exposes ActiveMatrix node life cycle operations.

*Figure 18  ActiveMatrix Administrator Graphical Interface*

---

**Enterprise and Environment Administration**

You administer ActiveMatrix enterprises and environments, shared resource configurations, nodes, containers, and managed resources with ActiveMatrix Administrator graphical and command-line interfaces.
In the graphical interface, enterprise and environment administration is carried out in the Configure Enterprise Assets and Configure an Environment perspectives.

**Service Administration**

You administer ActiveMatrix services with ActiveMatrix Administrator graphical and command-line interfaces. Service administration consists of deployment tasks and monitoring and management tasks. In the graphical interface, these tasks are carried out in the Deploy to an Environment and Monitor & Manage perspectives.

**Service Deployment**

The first phase of service administration is deployment. During deployment, the service units within a service assembly are mapped and then deployed into their respective containers, the services provided by the service units are registered with the ActiveMatrix container, and the service endpoints are activated.

The choice of how to distribute services across nodes is determined by the desired level of service performance and availability. Service performance and availability can be enhanced if you deploy a service unit across multiple nodes, which allows Messaging Bus to distribute requests between the service instances.

**Highly Available Services**

Services deployed on multiple containers are *highly available*; if one container fails, service requests will be handled by one of the remaining containers. No configuration is required to make services highly available. Messaging Bus automatically routes to any available service instance identified in the message exchange.

**Load Balanced Services**

Requests to services deployed on multiple containers are *load balanced* between the available providers. No configuration is required to load balance between services. Messaging Bus uses a round robin algorithm for routing requests to service instances.
Service Monitoring and Management

TIBCO ActiveMatrix Administrator not only allows you to configure and deploy services, but also lets you monitor and manage the deployed services. Monitoring the system performance is not a one-time activity but needs to be performed on a day-to-day basis. The Monitor & Manage perspective of TIBCO ActiveMatrix Administrator keeps track of system health without much overhead.

In the Monitor & Manage perspective you can monitor the overall health and performance of the grid infrastructure, applications, and services. You can monitor performance at various levels such as environment, machine, node, container, service assembly, and service unit.

The monitoring subsystem uses content-based metrics to measure the service performance, availability of services, service usage, and the number of successful to faulty service responses. These metrics provide real-time values by fetching data every minute and updating the values of the metrics. The real-time data is then displayed in a web-based dashboard provided with pre-defined views and visual alerts.
Appendix A  WSDL Overview

This appendix describes the two WSDL specifications in existence and differences in terminology between the specifications.

Topics

- Overview, page 50
Overview

There are two versions of WSDL: 1.1 and 2.0. WSDL 1.1 was published in 2001 and is supported by many tools and implementations. WSDL 2.0 became a W3C recommendation on June 2007. The current release of ActiveMatrix software supports WSDL 1.1.

WSDL 1.1 and 2.0 Terminology

Table 5 summarizes the elements defined in the two WSDL specifications.

The types, part, message, operation, and port type elements constitute the abstract WSDL. The abstract WSDL plus the binding, port, and service elements constitute the concrete WSDL.

Table 5  WSDL Terminology

<table>
<thead>
<tr>
<th>WSDL1.1 Element</th>
<th>WSDL 2.0 Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>definitions</td>
<td>definitions</td>
<td>Root element. Enumerates the namespaces referenced in the WSDL document and contains all other elements.</td>
</tr>
<tr>
<td>types</td>
<td>types</td>
<td>A container for data type definitions using some type system (such as XSD).</td>
</tr>
<tr>
<td>message</td>
<td>NA</td>
<td>An abstract definition of the data being transmitted between service providers and consumers.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>A WSDL 1.1 message contains part elements, each of which is associated with a type. In WSDL 2.0, messages are specified within the operation by referencing a schema definition.</td>
</tr>
<tr>
<td>part</td>
<td>NA</td>
<td>The logical abstract content of a message. For simple content, part represents a message parameter. If the message contents are sufficiently complex, then an alternative syntax may be used to specify the composite structure of the message using the types directly. In this usage, only one part may be specified.</td>
</tr>
<tr>
<td>operation</td>
<td>operation</td>
<td>An exchange of messages between consumer and provider according to a message exchange pattern that is understood by both participants.</td>
</tr>
<tr>
<td>port type</td>
<td>interface</td>
<td>A collection of related operations. A port type or interface can be implemented by more than one service.</td>
</tr>
</tbody>
</table>
### Table 5  WSDL Terminology (Cont’d)

