TIBCO iProcess® Conductor

Implementation

Software Release 11.2.1
August 2012

two-second advantage™

TIBCO®
The Power of Now®

Important Information

Contents

Preface vii

Changes from the Previous Release of This Guide. viii

Related Documentation ix

TIBCO iProcess Conductor Documentation ix

Other TIBCO Product Documentation ix

Third-party Documentation x

Typographical Conventions xi

Connecting with TIBCO Resources xiv

How to Join TIBCOmmunity xiv

How to Access TIBCO Documentation. xiv

How to Contact TIBCO Support xiv

Part I - Implementing iProcess Conductor 1

Chapter 1 About Order Fulfillment 3

Overview 4

Planning. 5

Design 6

Implementation 7

Chapter 2 Planning Order Fulfillment 9

TIBCO iProcess Conductor Approach 10

Identifying the Goals 11

Identifying the Tasks 12

Identifying Process Components 13

The Conceptual Hierarchy Approach 13

The Item/Location Approach 17

Summary 19

Chapter 3 Designing Process Components and Execution Plans 21

Designing Process Components 22

Designing Execution Plans 25

Summary Groups 29

Classifying Execution Plan Templates . . . 30
Evaluating the Design . . . 31
Summary. . . . 32

Chapter 4 Defining iProcess Procedures. . . . 33

Overview . . . 34
Defining the Fulfillment Procedure . . . 35
Decide Upon a General Design . . . 35
Mandatory Fields . . . 35
Mandatory Steps . . . 36
Field Inputs/Outputs. . . . 37
Defining Order Feasibility. . . . 39
Defining Execution Plan Development. . . . 39
Handling Manual Execution Plan Development . . . 41
Defining Execution Plan Orchestration . . . 42
Defining Procedures for Process Components . . . 44
Mandatory Fields . . . 44
Additional Fields . . . 45
Mandatory Steps . . . 45
Field Inputs/Outputs. . . . 46
Defining Error Handling Procedures . . . 51
System-Level Error Handling . . . 51
Process Component-Level Error Handling. . . . 52
Summary. . . . 53

Chapter 5 Defining Process Components and Execution Plans . . . 55

About Defining Process Components . . . 56
Using Process Component Versions . . . 56
About Defining Execution Plans . . . 60
Summary. . . . 61

Chapter 6 Defining the Rules . . . 63

Overview . . . 64
Using iProcess Decisions . . . 64
Using Statically Defined Rules. . . . 64
Creating the Rules Vocabulary . . . 65
iProcess Conductor Order Management Vocabulary. . . . 65
iProcess Conductor Vocabulary. . . . 67
Properties Required by iProcess . . . 68
Creating Rules for Fulfillment Procedure Selection . . . 69
Creating Rules for Template Selection . . . 70

Making the Rules Available 71
Configuring iProcess Conductor to Use Statically Defined Rules 72
Amending the Application.xml file. 72
Amending the AppConfig.xml file 72

Part II - EAI Steps 75

Chapter 7 Using EAI Orchestrator Steps 77
Why Use EAI Orchestrator Steps? 78
Using Messages in EAI Orchestrator Steps. 79
Message Invocation 79
Transaction Scope of EAI Orchestration Steps 80
Understanding the EAI Orchestrator Step Process 81
Creating an EAI Orchestrator Step 82
Jeopardy Management 86
Summary 88

Chapter 8 Using the EAI Order Step. 89
Why Use an EAI Order Step? 90
Transaction Scope of EAI Order Steps 91
Creating an EAI Order Step 92
Summary 102

Chapter 9 Using the EAI Transform Step 103
Why Use an EAI Transform Step? 104
Understanding the EAI Transform Step Process 105
Creating an EAI Transform Step 106

Appendix A Using iProcess Conductor with TIBCO EMS 115
Introduction 116
Using iProcess Conductor in a Cluster with EMS 117
Using iProcess Conductor under JBoss 4.2.1 with EMS 118
TIBCO EMS Configuration 118
iProcess Conductor Configuration 123
EAI Orchestrator Configuration. 124
Restart 124
Using iProcess Conductor under JBoss EAP 4.3 with EMS 125
TIBCO EMS Configuration 125
iProcess Conductor Configuration 130

EAI Orchestrator Configuration 131
Restart 131
Using iProcess Conductor under BEA WebLogic with EMS 132
TIBCO EMS Configuration. 132
iProcess Conductor Configuration 132
EAI Orchestrator Configuration 133
BEA WebLogic Configuration 134
Securing the iProcess Conductor JMS Gateway with EMS 146
Additional Customization 148
Changing Passwords 149

Appendix B Unsupported XPath Functions in XML Transform Plug-in 151

Unsupported XPath Functions 152

Glossary 155

Index 157

Preface

The objective of this guide is to describe how to design, plan and implement processes for the fulfillment of orders in TIBCO iProcess Conductor. This includes such tasks as defining iProcess procedures, process components and execution plans. This information is contained in Part I of this guide.

Part II contains detailed information about EAI Orchestration, EAI Order and EAI Transform steps.

You must have a thorough understanding of the TIBCO iProcess Conductor concepts, as discussed in *TIBCO iProcess Conductor Concepts*, before you carry out the tasks described in this guide.

Topics

- Changes from the Previous Release of This Guide, page viii
- Related Documentation, page ix
- Typographical Conventions, page xi
- Connecting with TIBCO Resources, page xiv

Changes from the Previous Release of This Guide

This section itemizes the major changes from the previous release of this guide.

There are no changes from the previous release of this guide.

Related Documentation

This section lists documentation resources you may find useful.

TIBCO iProcess Conductor Documentation

The following documents form the TIBCO iProcess Conductor documentation set:

- *TIBCO iProcess Conductor Concepts* Read this manual to gain an understanding of the product that you can apply to the various tasks you may undertake.
- *TIBCO iProcess Conductor Installation* Read this manual for instructions on site preparation and installation.
- *TIBCO iProcess Conductor Implementation* Read this guide for instructions on how to design, plan, and implement the fulfillment of orders.
- *TIBCO iProcess Conductor User's Guide* Read this guide for instructions on using the TIBCO iProcess Conductor user interface to orchestrate execution plans, create process components, amend orders, and so on.
- *TIBCO iProcess Conductor Administrator's Guide* Read this guide for instructions on common administrative tasks, such as archiving completed execution plans, managing users, and deploying the TIBCO iProcess Decisions rule sets.
- *TIBCO iProcess Conductor Release Notes* Read the release notes for a list of new and changed features. This document also contains lists of known issues and closed issues for this release.
- *TIBCO iProcess Conductor Utility Framework User's Guide* Read this guide for details of the Utility Framework and the applications used to export and import plans and process components.
- *TIBCO iProcess Conductor AOPD Message-Driven API Developer's Guide* Read this guide for details of the TIBCO iProcess Conductor's execution plan interfaces, and the facilities for automatic execution plan development.

Other TIBCO Product Documentation

You may find it useful to refer to the documentation set for TIBCO iProcess® Suite, which is used with TIBCO iProcess Conductor.

Third-party Documentation

You may find it useful to read the documentation for the following third-party products:

- Oracle® Database
- Oracle® WebLogic Server
- JBoss® Application Server

Typographical Conventions

The following typographical conventions are used in this manual.

Table 1 General Typographical Conventions

Convention	Use
iProcessConductor Dir	The directory where you install the TIBCO iProcess Conductor. For example: • on Windows systems, `c:\Program Files\iProcessConductor.` • on UNIX systems, `/opt/comDomain/iProcessConductor.`
iProcessConductor Domain	The target directory where you install the TIBCO iProcess Conductor domain. For example: • on Windows systems, *BEA_HOME*`\user_projects\domains\iPCDomain;` or *JBOSS_HOME*`\server\`*server_type*`\deploy` under JBoss. • on UNIX systems, *BEA_HOME*`/user_projects/domains/iPCDomain`. • or *JBOSS_HOME*`/server/`*server_type*`/deploy` under JBoss`.`
ORACLE_HOME	The pathname to your Oracle home. See your database administrator for details. If you are installing iProcess Conductor to work with a local database, this pathname must point to the Oracle database server. If you are installing iProcess Conductor to work with a remote database, this pathname must point to an Oracle client.
JAVA_HOME	The pathname to the directory where you install Java. For example: • on Windows systems, `C:\Program Files\Java\jdk1.6.0_21`. • on UNIX systems, `/opt/Java/jdk1.6.0_21`. **Note:** When you install the product on AIX platforms and need to communicate with iProcess Engine: — set *JAVA_HOME* as the pathname to the directory of Java incorporated by iProcess Engine 11.3.1 if you want to communicate with iProcess Engine 11.3.1 — install JRE 1.6.0 IBM J9 2.4 AIX SR6, and set *JAVA_HOME* as the pathname to the installation directory of JRE 1.6.0 IBM J9 2.4 AIX SR6 if you want to communicate with iProcess Engine versions earlier than 11.3.1
If you are using JBoss:	
JBOSS_HOME	The pathname to the home directory of your JBoss installation. For example: • on Windows systems, `c:\jboss\jboss-4.2.1`. • on UNIX systems, `/opt/JBoss/JBoss-4.2.1`.

Table 1 General Typographical Conventions (Cont'd)

Convention	Use
If you are using Oracle WebLogic:	
BEA_HOME	The pathname to the home directory of your Oracle WebLogic installation. For example: • on Windows systems, `c:\oraclewl\`. • on UNIX systems, `/opt/oraclewl/`. **Note:** Oracle WebLogic was previously known as BEA WebLogic. The existing variable name has been retained in the iProcess Conductor documentation.
WL_HOME	The directory where you install Oracle WebLogic. For example: • on Windows systems, `c:\bea\wlserver_10.3`. • on UNIX systems, `/opt/bea/wlserver_10.3`.
Other conventions:	
`code font`	Code font identifies commands, code examples, filenames, pathnames, and output displayed in a command window. For example: Use `MyCommand` to start the foo process.
`bold code font`	Bold code font is used in the following ways: In procedures, to indicate what a user types. For example: Type **`admin`**. In large code samples, to indicate the parts of the sample that are of particular interest. 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.
📝	The note icon indicates information that is of special interest or importance, for example, an additional action required only in certain circumstances.

Table 1 General Typographical Conventions (Cont'd)

Convention	Use
💡	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.
⚠	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

Convention	Use
`[ ]`	An optional item in a command or code syntax. For example: MyCommand [optional_parameter] required_parameter
|	A logical `OR` that separates multiple items of which only one may be chosen. For example, you can select only one of the following parameters: MyCommand para1 | param2 | param3
`{ }`	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} | {param3 param4} 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 | param2} {param3 | param4} 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 | param4}

Connecting with TIBCO Resources

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.

How to Access TIBCO Documentation

You can access TIBCO documentation here:

http://docs.tibco.com

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.

Part I - Implementing iProcess Conductor

This part contains information about designing, planning and implementing the necessary elements (such as process components, execution plans, iProcess procedures and so on) to implement processes for the fulfillment of orders in TIBCO iProcess Conductor. See Part II - EAI Steps for detailed information on related EAI steps.

Chapter 1 About Order Fulfillment

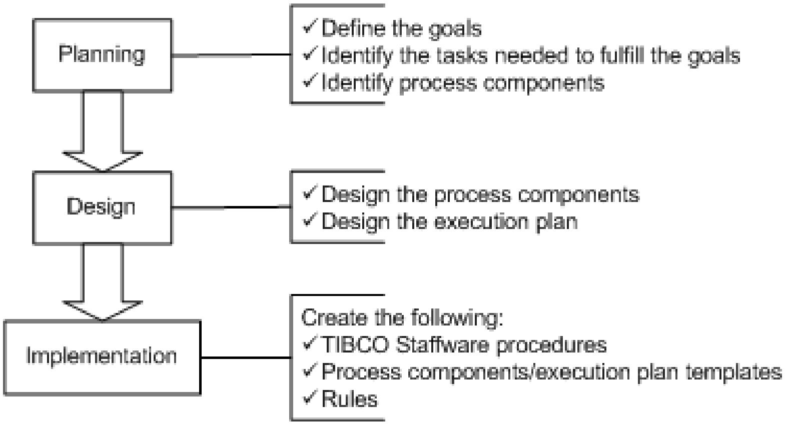
This chapter is an overview of the steps required to implement processes for the fulfillment of orders in TIBCO iProcess Conductor.

Topics

- Overview, page 4
- Planning, page 5
- Design, page 6
- Implementation, page 7

Overview

The following diagram shows the steps required to fulfill customer orders.


Planning
✓Define the goals
✓Identify the tasks needed to fulfill the goals
✓Identify process components
Design
✓Design the process components
✓Design the execution plan
Implementation
Create the following:
✓TIBCO Staffware procedures
✓Process components/execution plan templates
✓Rules

This section briefly describes these steps, each of which is discussed in detail in the subsequent chapters.

Planning

When planning the fulfillment of a customer order, you must:

- define the high-level goals.
- identify the tasks needed to achieve the goals.
- identify the necessary process components to carry out the tasks.

Design

Once you have finished the planning stage, you can:

- design the process components, including the process flows, process signatures rollback behavior and jeopardy conditions.
- design the execution plan, including any dependencies and plan-level jeopardy conditions.

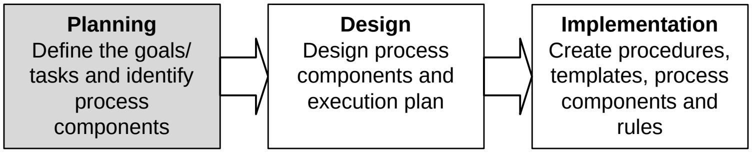
Implementation

After completing the design stage, you can begin to implement the elements that are necessary for order fulfillment, including:

- iProcess procedures
- Process components
- Execution plan template
- Rules. Rules are normally implemented by iProcess Decisions Plug-in, but you can configure iProcess Conductor to work without this plug-in

Chapter 2 Planning Order Fulfillment

This chapter is an overview of the steps required to plan the fulfillment of a submitted order.


Planning
Define the goals/ tasks and identify process components
Design
Design process components and execution plan
Implementation
Create procedures, templates, process components and rules

Topics

- TIBCO iProcess Conductor Approach, page 10
- Identifying the Goals, page 11
- Identifying the Tasks, page 12
- Identifying Process Components, page 13

TIBCO iProcess Conductor Approach

When designing end-to-end processes in TIBCO iProcess Conductor, it is easier to think in terms of a top-down approach rather than a start to finish approach. This means thinking about order fulfillment in terms of the goals you want to achieve and working back from the goal to your current position.

The more usual approach to designing processes in iProcess Suite is the start to finish approach (starting from the current position and working towards a goal). This does not work well when planning large, complex processes for use in fast-moving modern industries because of the changing, complex nature of customer requirements.

Identifying the Goals

To fulfill an order, you must first state the high-level goals or legal combinations of goals in simple terms. For example, fulfilling a customer order to a telecommunications provider might require you to:

- Provide a home network product
- Provide broadband routing and switching service
- Roll out a wireless network product.

When identifying the goals, you must consider that the goal is likely to change part way through the fulfillment process. For example, a customer could order more network capacity or a different product.

Identifying the Tasks

To achieve the stated goals of order fulfillment, there are a number of “chunks” of functionality or tasks that are required.

At this stage in the process, it is necessary only to identify the tasks. Later you will translate these tasks into process components and consider their dependencies.

Example

Suppose a telecommunications provider has two new products whose goals are to:

- deliver home broadband.
- deliver a home network.

To achieve these goals, the following tasks must be completed:

- Supply and deliver a wireless router
- Provide an exchange connection
- Provide ADSL service
- Provide a local loop
- Supply and deliver an ADSL modem.

Identifying Process Components

Process components represent reusable building blocks that encapsulate the management of a particular item. The process components form a reusable library that you can call upon to fulfill different types of orders.

The previous sections discussed goals and the tasks required to support those goals. While you could simply translate these goals into process components, this would not be efficient because it ignores the principles of reuse and optimization.

When identifying the process components you require, it is important to analyze the entire business. There are two main approaches to identifying process components:

- Thinking in terms of concepts such as products, services and resources (by considering the specifications for each).
- Considering physical items or locations on which actions are performed.