<table>
<thead>
<tr>
<th>Element</th>
<th>WSDL 1.1 Example</th>
<th>WSDL 2.0 Example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>binding</td>
<td>binding</td>
<td>binding</td>
<td>A concrete protocol and data format specification for the operations defined by a particular port type or interface.</td>
</tr>
<tr>
<td>port</td>
<td>endpoint</td>
<td>endpoint</td>
<td>A combination of a binding and a network address.</td>
</tr>
<tr>
<td>service</td>
<td>service</td>
<td>service</td>
<td>A collection of related ports or endpoints. A service implements a port type or interface.</td>
</tr>
</tbody>
</table>

### Table 6  WSDL Examples (Sheet 1 of 3)

<table>
<thead>
<tr>
<th>Element</th>
<th>WSDL 1.1 Example</th>
<th>WSDL 2.0 Example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>&lt;xsd:element name=&quot;QuoteRequest&quot; type=&quot;xsd:string&quot;/&gt;</td>
<td>&lt;xsd:element name=&quot;QuoteRequest&quot; type=&quot;xsd:string&quot;/&gt;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&lt;xsd:element name=&quot;QuoteResponse&quot; type=&quot;xsd:float&quot;/&gt;</td>
<td>&lt;xsd:element name=&quot;QuoteResponse&quot; type=&quot;xsd:float&quot;/&gt;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&lt;/xsd:schema&gt;</td>
<td>&lt;/xsd:schema&gt;</td>
<td></td>
</tr>
</tbody>
</table>
### Table 6  WSDL Examples (Sheet 2 of 3)

<table>
<thead>
<tr>
<th>Element</th>
<th>WSDL 1.1 Example</th>
<th>WSDL 2.0 Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>types</td>
<td>&lt;types&gt;</td>
<td>&lt;types&gt;</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>&lt;/xsd:schema&gt;</td>
<td>&lt;/xsd:schema&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;/types&gt;</td>
<td>&lt;/types&gt;</td>
</tr>
<tr>
<td>part</td>
<td>&lt;part name=&quot;symbol&quot; element=&quot;tns:QuoteRequest&quot;/&gt;</td>
<td>NA</td>
</tr>
<tr>
<td></td>
<td>or</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&lt;part name=&quot;symbol&quot; type=&quot;xs:string&quot; /&gt;</td>
<td></td>
</tr>
<tr>
<td>message</td>
<td>&lt;message name=&quot;QuoteInput&quot;&gt;</td>
<td>NA</td>
</tr>
<tr>
<td></td>
<td>&lt;part name=&quot;symbol&quot; element=&quot;tns:QuoteRequest&quot;/&gt;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&lt;/message&gt;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&lt;message name=&quot;QuoteOutput&quot;&gt;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&lt;part name=&quot;quote&quot; element=&quot;tns:QuoteResponse&quot;/&gt;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&lt;/message&gt;</td>
<td></td>
</tr>
<tr>
<td>operation</td>
<td>&lt;operation name=&quot;getQuote&quot;&gt;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&lt;input message=&quot;tns:QuoteInput&quot;/&gt;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&lt;output message=&quot;tns:QuoteOutput&quot;/&gt;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&lt;/operation&gt;</td>
<td></td>
</tr>
<tr>
<td>port type/</td>
<td>&lt;portType name=&quot;StockQuotePortType&quot;&gt;</td>
<td></td>
</tr>
<tr>
<td>interface</td>
<td>&lt;operation name=&quot;getQuote&quot; ... /&gt;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&lt;/portType&gt;</td>
<td></td>
</tr>
</tbody>
</table>
## Table 6  WSDL Examples (Sheet 3 of 3)

<table>
<thead>
<tr>
<th>Element</th>
<th>WSDL 1.1 Example</th>
<th>WSDL 2.0 Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>binding</td>
<td>&lt;binding name=&quot;StockQuoteSoapBinding&quot; type=&quot;tns:StockQuotePortType&quot;&gt;</td>
<td>&lt;binding name=&quot;StockQuoteSoapBinding&quot; interface=&quot;tns:StockQuoteInterface&quot; type=&quot;<a href="http://www.w3.org/2006/01/wsd">http://www.w3.org/2006/01/wsd</a></td>
</tr>
<tr>
<td></td>
<td>&lt;operation name=&quot;getQuote&quot;&gt;</td>
<td>&lt;operation ref=&quot;tns:getQuote&quot; wsoap:mep=&quot;http://www.w3.org/2003/05/soap/mep/soap-response&quot; /&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;input&gt;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&lt;soap:body use=&quot;literal&quot;/&gt;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&lt;/input&gt;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&lt;output&gt;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&lt;soap:body use=&quot;literal&quot;/&gt;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&lt;/output&gt;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&lt;/operation&gt;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&lt;/binding&gt;</td>
<td></td>
</tr>
<tr>
<td>port/endpoint</td>
<td>&lt;port name=&quot;StockQuotePort&quot; binding=&quot;tns:StockQuoteSoapBinding&quot;&gt;</td>
<td>&lt;endpoint name=&quot;StockQuoteEndpoint&quot; binding=&quot;tns:StockQuoteSoapBinding&quot;&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;soap:address location=&quot;<a href="http://ns.tibco.com/stockquote%22/%3E">http://ns.tibco.com/stockquote&quot;/&gt;</a></td>
<td>address=&quot;<a href="http://ns.tibco.com/stockquote%22/">http://ns.tibco.com/stockquote&quot;/</a> &gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;/port&gt;</td>
<td></td>
</tr>
<tr>
<td>service</td>
<td>&lt;service name=&quot;StockQuoteService&quot;&gt;</td>
<td>&lt;service name=&quot;StockQuoteService&quot; interface=&quot;tns:StockQuoteInterface&quot;&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;port name=&quot;StockQuotePort&quot; binding=&quot;tns:StockQuoteSoapBinding&quot; ... /&gt;</td>
<td>&lt;endpoint name=&quot;StockQuoteEndpoint&quot; binding=&quot;tns:StockQuoteSoapBinding&quot; ... /&gt;</td>
</tr>
<tr>
<td></td>
<td>/port&gt;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&lt;/service&gt;</td>
<td></td>
</tr>
</tbody>
</table>
Glossary

A

authentication realm
Defines the method of storing and retrieving information about ActiveMatrix users and groups. ActiveMatrix Administrator supports the following authentication realms: Local XML File, Database, TIBCO Administrator, and LDAP.

B

binding (WSDL)
A concrete protocol and data format specification for the operations contained in a port (WSDL 1.1) or interface (WSDL 2.0).

C

cluster
A group of ActiveMatrix Administrator servers. Servers in a cluster are maintained as clones, that is, they are automatically kept in an identical state for failover purposes. All servers in a cluster share the same database tables and authentication realm.

component
Represents an implementation of one or more port types. The component attributes are:

- Name
- Implementation type
- Port types provided and consumed by the component
- Links to services that export the provided port types
- Links to partner references or topics that export the consumed port types
- Links to artifacts that implement the provided port types
- Resource profiles
- Properties

component reference
See reference.

component service
See service.

composite
The output of the ActiveMatrix design phase. A container for composite elements. The suffix of a composite file is .composite.

composite element
A child element of a composite. Services, components, references, and topics, wires, and the properties that configure components.

composite reference
See reference.

composite service
See service.
consumer
A software service that sends and receives messages defined by a WSDL document to a provider.

carrier
A software service that sends and receives messages defined by a WSDL document to a provider.

container
An ActiveMatrix runtime entity that hosts component implementations and service bindings.

E

endpoint (WSDL 2.0)
A combination of a binding and a network address. The URL at which a consumer can access a service. An internal endpoint is accessible only to consumers within an ActiveMatrix environment. An external endpoint has a binding that provides access to consumers outside the ActiveMatrix environment. See also port (WSDL 1.1).

environment
A logical grouping of ActiveMatrix nodes administered as one entity.

J

JDBC
A Java API that allows applications to invoke SQL commands to create database tables, access the data stored in a table, and create and manage distributed transactions.