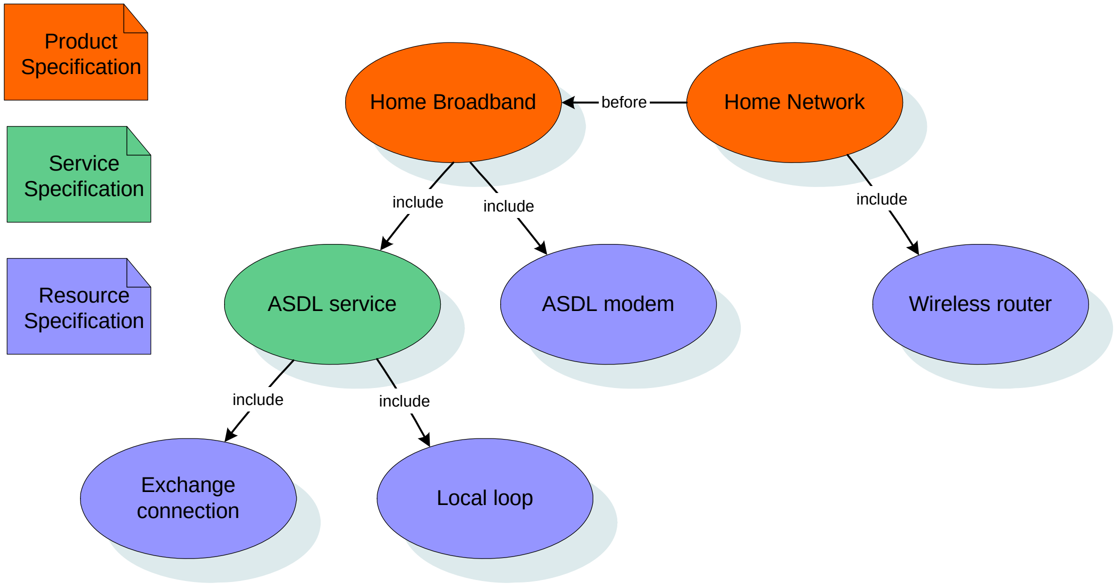
It is possible that when you identify your process components, you will use a combination of these approaches. The following sections discuss each approach in greater detail.

The Conceptual Hierarchy Approach

Using this approach, the process components are derived from the specifications for the intangible concepts that are required to fulfill the goals of the order. For example, in the telecommunications industry these might be products, services and resources; in the insurance industry they might be policies, claims and excess.

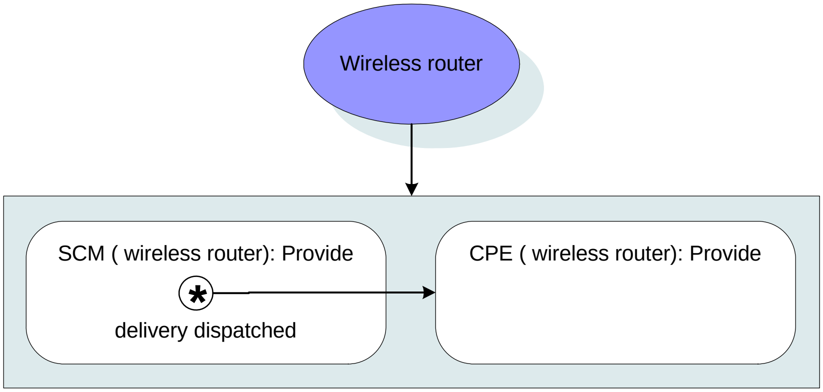
Example

This is an example from the telecommunications industry. The diagram is a simplified view of the product, service, resource specification domain as described in the Shared Information/Data (SID) model (document GB922, Addendum 2), which can be found on the TeleManagement Forum web site (www.tmforum.org).



The goals are to provide Home Broadband and Home Network products. This diagram shows the products and their breakdown into the required service and resource specifications.

Each of the items in the previous diagram are candidates for process components. However, some items can be optimized to include one or more generic process components. For example:


Wireless router
SCM (wireless router): Provide
delivery dispatched
CPE (wireless router): Provide

The wireless router item is accomplished by using two generic process components:

- Supply Change Management (SCM) to deliver a wireless router
- Customer Premises Equipment (CPE) to configure the wireless router.

While it is perfectly acceptable to have a single process component called `wireless router` which handles both the delivery and configuration of the wireless router, using two process components exposes these processes so that they can be used in other contexts as shown in the following section.

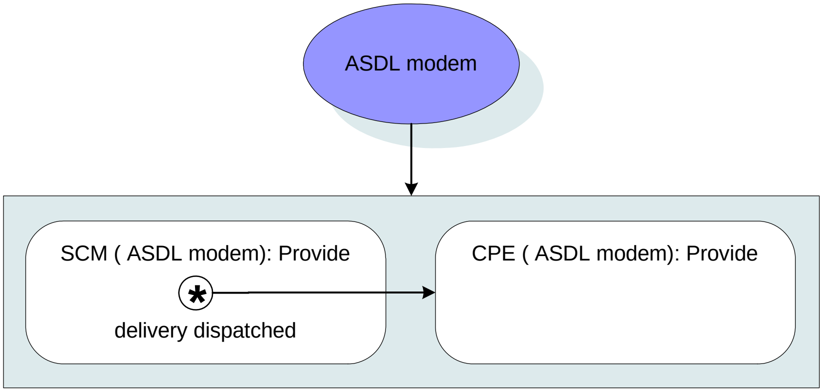
In these process components, Provide is the task action. For more information, see About Task Actions on page 22.

Reusing Process Components

When designing process components, you should look for common processes or sub-processes whose process components can be reused by parameterization. This means that the two different activities use the same process component, but they behave differently because they are passed different initial parameters.

Example

The previous example showed how the wireless router item was optimized to use two generic parameterized process components: SCM and CPE. Could these process components be reused to provide other hardware? The following example shows how these same process components are used to provide an ADSL modem:



Breaking down process components into further process components is necessary when processes start to be replicated between process components. In this case, it may be desirable to split the repeating process steps out into new process components. However, do not over-optimize and make process components overly specialized, because this reduces the opportunities for their reuse.

Summary

Having examined each part of the product/service/resource hierarchy diagram, we can now list the process components that are required:

Diagram item	Process component(s)
ADSL modem	• SCM (ADSL modem): provide • CPE (ADSL modem): provide
ADSL service	ADSL service: provide
Exchange connection	Exchange connection: provide
Local loop	Local loop: provide

Diagram item	Process component(s)
Wireless router	• SCM (wireless router): provide • CPE (wireless router): provide

Depending on what is uncovered after testing, the process component design may need to be revised, but this is an expected part of the iterative process for identifying process components.

The Item/Location Approach

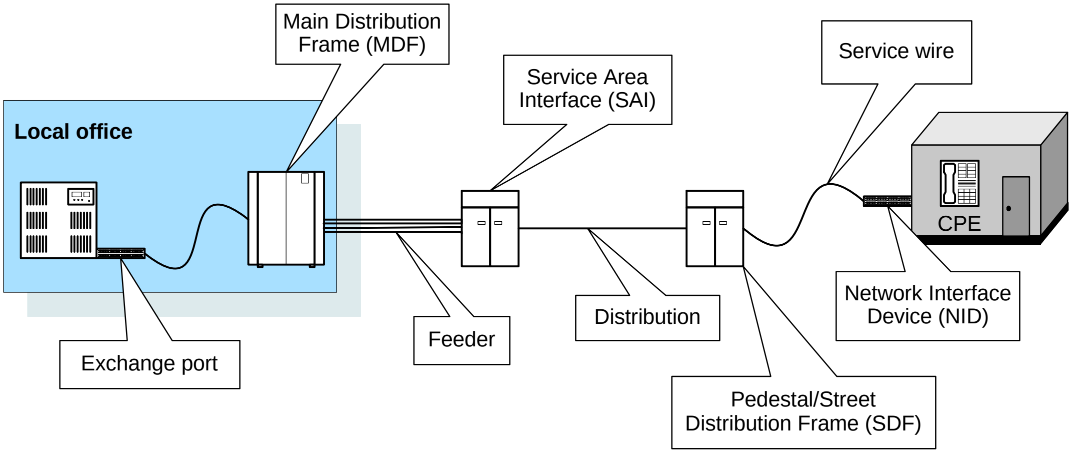
This approach considers the physical items or locations on which actions are performed, and consists of the following steps:

1. Identify the items involved (in the telecommunications example, these could include for instance a service wire or feeder).
2. Determine which items have to be managed to deliver the goal.
3. Optimize related items.
4. Translate remaining items into process components.

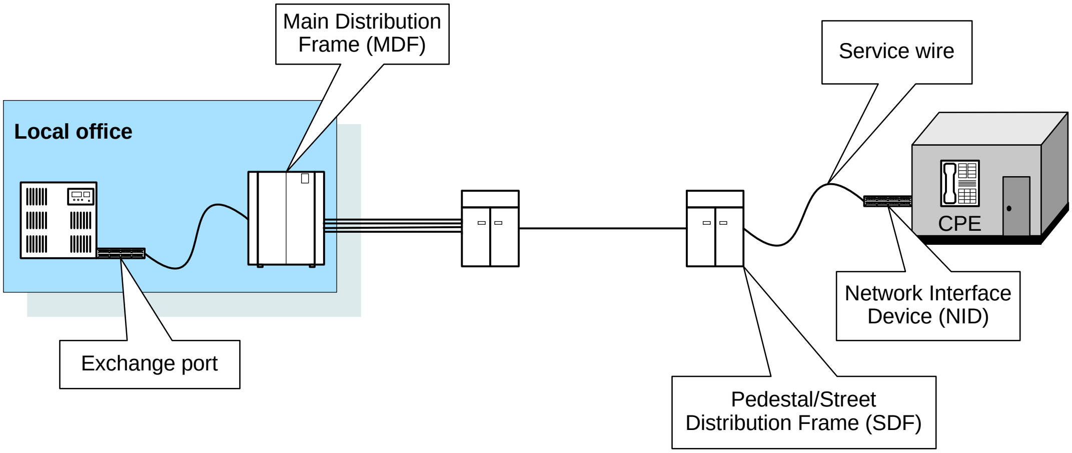
Example

The following example illustrates the Item/Location approach by showing a simplified view of the provision of Plain Old Telephone Service (POTS).

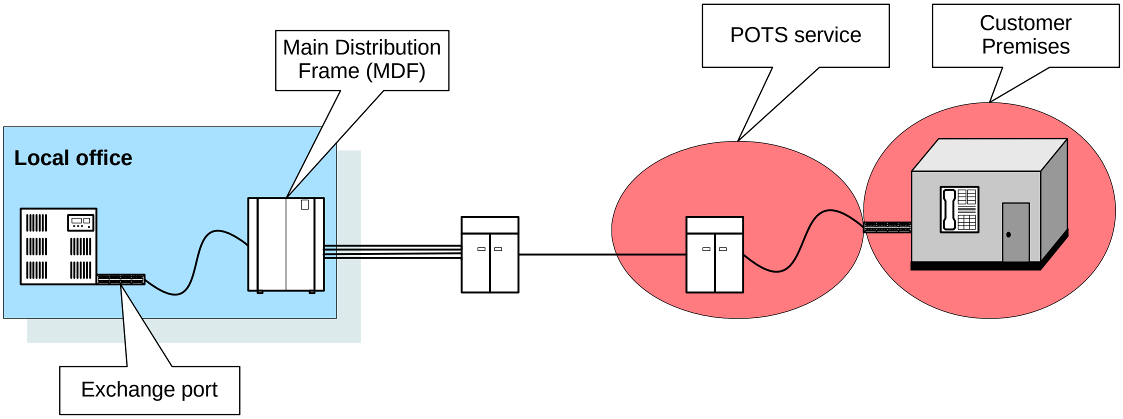
As mentioned, the first step is to identify the physical items/locations involved:


Main Distribution Frame (MDF)
Service wire
Service Area Interface (SAI)
Local office
CPE
Network Interface Device (NID)
Distribution
Feeder
Exchange port
Pedestal/Street Distribution Frame (SDF)

The items that do not need to be managed to deliver POTS (because they are already present or require no change) can be removed from consideration:


Main Distribution Frame (MDF)
Service wire
Local office
CPE
Network Interface Device (NID)
Exchange port
Pedestal/Street Distribution Frame (SDF)

Some of the remaining items can be optimized. For example, configuration of the NID and the CPE both occur on the customer premises, so these items can be managed as a single entity. Similarly, the configuration of the service wire and the SDF can be optimized:



All of the items left in the previous diagram are candidates for conversion into process components.

Summary

In this chapter we learned how to identify the goals and required tasks for order fulfillment.

Then we examined two possible approaches to identifying process components. The approach that you use for identifying your process components depends on your business, but in general:

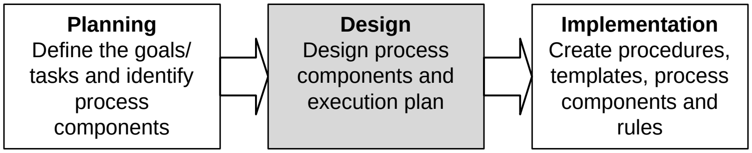
- the conceptual hierarchy approach works best for abstract or bundled goals.
- the item/location approach works best when tangible items are being managed to accomplish the goal.

Both approaches can yield similar results, and it is possible that you will use a combination of both. The process components should be tested thoroughly to validate (and if necessary revise) the original design.

The next chapter discusses how to design process components and execution plans.

Chapter 3 Designing Process Components and Execution Plans

This chapter describes how to design process components.


Planning
Define the goals/ tasks and identify process components
Design
Design process components and execution plan
Implementation
Create procedures, templates, process components and rules

Topics

- Designing Process Components, page 22
- Designing Execution Plans, page 25

Designing Process Components

When designing a process component, you need to determine:

- the logical flow of the activities that must occur when you provide, cease or update the process component.
- what happens when a Provide, Cease or Update is cancelled (the rollback behavior).
- the inputs and outputs required by each activity in the process.
- the milestones of the process component.
- jeopardy conditions and jeopardy consequential actions.

About Task Actions

When your process components are associated with tasks in the execution plan, you must assign one of the following actions to each plan task:

- Provide
- Cease
- Update

When designing process components, you must plan what happens for each action. Also, suppose an order is cancelled while it is in the middle of providing a product. The design of your process component should specify the rollback behavior when this happens for each action.

Data Exchange Between Process Components

Fields are mapped between process components by matching the field names. For example, if one process component provides a field named IPADDR, any other process components that require the IP address must have the IPADDR field defined.

Designing the Milestones

Often milestones arise as you consider the inputs and outputs of each activity. For example, the output of the `exchange connection: provide` process component may be an exchange ID. It could be that another activity requires the exchange ID as input. Both of these activities would become milestones, and the exchange ID part of the process signature (see Designing the Process Signature on page 23).

Milestones may also arise from a significant activity in your process that you want to synchronize with other activities. For example, suppose in your process component there is an expensive activity that you would rather not have to cancel once it begins. You make this activity a milestone near the end of the process component and have it depend on the other milestones finishing.

A process component can have as many milestones as you want, although TIBCO recommends that you keep your milestones to a maximum of five. This is because it makes your process components easier to understand.

A milestone is an EAI Orchestration step in your iProcess procedure. See Using EAI Orchestrator Steps on page 77 for more information about EAI Orchestration steps.

Designing Jeopardy Conditions for a Process Component

Jeopardy management enables you to manage the risk associated with process components that fall behind schedule. You need to define how long each of your process components typically takes to complete, and also what is the maximum acceptable time that it can be allowed to take. You can then decide whether to monitor its duration against one or both of these figures. If the process component exceeds a monitored duration, a jeopardy condition is recognized, and you can specify what consequential action TIBCO iProcess Conductor should take in this case.

See *TIBCO iProcess Conductor Concepts* for an introduction to the concepts of jeopardy management

You can also specify jeopardy conditions and consequential actions at the level of the execution plan. You should consider both plan level and process component level jeopardy management together when deciding the jeopardy management strategy for your execution plan. See Designing Jeopardy Conditions for an Execution Plan on page 25.

Designing the Process Signature

Business data related to the order is stored in iProcess procedures, in the iProcess fields. It can be manipulated as the order progresses.

Each process component has a process signature. A process signature consists of EAI Orchestration steps and the data contained in the iProcess fields that are defined as input and outputs in the EAI Orchestration steps in the iProcess procedure.

For each process component that you design, you should list the inputs and outputs that the process component requires. Doing this helps you create an "execution plan vocabulary," which you can use to ensure that all the required inputs are supplied, either by other process components or as default values in the execution plan.

Designing Execution Plans

When designing an execution plan or execution plan template, you do the following:

- List the process components that are necessary to fulfill the goal.
- Define jeopardy conditions for the execution plan.
- Define the dependencies between the process components (optional).

When designing execution plans, it is best to use an iterative approach rather than trying to design a perfect plan at the first attempt. Simulation software can be useful in testing an execution plan design and cutting down the number of iterations required.

Designing Jeopardy Conditions for an Execution Plan

TIBCO iProcess Conductor can monitor the forecast end date and time of an execution plan and compare it against several threshold dates and times, to determine whether the plan is running on time or is taking longer to complete than it is predicted to. You need to decide which of these thresholds to monitor. If the plan exceeds, or is predicted to exceed, a monitored threshold, then a jeopardy condition is recognized, and you can specify what consequential action TIBCO iProcess Conductor should take in this case.

See *TIBCO iProcess Conductor Concepts* for an introduction to the concepts of jeopardy management

You can also specify jeopardy conditions and consequential actions for process components. You should consider both plan level and process component level jeopardy management together when deciding the jeopardy management strategy for your execution plan. See Designing Jeopardy Conditions for a Process Component on page 23.

About Dependencies

TIBCO iProcess Conductor prevents you from defining execution plans with dependency loops. TIBCO iProcess Conductor validates the execution plan when you change the status of the execution plan from Draft to Pending or Template.

For plan tasks with dependencies, you must ensure that any required input fields are supplied either by:

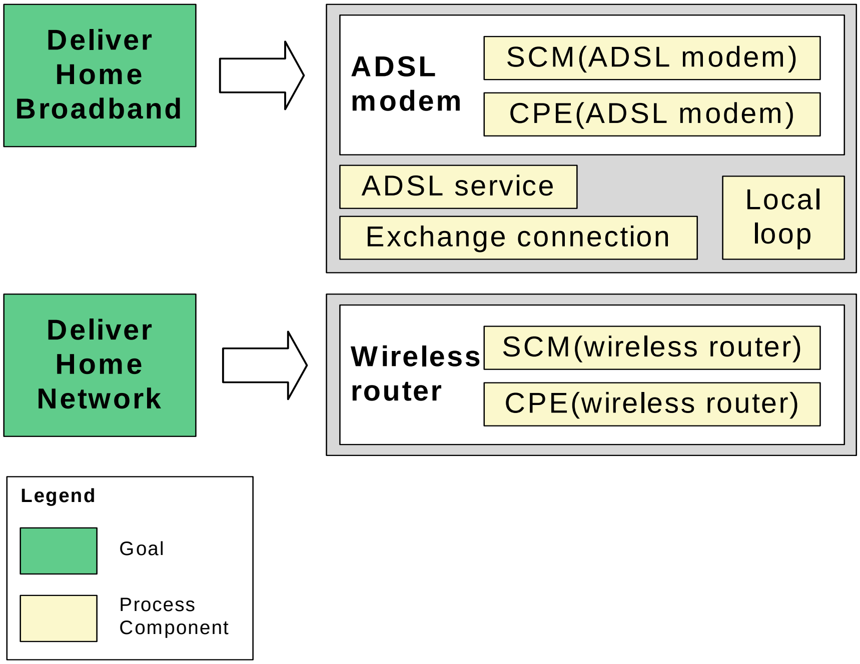
- one of the dependent plan tasks, or
- a hard-coded value in the execution plan itself.

Input data for a plan task can be supplied by a plan task that has already completed.

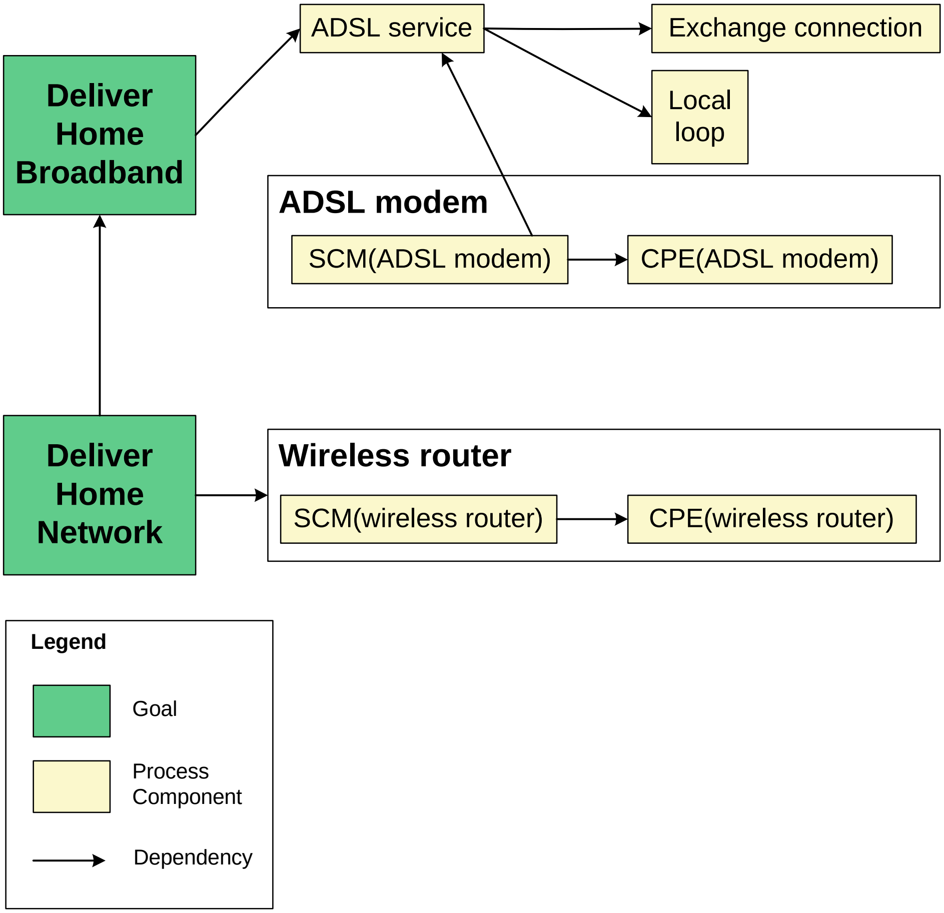
Example Execution Plan: Home Broadband and Networking

An example is to create an execution plan template for providing home broadband and network services in the telecommunications industry, drawing on the conceptual hierarchy example already discussed (see The Conceptual Hierarchy Approach on page 13).

You can collect together the set of required process components as follows:

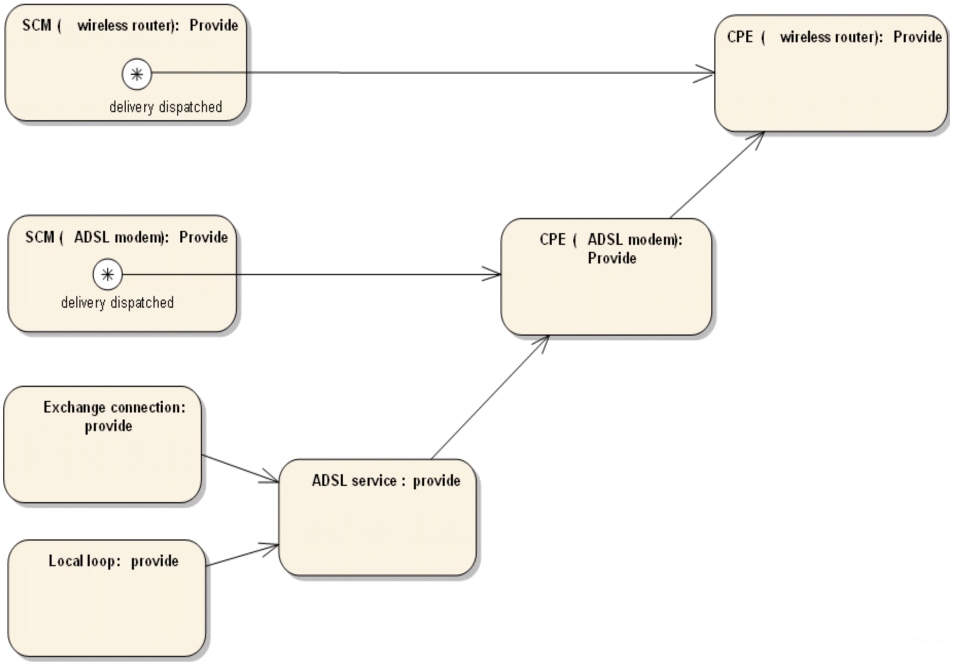


The previous diagram does not list the dependencies, so it is necessary to express them as follows:



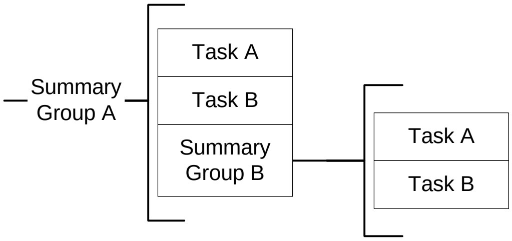
This means, for example, that `CPE(wireless router)` depends on `SCM(wireless router)`, which is necessary to deliver a home network.

A UML diagram of the completed execution plan is as follows:


SCM (wireless router): Provide
delivery dispatched
CPE (wireless router): Provide
SCM (ADSL modem): Provide
delivery dispatched
CPE (ADSL modem): Provide
Exchange connection: provide
ADSL service : provide
Local loop: provide

Summary Groups

For plan tasks that are logically related, you can create summary groups. Summary groups can be contained within other summary groups in a hierarchical structure as follows:



Once you have created summary groups, you can specify dependencies between summary groups and other plan tasks groups or individual plan tasks. For example, in the previous diagram, you could specify that a task is dependent on Summary Group A completing before it can start. This means that Task A, Task B, and the included tasks in Summary Group B must all complete before the dependency is satisfied.

Example

In our home broadband/network example, we could decide to structure the provision of each product as a separate summary group. You could then have a dependency that specifies that the home network summary group cannot begin until all the tasks in the home broadband summary group finish. For more information on dependency types, see *TIBCO iProcess Conductor Concepts*.

Classifying Execution Plan Templates

As discussed in *TIBCO iProcess Conductor Concepts*, classification is useful for grouping orders, execution plans and process components in ways that are meaningful to your business. In the case of execution plan templates, any classifications that you create can be inherited by execution plans instantiated from that template.

Example

Suppose that you have different execution plans based on geographic locations in the UK. You could create a classification group called UK REGIONS and add classification values such as NORTHWEST, SOUTHWEST, WALES, and so on. Then you could associate an object type (in this case execution plan) with the classification group UK REGIONS. For details about how to create and manage classification groups, see *TIBCO iProcess Conductor User's Guide*.

As a result, when you create an execution plan, you can select a classification value (for example, NORTHEAST). This selection can be made mandatory, and in the case of execution plan templates, execution plans instantiated from the template can inherit the classification.

There are two ways of ensuring an object is created with mandatory classification values:

- *For an execution plan* - You can specify classification data in an EAI Orch step with a message type of Start Plan (see Using EAI Orchestrator Steps on page 77).
- *For an order* - In TIBCO iProcess Decisions Plug-in, you can create rules that supply the values for order requests at run-time in the form of an XML payload (see Defining the Rules on page 63).

You can supply classification values in the user interface (see *TIBCO iProcess Conductor User's Guide*) or by using the TIBCO iProcess Conductor API.

Evaluating the Design

As you design the process components and execution plans, you should be able to identify any shortcomings. For example, any of the following can require a re-design:

- Insufficient process components to cover all eventualities
- Process signatures of process components in the end to end processes do not match
- Too many inputs to one process component.

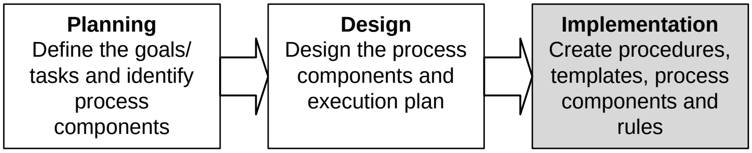
Ideal execution plans are readily comprehensible and have process components that are cohesive, with each having a manageable, logical number of inputs.

Summary

We have now identified and designed the process components and the execution plan, but not actually defined them in TIBCO iProcess Conductor. The next step is to define the relevant iProcess procedures as described in the following chapter.

Chapter 4 Defining iProcess Procedures

This chapter describes how to use iProcess procedures with TIBCO iProcess Conductor.



Planning
Define the goals/ tasks and identify process components

Design
Design the process components and execution plan

Implementation
Create procedures, templates, process components and rules

Topics

- Overview, page 34
- Defining the Fulfillment Procedure, page 35
- Defining Procedures for Process Components, page 44
- Defining Error Handling Procedures, page 51

Overview

Now that you have designed (but not actually defined) your process components and execution plan templates in TIBCO iProcess Conductor, your fulfillment process may involve either using iProcess procedures, or be achieved using JMS messaging. This chapter assumes that you are using iProcess, and tells you how to define the iProcess procedures that the process components and execution plans will use.

How you design and structure your procedures is specific to your business, but you must have the following:

- a fulfillment procedure. The fulfillment procedure manages the processing of an order including order feasibility, execution plan development and execution plan orchestration (all of these can be accomplished within the main fulfillment procedure or using sub procedures). It also instructs TIBCO iProcess Conductor which iProcess Suite processes to graft for the execution plan to orchestrate.
- process component procedures. For each individual task, for example, Exchange Connection, you need to define an iProcess procedure. Each of these individual iProcess procedures will be used by the corresponding process component that you will create in TIBCO iProcess Conductor.

Defining the Fulfillment Procedure

As described in *TIBCO iProcess Conductor Concepts*, the fulfillment procedure should handle order feasibility, execution plan development and execution plan orchestration. The goals are to:

- identify the execution plan template that the execution plan should be instantiated from, and
- to notify TIBCO iProcess Conductor which iProcess Suite processes to orchestrate to fulfil the order.

Decide Upon a General Design

Depending on your requirements, there are several options for defining how the elements of fulfillment should fit together. For example:

- You can define one overarching fulfillment procedure that calls each of the elements as a sub-procedure. This approach is used as the basis for the examples in this section.
- Alternatively, you can break down the elements into a combination of procedures. For example, you could define a procedure that deals with order request validation and that calls a sub-procedure that handles the execution plan development and execution plan orchestration.

Mandatory Fields

The following table describes the mandatory iProcess fields passed into the fulfillment procedure (and related sub-procedures):

TIBCO iProcess Field	Type	Length	Description
swo_amendindex	NUMERIC		Used to specify whether an order is a new request, or an amendment: • 0 - new order • 1 or more - amendment Supplied into the fulfillment process when the order is created.

TIBCO iProcess Field	Type	Length	Description
swo_orderref	TEXT	20	Unique identifier of the order. Provided by the user and supplied into the fulfillment process when the order is created.
swo_ordernumber	TEXT	20	The order number is generated when an order is received by TIBCO iProcess Conductor.