JMS (Java Message Service)
A Java API that allows applications to create, send, receive, and read messages. The JMS API enables communication that is decoupled, asynchronous, and reliable.

JMS can deliver messages to a client as they arrive; a client does not have to request messages in order to receive them. JMS can ensure that a message is delivered once and only once. Lower levels of reliability are available for applications that can afford to miss messages or to receive duplicate messages.

JNDI (Java Naming and Directory Interface)
A Java API that allows applications to store and retrieve named Java objects of any type. In addition, JNDI provides methods for performing standard directory operations, such as associating attributes with objects and searching for objects using their attributes.

HTTP (Hypertext Transfer Protocol)
The Internet protocol used to retrieve hypertext objects from remote hosts. HTTP messages consist of requests from client to server and responses from server to client. HTTP/S is a secure version of HTTP.
K

keystore
A database of private keys and their associated X.509 certificate chains authenticating the corresponding public keys.

keystore entry
An entry in a keystore. An entry can be a trusted certificate entry, which contains a single public key certificate belonging to a trusted party or a key-certificate entry, which contains a private key and a corresponding public key certificate. Each entry in a keystore is identified by an alias.

L

location transparency
Network locations that provide services are hidden from the service consumers (and replaced with some other network location). The actual locations are transparent, that is, not visible to the service consumer.

M

mediation
Performing programmatic tasks on messages in between service providers and the service consumer, to enhance the services provided.

mediation flow
Software program that performs activities between consumers and providers of services, such as routing and transforming messages.

mediation service
Service provided by a mediation component. The service provided combines one or more target services, which provide business functions, with mediation functions such as routing and data transformation.

mediation task
Set of TIBCO Business Studio plug-ins that performs a mediation function. Mediation tasks are placed on mediation paths, and are processed in sequence at runtime to provide mediation logic (some coherent set of mediation tasks designed to support mediation goals, in the same way that business logic supports business goals).

Messaging Bus
The message exchange layer responsible for routing messages between service providers and consumers. Messaging Bus supports performing such message delivery with varying qualities of service depending on application needs and the nature of the messages being delivered. It also performs transaction propagation, policy enforcement, and so on.

message exchange pattern (MEP)
The sequence and cardinality of messages sent between the consumer and the provider. The messages include both normal and fault messages.

N

.NET Framework
A software component that can be added to the Microsoft Windows operating system. It consists of a class library and a runtime environment.

The class library provides solutions to common program requirements and manages the execution of programs written specifically for the framework. The class library covers a range of areas including user interface, data access,
cryptography, numeric algorithms, and network communications. There are several language bindings to the class library, including C#, Visual Basic, and JScript.

The runtime environment is known as the Common Language Runtime (CLR). CLR provides an application virtual machine, so that programmers need not consider the capabilities of the specific CPU that will execute the program.

node
A Java Virtual Machine running ActiveMatrix containers and Messaging Bus.

normalized message
The messaging unit of interoperability between ActiveMatrix containers. It uses abstract WSDL message type definitions and consists of two parts:

- **Content** An XML document that conforms to an abstract WSDL message type, without any protocol encoding or formatting. (This is not a canonical form for the message.)

- **Properties** Data associated with a message gained during the processing of the message. Such properties can include security information (security principal for digital signers of received messages, for example), transaction context information, and container-specific information.

P

perspective
In TIBCO Business Studio, a set and layout of views in the Workbench window.

In ActiveMatrix Administrator, a set of screens and controls used to carry out a category of administration tasks.

point-to-point messaging
A messaging system involving a single sender and receiver. It is built on the concept of message queues. Senders send messages addressed to a specific queue; receivers extract messages from the queues established to hold their messages.

policy
A rule or property that dynamically affects the behavior of a service.

port (WSDL 1.1)
A combination of a binding (WSDL) and a network address. See also endpoint (WSDL 2.0).

port type (WSDL 1.1)
A collection of operations. See also interface (WSDL 2.0).

provider
A software service that conforms to a published WSDL document. The WSDL document describes the services it offers and the messages required to access the services. The provider generates and responds to messages sent by a consumer.

publish-subscribe messaging
A messaging system involving multiple publishers and subscribers. Publishers address messages to a specific node in a content hierarchy, called a topic. Publishers and subscribers are anonymous and can dynamically
publish or subscribe to the content hierarchy. The system takes care of distributing the messages arriving from a node’s multiple publishers to its multiple subscribers. This type of messaging style is also known as broadcast messaging because messages are received by all interested subscribers.