Mandatory Steps

The fulfillment procedure must contain at least the following steps:

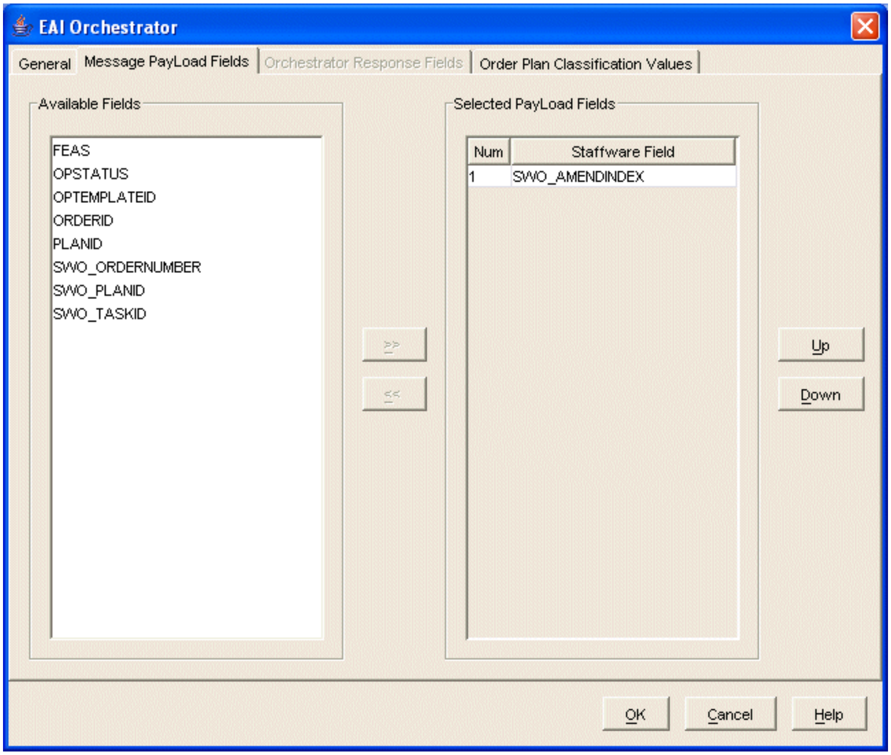
- Start Plan (EAI Orchestrator step, message type of Start Plan)
- Ad-hoc event step to handle amendments
- Task complete (EAI Orchestrator step)

About Ad-hoc Event Steps For Amendments

When an order amendment is received, depending upon whether the execution plan has started orchestrating, the existing fulfillment processes associated with the order may be required to withdraw all actions and terminate. This is handled by an ad-hoc event step. For a detailed examination of how TIBCO iProcess Conductor handles order amendments, see *TIBCO iProcess Conductor Concepts*.

Field Inputs/Outputs

The field input required by an execution plan is supplied via a message payload specified when you create the EAI Orchestrator step with a Start Plan step type. For example:



Available Fields:

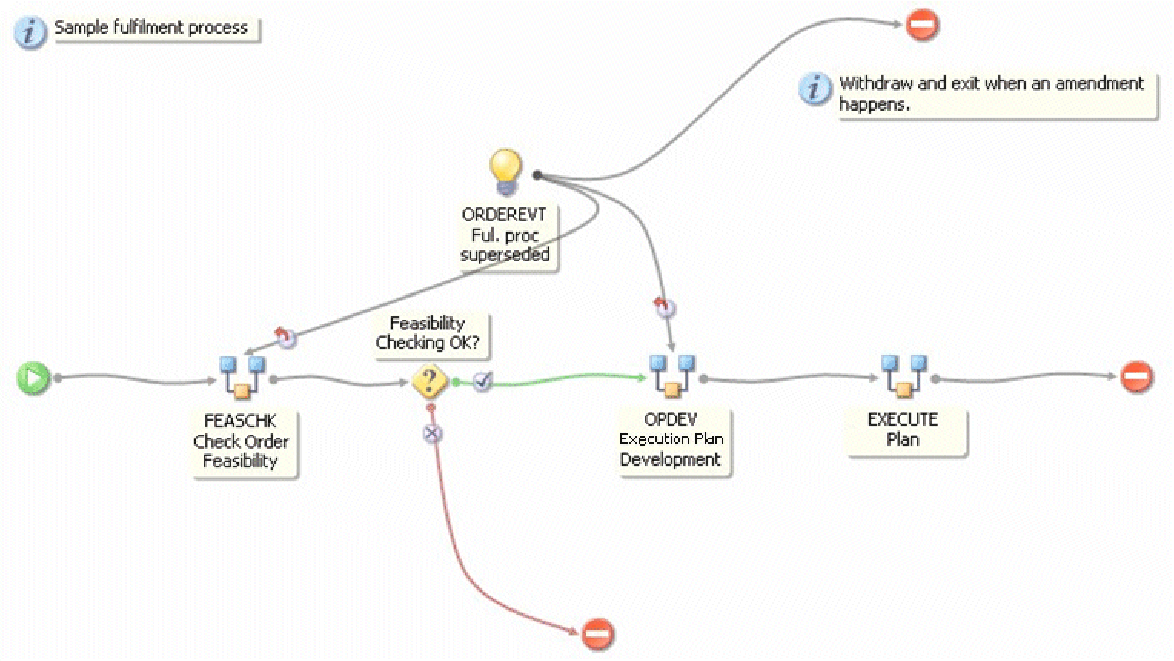
FEAS
OPSTATUS
OPTEMPLATEID
ORDERID
PLANID
SWO_ORDERNUMBER
SWO_PLANID
SWO_TASKID

Num	Staffware Field
1	SWO_AMENDINDEX

You also must ensure that the necessary inputs and outputs required by individual EAI Orchestration steps are passed as required.

Example

The following example shows an overarching fulfillment procedure:



The fulfillment procedure in this example does the following:

1. Calls the Feasibility sub-procedure.
2. Depending on the result of execution feasibility, the procedure either exits or continues with the Execution Plan Development sub-procedure.

   The ORDEREVT ad-hoc step handles withdrawing.
3. Calls a sub-procedure to orchestrate the execution plan.

   There is an event step that is used if the execution plan is superseded by an amendment coming in during either feasibility or execution plan development.

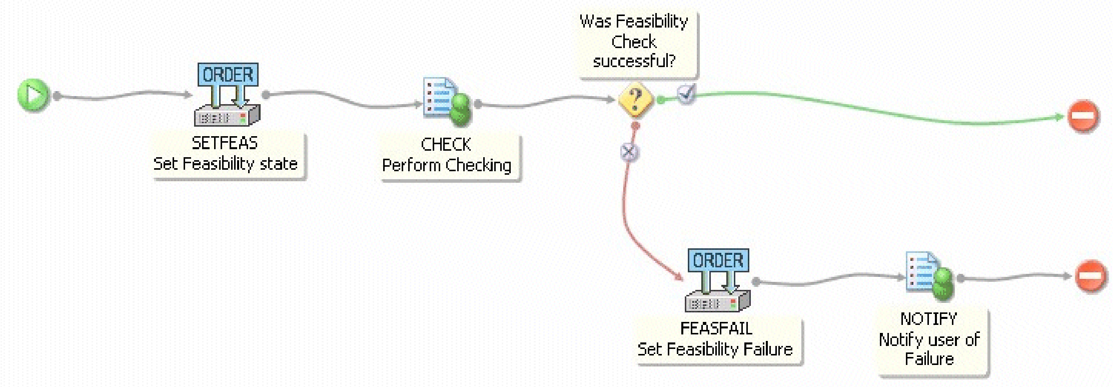
At any point in the fulfillment procedure, you can use EAI Order steps to change the event step that will be used if the fulfillment procedure is superseded. For more information, see Using the EAI Order Step on page 89.

Defining Order Feasibility

Order feasibility has the following objectives:

- To perform any order validation required by your business process (for example, order conditioning, credit or compatibility checks). It may be that your business process does not require any feasibility checking in which case you do not need to include it.
- To notify TIBCO iProcess Conductor:
  - — how to update the status of the order at each stage of the feasibility process.
  - — the name of the event step that is going to receive the order amendments or cancellations.

The following example shows a feasibility sub-procedure:



The feasibility sub-procedure does the following:

1. Sets the status of the order to Feasibility using an EAI Order step.
2. Performs the checks that are part of the order validation.
3. If the order did not pass the feasibility checking, set the order status to Feasibility Failed using an EAI Order step and notify the user.
4. Return to the main fulfillment procedure.

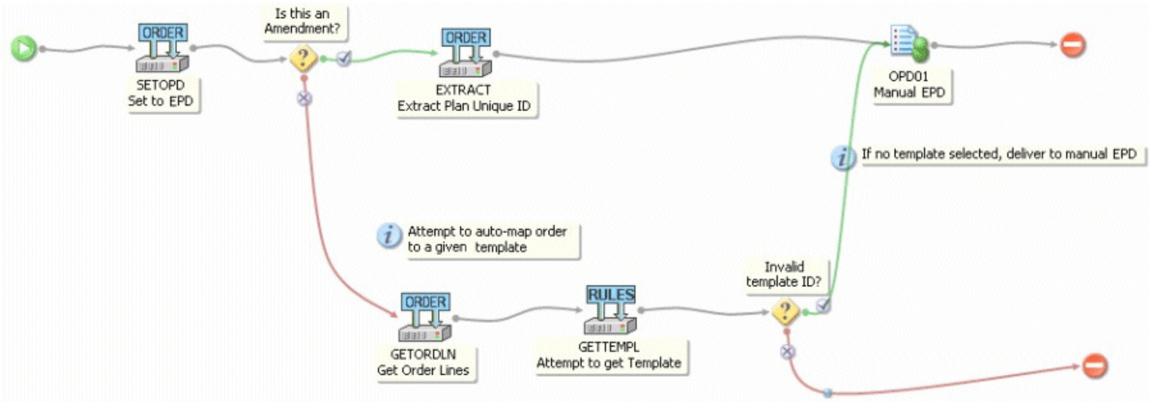
Defining Execution Plan Development

Execution plan development has the following objectives:

- Taking the order from the fulfillment element and passing it to TIBCO iProcess Decisions Plug-in.

- Enabling manual execution plan development if TIBCO iProcess Decisions Plug-in does not identify an execution plan template.

The following example shows an execution plan development sub-procedure:



In this example, the steps are processed as follows:

1. The status of the order is set to Execution Plan Development (EPD) using an EAI Order step.
2. The procedure determines if the order request is an amendment.
3. If the request is an amendment, the following happens:
   a. An EAI Order step gets the unique Execution Plan ID.
   b. The sub-procedure sends a work item to a user queue to initiate the creation of a manual execution plan.
4. If the request is not an amendment, the following happens:
   a. An EAI Order step fetches the order lines.
   b. An EAI Rules Manager step attempts to map the order to an execution plan template.
   c. If a template cannot be identified, the sub-procedure sends a work item to a user queue to initiate the creation of a manual execution plan.

Handling Manual Execution Plan Development

iProcess Conductor Order Management uses `openforms.xml` to handle manual orders and manual execution plan development. The `openforms.xml` functionality is provided by the iProcess Client (JSP) and is found at `/applications/COM-ORCH-2.0.ear/ORCH-web-2.0.war/openforms.xml`. See *TIBCO iProcess Client (JSP) Customization Guide* for information about `openforms.xml`.

When you open a work item in the iProcess Client (JSP), it is `openforms.xml` which determines how the work item is displayed. When you click on a work item in the iProcess Client (JSP) it will either take you to the default form or to the manual order or manual execution plan development forms.

In this example, `openforms.xml` determines which view should be used by the parameters specified in the `openforms.xml` file. Listed below is an example of an extract from the `openforms.xml` file:

```
<Procedure name="IT3%">
<DefaultHandler>forms.jsp</DefaultHandler>
<FormHandler>
<StepName>VIEWORD%</StepName>
<Handler>COM/SWWIRedirect.jsp</Handler>
</FormHandler>
</Procedure>
```

```
<Procedure name="IT3%">
<DefaultHandler>forms.jsp</DefaultHandler>
<FormHandler>
<StepName>OPD%</StepName>
<Handler>COM/SWWIRedirect.jsp</Handler>
</FormHandler>
</Procedure>
```

where:

- IT3% is the name of any fulfillment process beginning with IT3.
- VIEWORD% and OPD% are the names of any steps that begin with OPD or VIEWORD.

% is a wild card character.

This extract is configuring `openforms.xml` to display the default form unless it finds any step that begins with OPD or VIEWORD in a procedure beginning with IT3. If it finds a step beginning with OPD or VIEWORD, it should display the `SWWIRedirect.jsp` page. The `openforms.xml` then determines which view should be displayed based on the parameters that have been supplied as part of the iProcess fulfillment process.

⚠ You can amend the procedure name and step name values. However, the handler field should point to COM/SWWIRedirect.jsp as this is how iProcess Conductor Order Management knows how to display the work items based on the supplied parameters.

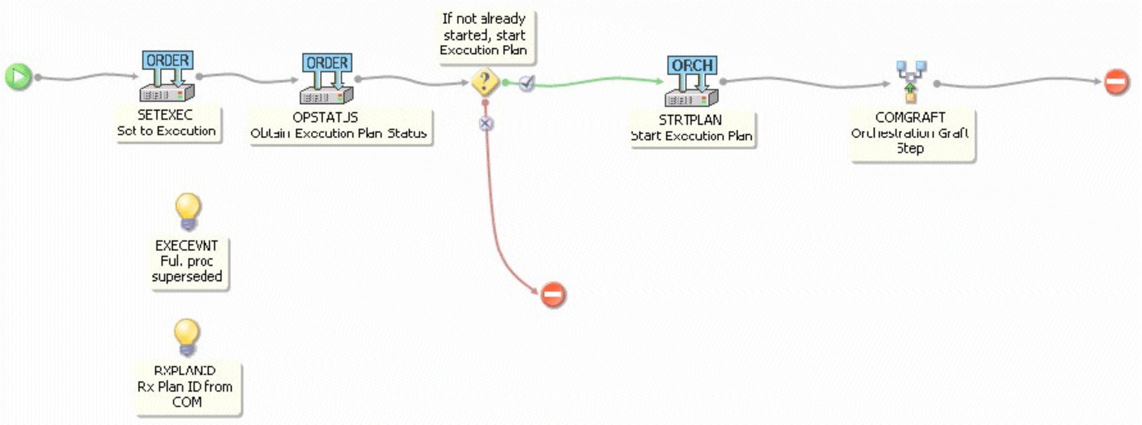
Parameters are passed to the iProcess Client (JSP) by setting field values in your iProcess Suite process, then including those fields as display fields on the iProcess form for the work item. When the user opens the OPD work item in the iProcess Client (JSP), iProcess Conductor Order Management redirects the user to manual OPD pages. Once the user has selected a plan, they release the work item from the execution plan page. This updates the SWO_OBJECT process field with the primary key of the selected plan or template. See Defining Order Feasibility on page 39 and Defining Execution Plan Development on page 39.

Defining Execution Plan Orchestration

During Execution Plan Orchestration:

- The specified plan ID is submitted for orchestration. If it is a template, the iProcess Conductor copies the template and uses it to orchestrate the plan.
- The processes from the iProcess procedures that make up the execution plan are started and orchestrated.

The following example shows an Execution Plan Orchestration sub-procedure:



In this example, the steps are processed as follows:

1. The first step is an EAI Order step that sets the status of the order to Execution.
2. Another EAI Order step gets the status of the execution plan.
3. A condition checks to see if the execution plan has already been started.
4. If the plan has not started:
    a. an EAI Orch step instructs TIBCO iProcess Conductor to start the execution plan that it has instantiated. The message type for the EAI Orch step should be Start Plan.
    b. an Orchestration graft step which references the sub-procedure template of iProcess Suite processes to graft to complete the execution plan.

Defining Procedures for Process Components

Procedures associated with process components have the following objectives:

- Passing data to and from TIBCO iProcess Conductor.
- Performing any tasks that are required as part of the activity that the process component represents.
- Handling cancel or suspend events.

The iProcess procedure must be a sub-procedure so that it can be grafted onto parent the fulfillment procedure.

Mandatory Fields

The following iProcess fields are passed into the procedures for process components:

TIBCO iProcess Field	Type	Length	Description
swo_action	TEXT	20	Provide, Cease or Update as defined in the plan task.
swo_exception	TEXT	20	The exception handler defined when you create the process component.
swo_planid	TEXT	20	This is the unique execution plan ID generated by the TIBCO iProcess Conductor when an execution plan is instantiated. The fieldname must be SWO_PLANID.
swo_taskid	TEXT	255	Unique ID of the plan task that is currently active. It is generated by the TIBCO iProcess Conductor.
swo_ordernumber	TEXT	20	The order number is generated when an order is received by the TIBCO iProcess Conductor.

Additional Fields

The following iProcess fields may also optionally be passed into the procedures for process components:

TIBCO iProcess Field	Type	Length	Description
swo_orchcmd	TEXT	20	Specifies whether the execution plan is to be cancelled. Values are CONTINUE in normal situation or CANCEL if the plan is cancelled. If your iProcess sub-procedure for the iProcess component does not define this field, iProcess Engine will generate a message in sw_warn when iProcess Conductor passes this field to iProcess Engine. The field is not mandatory because cancellation is usually handled using an Event Step.

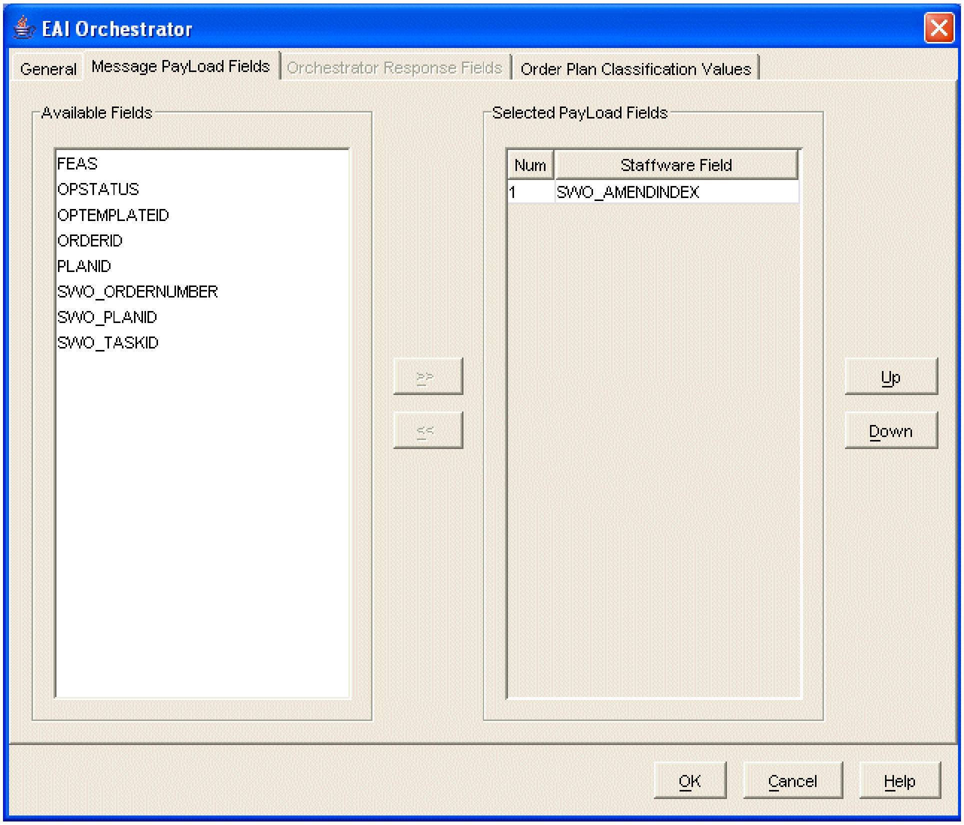
Mandatory Steps

The procedures for process components must contain at least the following steps:

- Paths to handle provide, cease, or update
- Ad-hoc event step to handle cancel and suspend operations
- Task complete

Field Inputs/Outputs

The field input required by an execution plan is supplied via a message payload specified when you create the EAI Orchestrator step with a Start Plan step type. For example:



Available Fields

FEAS
OPSTATUS
OPTEMPLATEID
ORDERID
PLANID
SWO_ORDERNUMBER
SWO_PLANID
SWO_TASKID

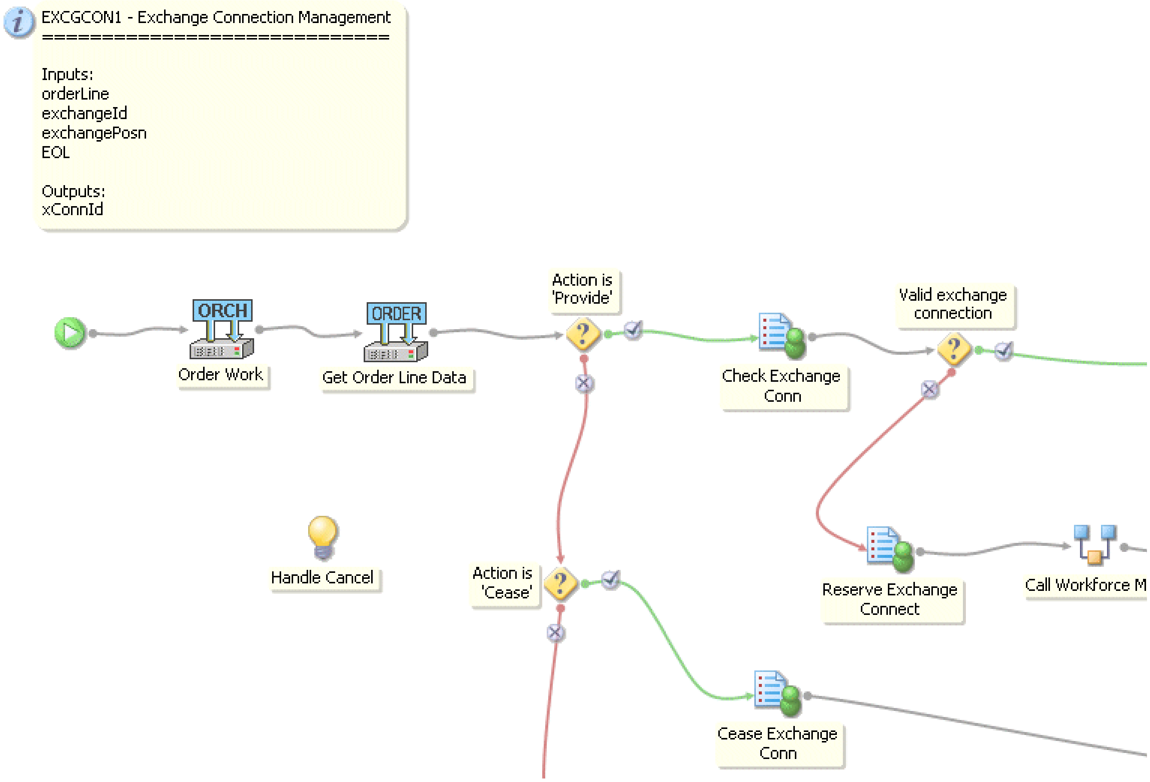
Selected PayLoad Fields

Num	Staffware Field
1	SWO_AMENDINDEX

You also must ensure that the necessary inputs and outputs required by individual EAI Orchestration steps are passed as required.

Example

An example fragment of the procedure that will be associated with the `exchange connection` process component from the Home Broadband and Networking example is illustrated below:



```
EXCGCON1 - Exchange Connection Management
==============================

Inputs:
orderLine
exchangeId
exchangePosn
EOL

Outputs:
xConnId
```

As you can see from this procedure the required inputs and outputs are annotated in the upper left corner, and there are paths in the procedure for Provide, Cease and Update.

Example Process Component: Home Broadband and Networking

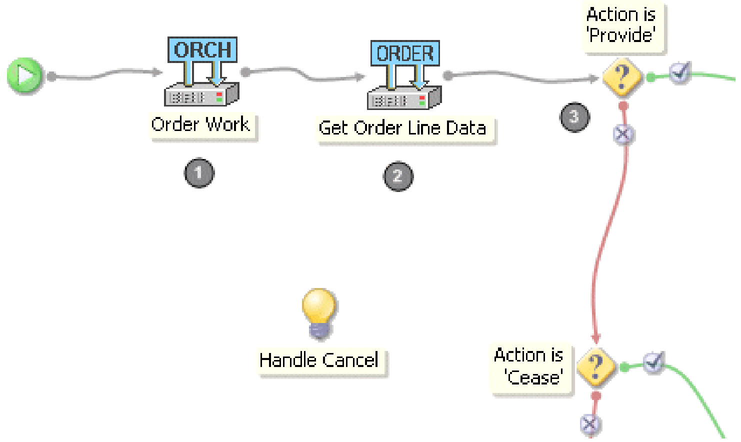
The objectives of the `Exchange Connection` process component from the Home Broadband and Networking example are to:

- Check the inventory system to see if an exchange connection has been provisioned for the customer.
- If no valid exchange connection is listed, reserve an exchange connection in the inventory system.

- Schedule workforce management to send an engineer to configure the physical connection.

This section examines some steps from this example process component:

The numbers in the diagrams correspond to the numbered explanations that follow each diagram.

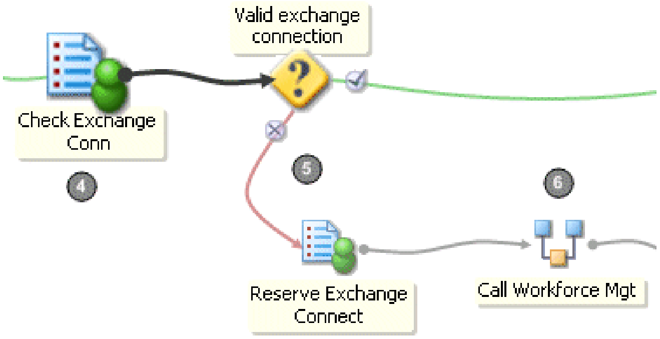


1. The first step (Order Work) is an EAI Orch step that corresponds to a milestone in the execution plan.
2. Next the EAI Order step gets the order line status.

The procedure should also contain an ad-hoc step which is a TIBCO iProcess event step that is used to handle the cancel and suspend operations. In this case, the Cancel path has been omitted for clarity.

3. A condition checks to see if the action is Provide.

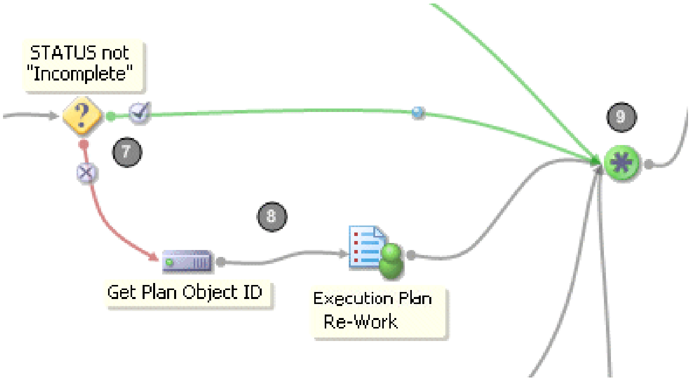
The Cease and Update paths have been omitted for clarity.



4. In the case of a Provide action, the next step checks the inventory (this is simulated in the example by a normal step).
5. The next step is a condition:

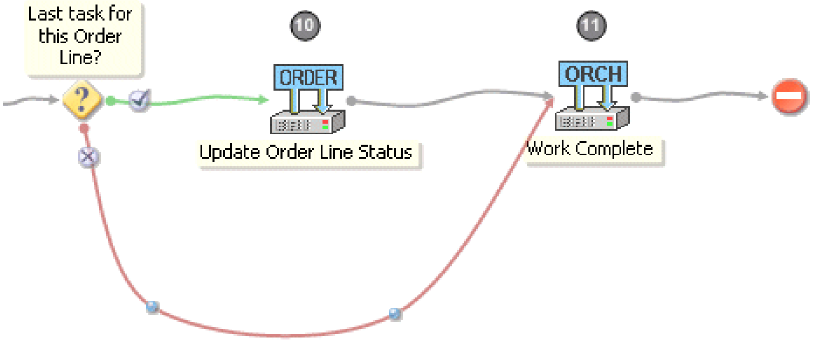
   — If the exchange connection is valid, the procedure joins the complex router (shown in the next screen example).

   — If the exchange connection is not valid, the next step reserves a new exchange connection in the inventory system.
6. This step reserves an appointment with an engineer in a workforce management system for the physical configuration of the exchange.



7. A condition checks the status. The syntax "STATUS is not 'Incomplete'" is used because there are actually two different Complete conditions ("Complete" and "Complete with Exceptions"). This means that although the syntax of the condition is difficult to understand, it is used because it makes the overall plan more comprehensible.
8. If the job could not be done (the status is "Incomplete"), the execution plan is developed manually.

9. The Provide, Cease and Update branches all merge in a complex router.



10. The order line status is updated.
11. The last step is an EAI Orchestration step that corresponds to a milestone in the execution plan.

Defining Error Handling Procedures

There are two types of error handling that are described in this section:

- System-level error handling
- Process component-level error handling.

System-Level Error Handling

You must define two iProcess procedures to be invoked whenever there is a system-level order submission error or orchestration error, respectively, in TIBCO iProcess Conductor.

Order Submission Error Handling

An example of a system-level order handling error is when an order request not adhering to the XML Schema is sent to TIBCO iProcess Conductor.

Order submission error handling is configured in the OrchExceptionHandlerConfig.properties file (located in the appframeworks directory in the iProcess Conductor domain, for example `\opt\bea\user_projects\domains\iProcessConductor\appframeworks\`). By default, the configuration file calls an order submission error handling procedure named ORERRHLR. If you create your error handling procedure with the same name, you do not need to change the configuration file. However, if your procedure has a different name, you must change the orderException.sw.procName property in the configuration file.

Orchestration Error Handling

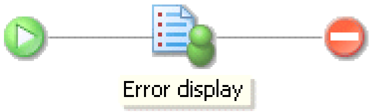
An example of a system-level orchestration error is when a plan task attempts to execute using a procedure that has an invalid data range.

Orchestration error handling is also configured in the `OrchExceptionHandlerConfig.properties` file. By default, the configuration file calls an orchestration error handling procedure named ERRHLR. If you create your error handling procedure with the same name, you do not need to change the configuration file. However, if your procedure has a different name, you must change the `orchestratorException.sw.procName` property in the configuration file.

If you want to alter the default exception handling behavior, you must modify the configuration file. It is also possible to create your own exception handler.

Example

The following example shows a simple iProcess error handler procedure:



This procedure contains an error field that displays the details of the error when the associated work item is opened. You can send the work item to any work queue you want. This example procedure sends the work item to the iProcess Administrator user's and the `COMUsers` queues.

Process Component-Level Error Handling

In addition to system-level error handling, you can create iProcess procedures that handle errors on a process component level.

When you create a process component, you must add a process version to the process component and specify an iProcess procedure in the Exception Handler field. The procedure that you specify should handle errors that arise during execution of the process. The name of the process-component level procedure for error handling is passed to the process in the SWO_EXCEPTION field.

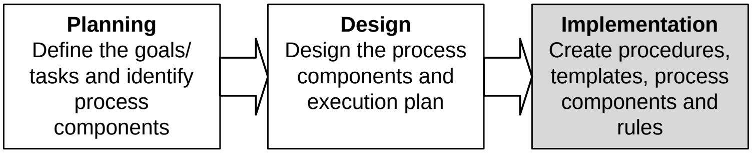
Summary

In this chapter we learned how to design iProcess procedures required to fulfill a customer order, specifically procedures for fulfillment and for process components.

The next chapter describes how to implement order fulfillment by defining the necessary process components.

Chapter 5 Defining Process Components and Execution Plans

This chapter describes how to define process components in TIBCO iProcess Conductor.


Planning
Define the goals/ tasks and identify process components
Design
Design the process components and execution plan
Implementation
Create procedures, templates, process components and rules

Topics

- About Defining Process Components, page 56
- About Defining Execution Plans, page 60

About Defining Process Components

The following steps must be performed when defining a process component in TIBCO iProcess Conductor:

1. Create the process component and define the basic process component information. For example, the process component identifier and description.
2. Associate an iProcess procedure with the process component.

The first iProcess procedure associated with a new process component determines the process component signature. Any further iProcess procedure associated with that process component must have the same signature (public steps, event names and field inputs/outputs for each milestone).

TIBCO iProcess Conductor validates the process signature and if you attempt to add a procedure with an incorrect process signature, it displays a message indicating the reason for the incompatibility.

When you select an iProcess procedure for the process component, you must also do the following:

- Specify durations and resource requirements for the sections of the process (the overall schedule, including maximum and typical durations are calculated automatically).
- Specify any process-level resources required.

3. Define effective date ranges for the procedures associated with the process component.

See *TIBCO iProcess Conductor User's Guide* for more information about how to define process components.

The following section describes process component versions in detail.