Q

QName
A qualified name, a data type specified by XML Schema. A qualified name consists of a namespace URI and a local part.

queue
See point-to-point messaging.

R

reference
Represents an endpoint consumed by an ActiveMatrix composite element. A composite reference represents the endpoints consumed by a component. A composite reference enables ActiveMatrix composite elements to consume port types implemented outside ActiveMatrix. The reference attributes are:

- Name
- Port types provided by the reference
- Bindings that enable composite elements to access the provided port types
- Resource profiles

S

service
In the WSDL specifications, a collection of ports (WSDL 1.1) or endpoints (WSDL 2.0). See also web service.

In the ActiveMatrix architecture, endpoints provided by an ActiveMatrix composite or composite element. A composite service enables consumers outside the ActiveMatrix environment to access a service provided by a composite element. A composite service represents a service provided by the component. The service attributes are:

- Name
- Port types provided by the service
- Bindings that enable external consumers to access the provided port types
- Links to components, references, or topics that export the provided port types
- Resource profiles

service assembly
An ActiveMatrix deployment package. It contains service units and a descriptor that indicates the container into which each service unit is to be deployed. The suffix of a service assembly file is .saf. See also service unit.

service unit
An ActiveMatrix package that is deployed into a specific type of container. A service unit contains a descriptor file that contains information about the services provided and consumed, implementation artifacts, composite files, shared resource files, and shared variable files.

service virtualization
Virtual services abstract the locations from which services are provided (location transparency) and conceals details of how the service is
provided (transport protocol, interaction protocol, and operations). The presence of the virtual service allows the target services to change without affecting consumers of those services.

**shared resource**
An object that allows services, components, and references to share access to communication, data, naming, and security resources. The shared resource types supported by ActiveMatrix are:

- HTTP connections
- Identity configurations
- JDBC connections
- JMS configurations
- JNDI configurations
- Rendezvous transports
- SSL configurations

**shared resource definition**
A set of properties that define how to create a shared resource.

**shared resource profile**
Represents a reference to a shared resource.

**SOA (service-oriented architecture)**
A software architecture in which applications and data are decomposed into discrete, operationally independent services, which can be executed in a highly distributed manner. SOA is based on three principles:

- **Modularity** Breaking tasks and services into smaller tasks or services.
- **Encapsulation** Clearly defined interfaces that insulate a service's internal workings from outside contact.

- **Reuse** Re-assembling services into composite applications that support new business processes.

SOA inherently encompasses a heterogeneous collection of platforms and sources. In particular, enterprise applications may be hosted on Java, and .NET platforms, as well as third-party packaged systems and legacy applications. SOA infrastructure spans these varied application architectures and provide mediation between them.

**SOAP (Simple Object Access Protocol)**
A protocol for exchanging XML-based messages over a computer network, normally using HTTP. SOAP also defines a way to perform remote procedure calls (RPCs) using HTTP as the underlying communication protocol. SOAP forms the foundation layer of the web services stack. See also WSDL (Web Services Description Language).

T

**TIBCO ActiveMatrix Administrator**
Product component for administering TIBCO ActiveMatrix infrastructure and services.

**TIBCO ActiveMatrix Policy Manager**
Product that supports the definition and enforcement of user-defined policies that modify the runtime behavior of services.

**TIBCO ActiveMatrix Registry**
Product that implements a UDDI registry for publishing and discovering services. A registry enables service providers to publish service listings and consumers to discover providers.
TIBCO Business Studio
Product component for developing TIBCO ActiveMatrix SOA projects.

TIBCO Enterprise Message Service
Product that implements JMS. ActiveMatrix Messaging Bus uses Enterprise Message Service as the messaging backbone.

TIBCO Management Daemon
Product that manages the resources of a TIBCO home directory.

TIBCO Rendezvous
Product that enables distributed messaging.

topic
Represents a publish-subscribe messaging channel between service consumers and providers. Topics can have multiple consumers and can consume multiple providers. Topic attributes include:
- Name
- Port type provided by the topic
- Links to components or references that export the provided port type
See also publish-subscribe messaging.

transport bridging
Enabling communication between a service consumer and service provider that use different transport protocols.