Using Process Component Versions

When defining a process component you must associate an iProcess procedure with the process component. By associating an iProcess procedure to a process component, you are creating a version of a process component. You can associate more than one iProcess procedure with a process component.

When you associate an iProcess procedure with a process component, you must specify the effective date for the iProcess procedure. You have the following options:

- Valid From: you must specify either the current date/time or a future date/time from which the procedure takes effect.
- Valid Until: optionally, you can specify a future date/time on which to withdraw the procedure.

You cannot specify an historic date and time for the Valid From field. Also, the date/time for the Valid Until field must be later than the Valid From date/time.

Version Selection

TIBCO iProcess Conductor determines at run-time which version of a process component to use by the date and time that you specify when you define the process component version.

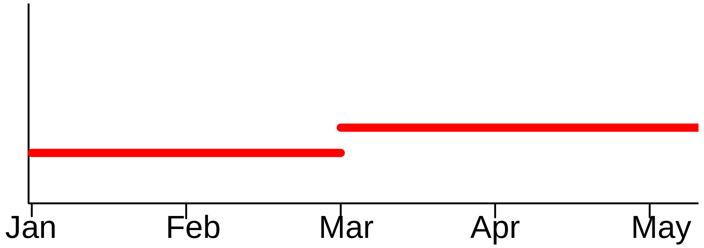
Only one version of a process component can be valid at any one time. When TIBCO iProcess Conductor orchestrates an execution plan it starts the iProcess process that is associated with the process component. If there are multiple iProcess procedures associated with the process component, the following rules apply:

- If there is more than one version, TIBCO iProcess Conductor selects the version with the latest start date and time.
- If there are two versions with the same start dates/times, TIBCO iProcess Conductor selects the version that was created last.

- If you have defined overlapping versions, it may be that different versions are selected between design-time and run-time.
- If there are no effective versions of a process component the execution plan fails to activate.

Examples

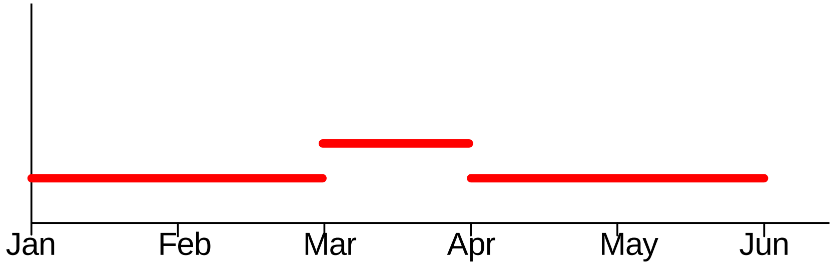
Suppose you may have a process component that fulfils the task of providing a contractor to dig a hole in the road. It may be that you have decided to use a new contractor to dig holes in the road and that the new contractor has a different business process for doing this. Therefore, you define a new version of the iProcess procedure for the new contractor. In this situation, you may want your existing procedure version to be used until the new contractor starts working for you, at which time the new procedure supersedes the old one:



To ensure this happens you need to:

- Define a Valid until date and time for the original version, which means that version of the process component would be withdrawn on that date.
- Define a Valid from date and time for the new process component version. For example, you may want to withdraw the original process component version on 03.06.2004 at 18:00 and start the new version at 04.06.2004 at 09:00.

Another use of process component versions is when you want to temporarily replace your existing version. For example, suppose that you only want to use the new contractor temporarily because your original contractor is unavailable for a month.



To ensure this happens, you only need to define an effective valid from date/time and valid until date/time for the new procedure version for one month. The original process component version would not need its dates changing. This is because, once the one month period is finished, the temporary version of the procedure is withdrawn and the original procedure version is used again.

You may define as many process component versions as you like when first defining a process component. Remember that the process component versions must have the same process signature.

About Defining Execution Plans

Execution plans can be, either:

- a single, standalone execution plan, or
- based on execution plan templates that are defined in TIBCO iProcess Conductor. TIBCO iProcess Conductor instantiates an execution plan on an execution plan template when a customer order is received.

The following elements make up an execution plan:

- Default field values
- Plan tasks
- Task actions
- Dependencies
- Jeopardy conditions and consequential actions.

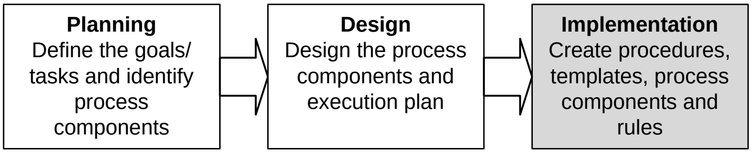
See *TIBCO iProcess Conductor User's Guide* for more information about how to define execution plans.

Summary

Once you have defined your process components and execution plans, continue with the next chapter to set up rules (for example to select the specific execution plan template to be used with a customer order).

Chapter 6 Defining the Rules

This chapter is an overview of the steps required to define the necessary rules in TIBCO iProcess Decisions Plug-in.


Planning
Define the goals/ tasks and identify process components
Design
Design the process components and execution plan
Implementation
Create procedures, templates, process components and rules

Topics

- Overview, page 64
- Creating the Rules Vocabulary, page 65
- Creating Rules for Fulfillment Procedure Selection, page 69
- Creating Rules for Template Selection, page 70
- Making the Rules Available, page 71
- Configuring iProcess Conductor to Use Statically Defined Rules, page 72

Overview

The rules that form a necessary part of an order fulfillment process can be defined in one of two ways:

- By using TIBCO iProcess Decisions Plug-in
- By referring to a set of statically defined rules

Using iProcess Decisions

The default installation of TIBCO iProcess Conductor uses TIBCO iProcess Decisions Plug-in. iProcess Decisions is used to do the following:

- Select fulfillment processes
- Select execution plan templates
- Evaluate business logic in process components

This chapter is not intended as a complete description of how to create rule sets, but rather an overview of some of the aspects of rule creation that are unique to TIBCO iProcess Conductor. For more information on defining rules, see the TIBCO iProcess Decisions Plug-in documentation.

Using Statically Defined Rules

Alternatively, you can configure iProcess Conductor to work with a set of statically defined rules. In this case it will not need an installation of iProcess Decisions Server.

This configuration is done by editing a properties file to specify that iProcess Decisions is not to be used, and that the configured iProcess Engine fulfillment process is used in all circumstances. This process then selects fulfilment processes for order requests and amendments. See Configuring iProcess Conductor to Use Statically Defined Rules on page 72 for details.

Setting up iProcess Conductor to work without iProcess Decisions is a complex process. You should only undertake this is you have experience in updating deployment descriptors for J2EE enterprise application archives.

Creating the Rules Vocabulary

Before you define rules for use with TIBCO iProcess Conductor you must create the Vocabulary that you can use to create your own rules. Both iProcess Conductor and iProcess Conductor Order Management have different Vocabularies. These are described in the following sections.

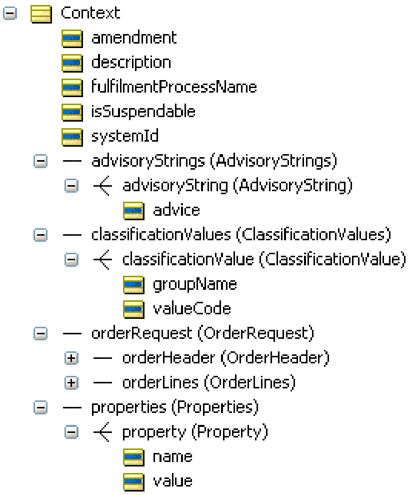
When you create rules, you create conditions that reference the Vocabulary. For example, to evaluate the Customer Reference, specify the following in your rule condition.

```
Context.orderRequest.orderHeader.orderRef
```

If the attributes you want to use in your rules are not part of the default vocabulary, you can add them.

iProcess Conductor Order Management Vocabulary

When receiving XML from rules for use with the iProcess Conductor Order Management, the TIBCO iProcess Conductor expects the following elements:



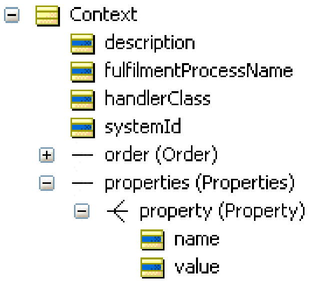
```
⊟ Context
    amendment
    description
    fulfilmentProcessName
    isSuspendable
    systemId
  ⊟ — advisoryStrings (AdvisoryStrings)
    ⊟ advisoryString (AdvisoryString)
        advice
  ⊟ — classificationValues (ClassificationValues)
    ⊟ classificationValue (ClassificationValue)
        groupName
        valueCode
  ⊟ — orderRequest (OrderRequest)
    ⊞ — orderHeader (OrderHeader)
    ⊞ — orderLines (OrderLines)
  ⊟ — properties (Properties)
    ⊟ property (Property)
        name
        value
```

Certain elements such as `AdvisoryString`, `OrderHeader`, `OrderLine` and so on have been created to model repeating parts of the data structure.

Fieldname	Direction	Rules Manager Type	Description
amendment	Passed to Rules	Boolean	Indicates whether the order request is an amendment
description	Filled in by Rules	String	Case description of the fulfillment process
fulfilmentProcess Name	Filled in by Rules	String	Name of the fulfillment process to be started
isSuspendable	Filled in by Rules	Boolean	Indicates to the system whether the fulfillment process should be suspended if an order amendment is received (typically `false` if an error handling process was started)
systemId	Filled in by Rules	String	References the system used to host the fulfillment process (usually `STAFFWARE_i10`) There are additional properties required when you specify an iProcess host (see Properties Required by iProcess on page 68)
advice	Filled in by Rules	String	Value for advisory string
groupName	Filled in by Rules	String	Order Classification Group name (see *TIBCO iProcess Conductor Concepts*)
valueCode	Filled in by Rules	String	Order Classification Group value (see *TIBCO iProcess Conductor Concepts*)
name	Filled in by Rules	String	Property name
value	Filled in by Rules	String	Property value
orderRequest	Passed to Rules	OrderRe quest	Rules Manager type modelling the parts of the order request that are relevant to rules decisions

iProcess Conductor Vocabulary

When receiving XML from rules for use with iProcess Conductor, TIBCO iProcess Conductor expects the following elements:



```
Context
    description
    fulfilmentProcessName
    handlerClass
    systemId
    order (Order)
    properties (Properties)
        property (Property)
            name
            value
```

Fieldname	Direction	Rules Manager Type	Description
description	Filled in by Rules	String	Case description of the fulfillment process
fulfilmentProcess Name	Filled in by Rules	String	Name of the fulfillment process to be started
handlerClass	Filled in by Rules	String	Indicates the class to be used to persist the order. Default implementation is `com.staffware.frameworks.orchestrator.request.runtime.GenericRequestHandler`
systemId	Filled in by Rules	String	References the system used to host the fulfillment process (usually `STAFFWARE_i10`). There are additional properties required when you specify an iProcess host (see Properties Required by iProcess on page 68)

Fieldname	Direction	Rules Manager Type	Description
order	Passed to Rules	Order	Rules Manager type modelling the parts of the order that are relevant to rules decisions. The structure of this type depends on the chosen structure for Orchestrator-only XML orders.
name	Filled in by Rules	String	Property name
value	Filled in by Rules	String	Property value

Properties Required by iProcess

If you specify a `systemID` that references an iProcess host (for example, `STAFFWARE_i10`), you must also pass the following properties back to TIBCO iProcess Conductor:

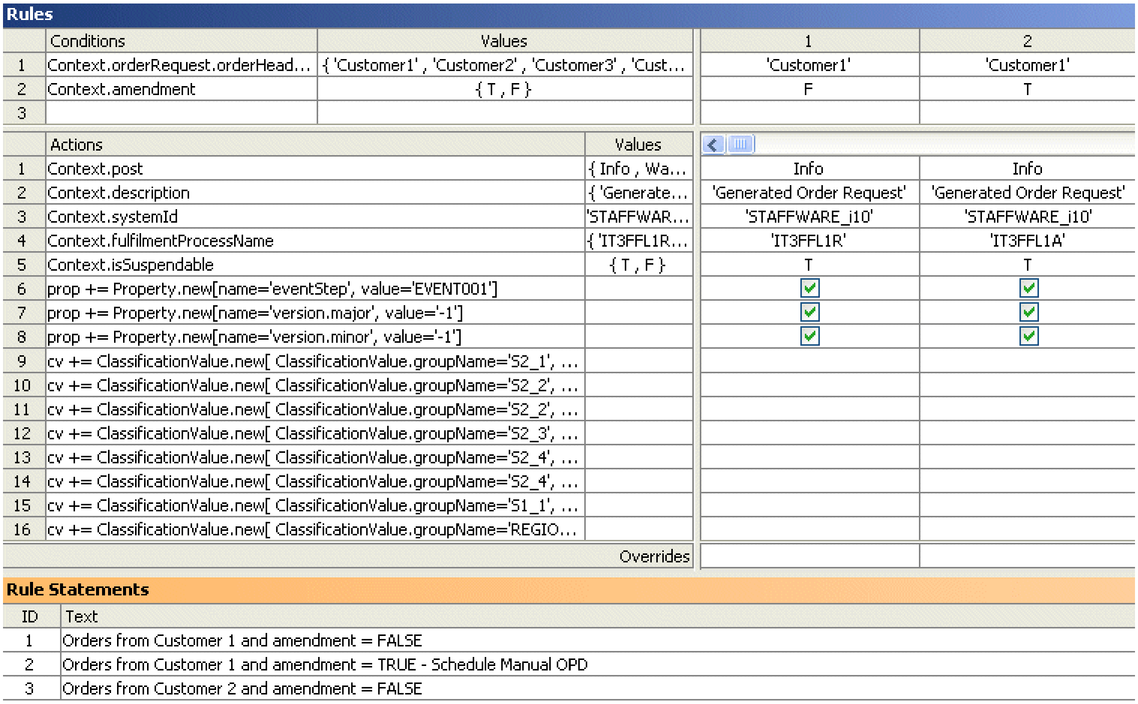
Property	Description
eventStep	(iProcess Conductor Order Management only) Specifies the event step that will be used to handle the situation when an amendment is received and the fulfillment procedure/current amendment procedure is superseded
version.major	Process major version (part of the version number before the decimal) of the fulfillment process
version.minor	Process minor version (part of the version number after the decimal) of the fulfillment process

Creating Rules for Fulfillment Procedure Selection

Once you have identified the orders and the execution plans required to fulfil them, you need to decide what rules you are going to use to identify which fulfillment procedure to use. This is because, depending on the customer order, your business may require different order feasibility and validation. For example, there may be different types of credit checking that need to be applied depending on the payment method the customer uses.

When iProcess Conductor Order Management receives an order or an amendment, it creates an order request and stores a consolidated view of the order in the database. It then sends the order request to iProcess Decisions Plug-in. The order request is in XML format and its contents can be used to create rules. For example, if all orders from wholesale customers have a unique type of customer reference, you can create a rule that selects the appropriate fulfillment procedure based on the customer reference.

Example



Rules

	Conditions	Values	1	2
1	Context.orderRequest.orderHead…	{ 'Customer1' , 'Customer2' , 'Customer3' , 'Cust…	'Customer1'	'Customer1'
2	Context.amendment	{ T , F }	F	T
3				

	Actions	Values	1	2
1	Context.post	{ Info , Wa…	Info	Info
2	Context.description	{ 'Generate…	'Generated Order Request'	'Generated Order Request'
3	Context.systemId	'STAFFWAR…	'STAFFWARE_i10'	'STAFFWARE_i10'
4	Context.fulfilmentProcessName	{ 'IT3FFL1R…	'IT3FFL1R'	'IT3FFL1A'
5	Context.isSuspendable	{ T , F }	T	T
6	prop += Property.new[name='eventStep', value='EVENT001']		☑	☑
7	prop += Property.new[name='version.major', value='-1']		☑	☑
8	prop += Property.new[name='version.minor', value='-1']		☑	☑
9	cv += ClassificationValue.new[ClassificationValue.groupName='S2_1', …			
10	cv += ClassificationValue.new[ClassificationValue.groupName='S2_2', …			
11	cv += ClassificationValue.new[ClassificationValue.groupName='S2_2', …			
12	cv += ClassificationValue.new[ClassificationValue.groupName='S2_3', …			
13	cv += ClassificationValue.new[ClassificationValue.groupName='S2_4', …			
14	cv += ClassificationValue.new[ClassificationValue.groupName='S2_4', …			
15	cv += ClassificationValue.new[ClassificationValue.groupName='S1_1', …			
16	cv += ClassificationValue.new[ClassificationValue.groupName='REGIO…			
		Overrides		

Rule Statements

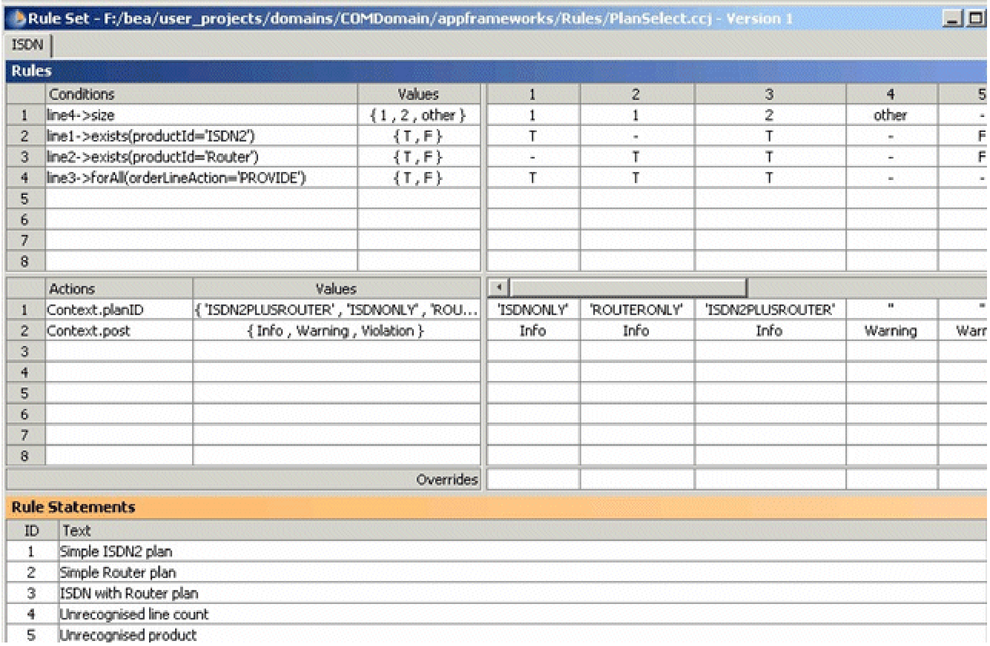
ID	Text
1	Orders from Customer 1 and amendment = FALSE
2	Orders from Customer 1 and amendment = TRUE - Schedule Manual OPD
3	Orders from Customer 2 and amendment = FALSE

1. Rule 1 deals with orders from Customer 1 that are not amendments. For orders that qualify, it specifies the case description, event step, fulfillment procedure and so on.
2. Rule 2 deals with orders from Customer 1 that are amendments, and triggers manual execution plan development.
3. Rule 3 captures orders from Customer 2 that are not amendments.

Creating Rules for Template Selection

You can use TIBCO iProcess Decisions Plug-in to select an execution plan template based on the contents of a customer order. The construction of these rules is entirely business specific.

Example



Rule Set - F:/bea/user_projects/domains/COMDomain/appframeworks/Rules/PlanSelect.ccj - Version 1

ISDN

Rules

	Conditions	Values	1	2	3	4	5
1	line4->size	{ 1 , 2 , other }	1	1	2	other	-
2	line1->exists(productId='ISDN2')	{ T , F }	T	-	T	-	F
3	line2->exists(productId='Router')	{ T , F }	-	T	T	-	F
4	line3->forAll(orderLineAction='PROVIDE')	{ T , F }	T	T	T	-	-
5							
6							
7							
8							

	Actions	Values					
1	Context.planID	{ 'ISDN2PLUSROUTER' , 'ISDNONLY' , 'ROU...	'ISDNONLY'	'ROUTERONLY'	'ISDN2PLUSROUTER'	''	''
2	Context.post	{ Info , Warning , Violation }	Info	Info	Info	Warning	Warr
3							
4							
5							
6							
7							
8							
		Overrides					

Rule Statements

ID	Text
1	Simple ISDN2 plan
2	Simple Router plan
3	ISDN with Router plan
4	Unrecognised line count
5	Unrecognised product

Making the Rules Available

Once you have defined the rules in iProcess Decisions Plug-in, you need to package up the rules into a JAR file and put the JAR file onto the system CLASSPATH. For more information about deploying the Rule sets, see *TIBCO iProcess Conductor Administrator's Guide*.

Configuring iProcess Conductor to Use Statically Defined Rules

You can configure iProcess Conductor to work with a set of statically defined rules. In this case it will not need an installation of iProcess Decisions Server.

This configuration is done by editing a properties file to specify that iProcess Decisions is not to be used, and that the iProcess Engine fulfillment process is used in all circumstances.

Setting up the required configuration involves changes to two files:

Ensure that the application server (JBoss or BEA WebLogic) is shut down before you carry out these operations.

Amending the Application.xml file

Open the `application.xml` file located in the *iProcessConductorDomain*/`applications/COM-ORCH-2.0.ear/META-INF` directory.

Remove the following entries from this file manually:

```
<module>
<java>CcServer.jar</java>
</module>
<module>
<java>CcConfig.jar</java>
</module>
<module>
<java>CcServerLicense.jar</java>
</module>
<module>
<java>CcI18nBundles.jar</java>
</module>
<module>
<java>CcThirdPartyJars.jar</java>
</module>
```

Amending the AppConfig.xml file

Open the `AppConfig.xml` file. The file is located:

- Under BEA WebLogic, in the *iProcessConductorDomain*/`appframeworks` directory.
- Under JBoss, in the *JBOSS_HOME*/`server`/*server_type*/`appframeworks` directory.

1. Navigate to the section:

```
"<!--======== Rules System settings (START)========-->
```

2. Change the `rules.enabled` setting to `false`:

```
<setting> <!-- Determines if iProcess Decisions is enabled -->
      <name>rules.enabled</name>
      <value>false</value>
</setting>
```

3. In the same section give the name of a valid `fulfillmentProcedure`, `description`, and the name of `eventStep` to be used for amendment.

```
<!--=== Values in this section will be used only if rules are not enabled
(START)===-->
      <setting>
      <!-- Please ensure that this points to an existing fulfillment procedure -->
            <name>fulfilment.procedure</name>
            <value>MYPROC</value>
      </setting>

      <setting>
            <name>fulfilment.procedure.description</name>
            <value>My procedure</value>
      </setting>

<!-- Specifies the event step that will be used to handle the situation when an
amendment is received -->
      <setting>
            <name>fulfilment.procedure.eventStep</name>
            <value>EVENT001</value>
      </setting>
```

4. If classification values need to be added to the newly created order, it can be done in `appConfig.xml` by uncommenting the following lines and then providing appropriate values:

```
<!-- Classification values section. Disabled by default. Please supply
classification values in groupName,valueCode format. Ensure that the name is of
syntax fulfilment.procedure.classificationValues-<SequenceNumber>-->
      <setting>
            <name>fulfilment.procedure.classificationValues-1</name>
            <value>WholeSale,Partner</value>
      </setting>
```

Part II - EAI Steps

This part contains information about EAI Orchestrator, EAI Order and EAI Transform steps.

Chapter 7 Using EAI Orchestrator Steps

This chapter describes how the EAI Orchestrator steps can be used, how they work and how they integrate with iProcess procedures.

Topics

- Why Use EAI Orchestrator Steps?, page 78
- Using Messages in EAI Orchestrator Steps, page 79
- Transaction Scope of EAI Orchestration Steps, page 80
- Understanding the EAI Orchestrator Step Process, page 81
- Creating an EAI Orchestrator Step, page 82

Why Use EAI Orchestrator Steps?

EAI Orchestrator steps are used to sequence the process components in an execution plan. They also enable you to send iProcess data to TIBCO iProcess Conductor and for TIBCO iProcess Conductor to process the data before sending the resulting data back to TIBCO iProcess Engine.

When an EAI Orchestrator step is reached in an iProcess procedure, it sends a message to TIBCO iProcess Conductor indicating that it has been reached. Depending on the message type, it can then wait for a message from TIBCO iProcess Conductor before it releases. This enables TIBCO iProcess Conductor to make sure that any tasks that have defined dependencies must have those dependencies satisfied (including the transfer of meta-data) before being allowed to proceed.

Using Messages in EAI Orchestrator Steps

When you define an EAI Orchestration step, you must define the message type that you want the EAI Orchestration step to use. The message type instructs TIBCO iProcess Conductor how it should process the EAI Orchestration step. For example, if you select a message type of Start Plan, this instructs TIBCO iProcess Conductor to activate an execution plan which includes grafting the processes that have been defined in the execution plan or execution plan template identified in the Order Creation sub-procedure.

Message Invocation

The messages use two different methods of invocation depending on the message type you select: delayed release or synchronous/immediate.

The invocation method used depends on whether or not a response is required from TIBCO iProcess Conductor:

- If the EAI Orchestration step does not require any data or is not dependent on another step being released before it can start, then the invocation method used is synchronous/immediate.
- If the EAI Orchestration step does require some data or is dependent on another step then the invocation method used is delayed release. The Activate Sink message type is the only message type to use delayed release. This means the EAI Orchestration step waits for TIBCO iProcess Conductor to send a message and any data back before it releases.

See Define the Call to the TIBCO iProcess Conductor on page 83 for more information about the message types.

Transaction Scope of EAI Orchestration Steps

When designing a procedure containing EAI Orchestration steps, you need to consider the transaction implications in your system design. This is because TIBCO iProcess Engine and iProcess Conductor operate within separate transaction coordinators (Oracle and either BEA WebLogic or JBoss, respectively).

For example, suppose that an EAI Orchestration step is linked to an EAI Oracle step. If there is a failure inside the EAI Oracle step, the transaction is aborted, and iProcess is rolled back to the point just before the execution of the EAI Orchestration step. When the transaction is retried, the EAI Orchestration step is re-processed. In this scenario, it is necessary to design your EAI Orchestration step so that it can handle being called repeatedly without causing a problem.

Refer to “Using Enterprise Application Integration (EAI) Steps” in *TIBCO iProcess Modeler - Integration Techniques* for more information about EAI steps and transactions.

Understanding the EAI Orchestrator Step Process

The following sequence of events occurs each time TIBCO iProcess Conductor is called using the EAI Orchestrator step:

1. The EAI Orchestrator step creates an XML payload based upon the details you have specified in the EAI Orchestrator step definition dialogs.
2. iProcess makes a call to TIBCO iProcess Conductor via the TIBCO iProcess Conductor Java API.
3. Depending on the message type you have selected in the EAI Orchestrator step definition dialogs, either:
   - the EAI Orchestrator step waits for TIBCO iProcess Conductor to send a message back to TIBCO iProcess Engine and then releases.
   - the EAI Orchestrator step releases straight away.

Creating an EAI Orchestrator Step

The following is an overview of the steps involved to create an EAI Orchestrator step in your procedure:

1. Define Basic EAIOrch Step Information - step name, description and type. Optionally, step deadline and audit trail information can also be defined.
2. Define the Call to the TIBCO iProcess Conductor. You need to specify:
   - message type
   - the message
   - the messaging information
   - which fields are passed to TIBCO iProcess Conductor
   - which fields are returned to TIBCO iProcess
3. Make the EAIOrch Step Public.

   When you have done this, the EAI Orchestrator step is defined and the icon appears.

Task A Define Basic EAIOrch Step Information

To define the basic information for the EAI Orchestrator step, do the following:

1. Start TIBCO iProcess Modeler, click **EAI Step tool** and click in the window where you want to place the EAI step.
2. In the EAI Step Definition dialog, enter the name and description for the step.
3. In the EAI Type drop-down list, select **EAI_ORCH - EAI Orchestrator plug-in**. You must enter this when you first create the step; it cannot be changed later. The list box displays EAI step types that have been installed as client EAI plug-ins.
4. Check the **Ignore Case Suspend** checkbox if you want the step to still be processed as normal while a case is suspended by an SPO or SAL application.

   If **Ignore Case Suspend** is not checked (the default option), the step is not processed while the case is suspended.

   Cases can only be suspended and re-activated from an SPO or SAL application. Audit trail messages indicate whether a case is active or suspended. Refer to *TIBCO iProcess Objects Programmer Guide* for more information about suspending cases.

5. Click the **Audit Trail** tab to define custom audit trail entry expressions. This enables you to define text expressions that are evaluated when the step is processed and inserted (as the value of the %USER variable) to identify the user in the audit trail entries:

   — In the Call-out Initiated field, enter a valid text expression that will replace the %USER value in the audit trail when the call out is initiated.

   — In the Call-out Complete field, enter a valid text expression that will replace the %USER value in the audit trail when the call out is complete.

6. The Delayed Release tab is disabled because you define the delayed release setting in the EAI Orchestrator step by specifying a message type - see below.

7. Click **Deadlines** if you want to enter deadline information for this step. Refer to "Using Deadlines" in *TIBCO iProcess Modeler - Basic Design* for an explanation of defining deadlines.

8. Click the **General** tab, then click **EAI Call-Out Definition**. The EAIOrch Step Definition dialog appears.

Task B Define the Call to the TIBCO iProcess Conductor

The General tab of the EAI Orch Step Definition dialog enables you to specify the parameters required to call to TIBCO iProcess Conductor.

To define a call to TIBCO iProcess Conductor:

1. (Mandatory) In the Message Type drop-down list, select the message type you want the EAI Orchestrator step to have.

   You must select a message type because the messages inform TIBCO iProcess Conductor what to do with the current step. The following table describes the message types you can specify:

Message Type	Description	Invocation Method
Start Plan	This instructs TIBCO iProcess Conductor that the execution plan in the parent fulfillment procedure should be started.	Synchronous / Immediate
Activate Sink	This defines a milestone of the plan and instructs TIBCO iProcess Conductor to wait for the completion of ALL activities upon which this milestone depends before releasing this step.	Delayed Release

Message Type	Description	Invocation Method
Task Complete	This instructs TIBCO iProcess Conductor that the process component has executed and is complete.	Synchronous / Immediate
Suspend Plan	You can request that a plan be suspended from within an iProcess procedure or TIBCO iProcess Conductor. If you want the suspend operation to be performed from an iProcess procedure, select this message type when you define the EAI Orchestration step.	Synchronous / Immediate
Activate Plan	You can request that a plan be activated from within an iProcess procedure or TIBCO iProcess Conductor. If you want the activate operation to be performed from an iProcess procedure, select this message type when you define the EAI Orchestration step.	Synchronous / Immediate
Refresh Plan	You can request that a plan be refreshed (re-read from the database) from within an iProcess procedure or TIBCO iProcess Conductor. If you want the refresh operation to be performed from an iProcess procedure, select this message type when you define the EAI Orchestration step.	Synchronous / Immediate
Cancel Plan	You can request that a plan be cancelled from within an iProcess procedure or TIBCO iProcess Conductor. If you want the cancel operation to be performed from an iProcess procedure, select this message type when you define the EAI Orchestration step.	Synchronous / Immediate

2. If you selected the **Start Plan** message type, you need to complete the following information; otherwise continue with the next step:

   — (mandatory) Plan Id Field - Enter the iProcess field that contains the unique execution plan ID generated by the TIBCO iProcess Conductor. The iProcess field must be `SWO_PLANID`. See Mandatory Fields on page 35.

   — (mandatory) Graft Step Name - Enter the name of the graft step defined in the Order Creation sub-procedure that instructs TIBCO iProcess Conductor which graft step to use to orchestrate the required execution plan. See Defining Execution Plan Orchestration on page 42.

- (mandatory) Execution Plan Id Override Field - Enter the iProcess field from your fulfillment procedure that contains the unique identifier for the execution plan or execution plan template. This ID is required because the Plan ID which is generated when the execution plan is instantiated is not available at the start point, so the Execution Plan ID is used instead. This ID is used to identify the execution plan template that this execution plan should instantiate. See Mandatory Fields on page 35.
- (mandatory) Master Record Id Field - Enter the iProcess field that contains the order number that is generated when TIBCO iProcess Conductor receives an order. The iProcess field must be `SWO_ORDERNUMBER`. See Mandatory Fields on page 35.
- (mandatory) Orchestrator Step Name - Enter the name of the iProcess field that contains the name of the ad-hoc orchestration event step. This enables the execution plan ID to be passed back to the fulfillment procedure. See Defining Execution Plan Orchestration on page 42 for more information about the ad-hoc orchestration event step.