U
UDDI (Universal Description, Discovery and Integration)
A platform-independent, XML-based framework for describing services, discovering businesses, and integrating business services using the Internet, as well as a registry. See www.uddi.org.

W
web service
A software system designed to support interoperable machine-to-machine interaction over a network. It has an interface that is described in a machine-processable format such as WSDL. Other systems interact with the web service in a manner prescribed by its interface using messages, which may be enclosed in a SOAP envelope.

wire
Represents a connection between composite elements.

WSDL (Web Services Description Language)
An XML-based language for describing web services. The services are described in WSDL documents or files.

A client program connecting to a web service can read the WSDL document to determine what operations are available on the server. Data types referenced in the document are embedded in the WSDL file in the form of XML Schema. The client uses SOAP to invoke the operations listed in the WSDL.

WSDL file
An XML document that describes the services offered by a service provider. A WSDL file can take two forms: abstract and concrete. An
abstract WSDL file defines operations, which define specific message request and response formats, and data types, which describe the types of the objects passed in the messages. A concrete WSDL file contains the abstract WSDL plus the communication protocols and data formats by which the operations defined in the abstract WSDL can be invoked. A binding connects a port type or interface (a collection of operations) to a protocol and data format. The combination of a binding and a network address is a port (WSDL 1.1) or an endpoint (WSDL 2.0). A service is a collection of ports (WSDL 1.1) or endpoints (WSDL 2.0).

X

XML (Extensible Markup Language)
A text-based markup language in which tags (markup) identify the content, data, and text in the documents. Although tags can be defined as needed in the generation of an XML document, a document type definition (DTD) or XML Schema is usually used to define the elements allowed in a particular type of document. An XML document can be compared by using the rules in the DTD or XML Schema to determine its validity and to locate particular elements in the document.

XML Schema
An XML language for describing the structure of XML documents. The suffix of an XSD document is .xsd.
Index

A
ActiveMatrix Administrator cluster 45
ActiveMatrix Administrator server 45
ActiveMatrix Database 45
ActiveMatrix resource wizard 35
administration architecture and tools 43, 44
administration tools 42, 44
at least once 27, 27
at most once 27, 27
authentication realm 46

B
best effort 27, 27

C
composite element editors 37
containers 16

D
deployment 15
design 10
development tools 33, 34, 34, 44

E
enterprise and environment administration 46
environment administration 46

F
fault tolerance 30, 30

H
highly available services 47

I
implementation 11

L
load balanced services 47

M
mediation flow editor 38
MEP support 26, 26
message exchange patterns 25, 25
messaging 23, 23
Messaging Bus 16
messaging quality of service options 27, 27
messaging server 30, 30
messaging styles 26, 26
Microsoft Visual Studio 34
N

nodes 18

O

optional servers 31, 31
overview 24, 50

P

packaging 15
point-to-point 26
publish-subscribe 26

S

secure communications 31
servers 18, 29, 29
service administration 19, 47, 47
service and binding definition editor 37
service assembly deployment 47
service assembly editor 42, 42
service deployment 47
service description 5
service development process 9
service life cycle 7
service monitoring and management 48
service runtime 15
service-oriented architecture 2
shared resources 20

T

TIBCO ActiveMatrix Administrator 44
TIBCO ActiveMatrix Administrator architecture 44, 44

TIBCO ActiveMatrix Concepts

TIBCO ActiveMatrix BusinessWorks 9
TIBCO ActiveMatrix Policy Manager 9
TIBCO ActiveMatrix Policy Manager 31, 31, 31
TIBCO ActiveMatrix Registry 9, 31, 31, 31
TIBCO ActiveMatrix Service Bus 8
TIBCO ActiveMatrix Service Grid 9
TIBCO ActiveMatrix services 5
TIBCO Business Studio 34
TIBCO Business Studio Overview 34
TIBCO Enterprise Message Service 30
TIBCO Resource Wizard 35
TIBCO_HOME xii

W

WSDL 1.1 and 2.0 Terminology 50
WSDL 1.1 and 2.0 terminology 50