3. If you selected **Suspend Plan**, **Activate Plan**, **Refresh Plan** or **Cancel Plan**, you need to complete the following information; otherwise continue with the next step:

   - (mandatory) Plan Id Field - Enter the iProcess field that contains the unique execution plan ID generated by TIBCO iProcess Conductor. The default is `SWO_PLANID`.

4. If you selected **Activate Sink** or **Task Complete**, you need to complete the following information:

   - (mandatory) Plan Id Field - Enter the iProcess field that contains the unique execution plan ID generated by TIBCO iProcess Conductor. The iProcess field must be `SWO_PLANID`. See Mandatory Fields on page 35.
   - (mandatory) Task Id - Enter the iProcess field that contains the unique ID of the plan task that is currently active. It is generated by TIBCO iProcess Conductor. The fieldname must be `SWO_TASKID`. See Mandatory Fields on page 35.

5. Complete the following information:

   - (optional) Message Queue Specification - enter a JMS queue if you want messages from TIBCO iProcess Conductor to go to a specific JMS queue rather than the default JMS queue.
   - (optional) Correlation Id Field - You can specify a correlation ID for the JMS message that is sent from the EAI Orchestrator step. Any messages that are

generated as a result of this step will be tagged with the same correlation Id. This means that you can correlate the JMS request response messages that are generated from the original JMS message from this EAI Orchestrator step. If you are capturing these, enter the name of the iProcess field where the correlation ID is stored. For example, you may want to capture this so that you have an audit trail for a particular batch of messages.

— (optional) Message Id Field - each JMS message generated from an EAI Orchestrator step is automatically allocated a message ID. If you are capturing these, enter the name of the iProcess field containing the JMS return value. For example, you may want to capture these so that you have an audit trail of the JMS messages from TIBCO iProcess Conductor.

Task C Define the Message Payload Fields

The Message Payload Fields tab enables you to specify iProcess fields as data that can be output from TIBCO iProcess Engine to TIBCO iProcess Conductor. Defining Message Payload Fields is optional.

> The fields defined here represent outputs from a plan task step.

1. The Available Fields list displays the iProcess fields that you can pass to TIBCO iProcess Conductor. Select a field and click >> to move it into the Selected Payload Fields list in the right hand pane. To remove a field from the Selected Payload Fields list, select the field and click <<.
2. You can use the Up and Down buttons to position your data, although it is not important to TIBCO iProcess Conductor what the position of the data is.

Jeopardy Management

You can use a payload field in the EAI Orchestrator step to enable jeopardy management to be disabled for individual orders. To do this, add the SWO_JEOPARDYDET field to your EAI Orchestrator step's selected payload fields.

Note that if the fulfillment procedure is a complex procedure which contains sub-procedures, and the EAI Orchestrator step is located in one of those sub-procedures, you must:

1. Add the SWO_JEOPARDYDET field to the EAI Orchestrator step's payload fields in the sub-procedure.

2. Add the SWO_JEOPARDYDET field in that sub-procedure's parent procedure, and in that parent's parent procedure if any, up to the root procedure.
3. Define a parameter:

   `Description=Jeopardy Detection;Field=SWO_JEOPARDYDET`

   in each sub-procedure's IO parameters, and map this parameter to the SWO_JEOPARDYDET field in the sub-procedure’s call definition in its parent procedure.

Jeopardy management can then be disabled for an individual order using the Jeopardy Detection field in the Order Management window. See "Defining an Order" in *TIBCO iProcess Conductor User’s Guide* for details.

Task D Define the Orchestrator Response Fields

The Orchestrator Response Fields tab enables you to specify iProcess fields as data that can be input from TIBCO iProcess Conductor to TIBCO iProcess Engine. Defining Orchestrator Response Fields is optional because you can call to TIBCO iProcess Conductor without passing in any fields.

The fields defined here represent inputs into a plan task step.

1. The Available Fields list displays the iProcess fields that you can pass from TIBCO iProcess Conductor. Select a field and click **>>** to move it into the Selected Payload Fields list in the right hand pane. To remove a field from the Selected Payload Fields list, select the field and click **<<**.
2. You can use the Up and Down buttons to position your data, although it is not important to TIBCO iProcess Conductor what the position of the data is.

Task E Make the EAIOrch Step Public

The EAI Orchestrator step must be made public, using the **Procedure > Public Steps** menu option. See "Defining Public Steps" in *TIBCO iProcess Modeler - Integration Techniques* for details of how to define public steps.

Summary

This chapter described how to incorporate EAI Orchestrator steps into iProcess procedures.

The next chapter examines EAI Order steps in greater detail.

Chapter 8 Using the EAI Order Step

This chapter describes how the EAI Order steps can be used, how they work and how they integrate with iProcess.

Topics

- Why Use an EAI Order Step?, page 90
- Transaction Scope of EAI Order Steps, page 91
- Creating an EAI Order Step, page 92
- Define Basic EAI Order Step Information, page 92
- Define the Call to the iProcess Conductor Order Management, page 93
- Defining an EAI Order Step that Changes the Status of an Order, page 95
- Defining an EAI Order Step that Updates/Outputs Order Parameters, page 97.

Why Use an EAI Order Step?

The EAI Order step has several functions:

- to provide an interface that enables data from a specific order request or order amendment to be used in iProcess fields and vice versa. For example, an EAI Order step enables you to extract a parameter from your order request or order amendment and map it to an iProcess field. This data can then be processed in the iProcess process. For example, you might want to extract an attribute about a phone, like the color, so that it can be retrieved from a warehouse.

  It is also possible to extract order details as XML. Tools are provided with the TIBCO iProcess Conductor that enable you to manipulate the data, depending on your requirements.
- to notify the TIBCO iProcess Conductor of the status of the order depending on what stage has been reached in the fulfillment process.
- to pass the name of the event step to the TIBCO iProcess Conductor so that it knows which step in the process to notify when an order amendment is received.

Transaction Scope of EAI Order Steps

When designing a process containing EAI Order steps, you need to consider the transaction implications in your system design. This is because the TIBCO iProcess Engine and the iProcess Conductor operate within separate transaction coordinators (Oracle and BEA WebLogic or JBOSS, respectively).

For example, suppose that an EAI Order step is linked to an EAI Oracle step. If there is a failure inside the EAI Oracle step, the transaction is aborted, and iProcess is rolled back to the point just before the execution of the EAI Order step. When the transaction is retried, the EAI Order step is re-processed. In this scenario, it is necessary to design your EAI Order step so that it can handle being called repeatedly without causing a problem.

Refer to “Using Enterprise Application Integration (EAI) Steps” in *TIBCO iProcess Modeler - Integration Techniques* for more information about EAI steps and transactions.

Creating an EAI Order Step

The following is an overview of the steps involved to create an EAI Order step in your process:

1. Define Basic EAI Order Step Information - step name, description and type. Optionally, step deadline and audit trail information can also be defined.
2. Define the Call to the iProcess Conductor Order Management. You need to specify one of the following:
   — whether the purpose of the EAI Order step is to change the status of an order, or
   — whether the purpose of the EAI Order step is to update or output order parameters to iProcess fields.

An EAI Order step can either change the status of an order OR update/output order parameters. It cannot do both. This is because you do not want a status change to fail because an update fails. If you want to perform a status change and an update, you should place two EAI Order steps one after the other; one that performs a status change and one that performs an update.

When you have done this, the EAI Order step is defined and the following icon is displayed: ORDER .

Task A Define Basic EAI Order Step Information

Listed below are the steps you need to follow to define an EAI Order step:

1. Start the TIBCO iProcess Modeler, click the **EAI Step tool** and click in the window where you want to place the EAI step.
2. In the EAI Step Definition dialog, enter the Name and Description for the step.
3. In the EAI Type drop-down list, select **EAI_Order - EAI Order plug-in**. You must enter this when you first create the step; it cannot be changed later. The list box displays EAI step types that have been installed as client EAI plug-ins.
4. Check the **Ignore Case Suspend** checkbox if you want the step to still be processed as normal while a case is suspended by a suitable API call.

If **Ignore Case Suspend** is not checked (the default option), the step is not processed while the case is suspended.

Cases can only be suspended and re-activated from an SPO or SAL application. Audit trail messages indicate whether a case is active or suspended. Refer to *TIBCO iProcess Objects: Programmer's Guide* for more information about suspending cases.

5. Click the **Audit Trail** tab to define custom audit trail entry expressions. This enables you to define text expressions that are evaluated when the step is processed and inserted (as the value of the %USER variable) to identify the user in the audit trail entries.

   — In the Call-out Initiated field, enter a valid text expression that will replace the %USER value in the audit trail when the call out is initiated.

   — In the Call-out Complete field, enter a valid text expression that will replace the %USER value in the audit trail when the call out is complete.

6. The Delayed Release tab is disabled because the EAI Order step is always immediate release.

7. Click **Deadlines** if you want to enter deadline information for this step. Refer to “Using Deadlines” in *TIBCO iProcess Modeler - Basic Design* guide for an explanation of defining deadlines.

8. Click the **General** tab, then click **EAI Call-Out Definition**. The EAI Order Step Definition dialog is displayed.

Task B Define the Call to the iProcess Conductor Order Management

To define the call to the TIBCO iProcess Conductor, you must complete the following information in the General tab of the EAI Order Step Definition dialog:

1. (Mandatory) In the Order Primary Key Field drop-down list, select the field that contains the order number.

   The order number is the order’s unique ID. By default, SWO_ORDERNUMBER is selected. This is because the primary key is normally SWO_ORDERNUMBER. You must have declared SWO_ORDERNUMBER in your iProcess Suite process before you are allowed to define the EAI Order step, otherwise an error is generated.

   This value is supplied into the fulfillment process when the order is created (either as part of the process or from the TIBCO iProcess Conductor user interface). This is when its name is defined. See *TIBCO iProcess Modeler - Basic Design* for more information about defining fields in the TIBCO iProcess Engine. The only circumstances in which you may choose another field to be the primary key is if you have copied the order number to another field and wanted to use that field instead.

2. (Mandatory) In the Order Amend Index drop-down list, select the field that contains the index that you want to use for this EAI Order step.

   By default, the SWO_AMENDINDEX field is selected. This is because the index field is normally SWO_AMENDINDEX. You must have declared SWO_AMENDINDEX in your iProcess Suite process before you are allowed to define the EAI Order step, otherwise an error is generated.

   This value is supplied into the fulfillment process when the order is created (either as part of the process or from the TIBCO iProcess Conductor User Interface). This is when the name of the field that contains the index is defined. See *TIBCO iProcess Modeler - Basic Design* for more information about defining fields in the TIBCO iProcess Engine. The only circumstances in which you may choose another field to be the index field is if you have copied the index number to another field and wanted to use that field instead.

3. Then, if you want the EAI Order step to:

   — change the status of an order, see Defining an EAI Order Step that Changes the Status of an Order on page 95.

   — update/output order parameters to iProcess fields, see Defining an EAI Order Step that Updates/Outputs Order Parameters on page 97.

Task C Defining an EAI Order Step that Changes the Status of an Order

The General tab of the EAI Order Step Definition dialog enables you to define an EAI Order step that changes the status of an order. To do this:

1. From the Order Transition State drop-down list, select the order status that you want this EAI Order step to set. See *TIBCO iProcess Conductor Concepts* for more information about valid state transitions for an order. The following table describes the state changes that the EAI Order step notifies iProcess Conductor Order Management to set:

State Change	Description
Order Plan Development	Set the status of the order from Submit to Plan Development.
Feasibility	Set the status of the order from Draft to Feasibility.
Feasibility Failed	Set the status of the order from Feasibility to Feasibility Failed.
Execution	Set the status of the order from Feasibility to Execution.
Suspended	Set the status of the order from Execution to Suspended.
Cancelled	Set the status of the order to Cancelled.

   The iProcess Modeler does not perform any validation to check that you have selected a valid state change. For example, the iProcess Modeler allows you to place an EAI Order step that sets the status of an order to Pending after an EAI Order step that sets the status of the order to Complete. However, if you do select an invalid state change then, at-runtime, the process will fail and a sw_warn entry is generated to inform you that you tried to perform an invalid state change.

2. (Optional) In the Ad-Hoc Event Step field, specify the name of the ad-hoc event step that you want iProcess Conductor Order Management to notify of order amendments. Once you have specified the name of the ad-hoc event step, you need to do the following:

- In the Event/Case Procedure box, you must specify which iProcess fields the EAI Order step should get the process name and case number from. From the drop-down list select either:

  — **This Procedure** if the process is using the default iProcess fields of SW_CASENUM and SW_PRONAME, or

  — **Specify Fields** if the process is using different iProcess fields for the case number and process name. Then, from the drop-down list of iProcess fields in the next two boxes, select the iProcess field you have defined for the case number and process name, respectively.

If the process is defined as a sub-process there will also be an option to select Main Procedure from the list, which then uses the fields SW_MAINCASE and SW_MAINPROC. These define the case number and the process name respectively, of the main process from which the sub-process was called.

3. (Optional) In the Order Line State Change field, you must specify:

— the location in the XML code where you want the status to change. This is achieved using XPath queries. For more information about using XPath, refer to the W3C web site at *http://www.w3.org*.

Use an XPath expression to locate the value in the XML that you want to change. XPath is the W3C standard for selecting an element in a document. An example XPath expression is:

```
/:order/:lines/:status[../:orderLine/:lineNumber = "ABC"]
```

this points to the line number of an order line.

You can also use an iProcess field rather than specifying an explicit value. For example:

```
/:order/:lines/:status[../:orderLine/:lineNumber = "%LINENO%"]
```

In this example `%LINENO%` is an iProcess field with the value ABC.

— the order line status that you want the EAI Order step to set. The following table describes the state changes you can specify:

State Change	Description
Cancelled	The cancel operation performed on the order line is complete.
Complete	The order line has orchestrated and has completed.

4. (Mandatory) Specify how you want non-fatal exceptions from the EAI Order step to be handled.

For fatal exceptions the process hangs and errors are written to the `\`*SWDIR*`\logs\sw_warn` and the eaijava log files in the `\`*SWDIR*`\logs\` directory. The iProcess Suite process is left halted to prevent any harm to the data.

The following table describes some examples of fatal and non-fatal exceptions:

Fatal Exceptions	Non-Fatal Exceptions
Attempting to change the status of an order to an invalid status. For example, attempting to set the order status to Feasibility when the order status is currently Execution Plan Development. See *TIBCO iProcess Conductor Concepts*.	Incorrect XPath has been specified in the EAI Order Step.
Attempting to update the status of a non-existent order line, or multiple order lines.	A record not found.

You can either:

— Select the **Fail Step on XML Exception** checkbox. This box is selected by default. Leaving this selected means that the EAI Order step causes the process to stop and logs the messages to the `eaijava` log files in the `\`*SWDIR*`\logs\` directory, or

— De-select the **Fail Step on XML Exception** checkbox. In the Write Exception to Field field, specify the name of an iProcess field that you want to populate with the message from the non-fatal exception. This is useful, if you want to be able to deal with the exception manually. For example, you could use this field in an error handling process. See Defining Error Handling Procedures on page 51.

Task D Defining an EAI Order Step that Updates/Outputs Order Parameters

The EAI Order step provides an interface that enables data from a specific order request or order amendment to be used in iProcess fields and vice versa.

Specify Update Mappings

The Updates tab enables you to extract data from an iProcess field and update an order header or line with the extracted data.

The following tabs enable you to do this:

- Order Header Elements
- Order Line Elements
- Order Request Fulfillment Elements

Defining Order Header Elements

This tab enables you to specify the iProcess fields that contain the data you want to extract and map to the parameters in the order header that contains the data you want to update. See Adding a Field Mapping on page 101.

You can delete any mappings you have defined at any time. See Deleting a Field Mapping on page 101.

Defining Order Line Elements

This tab enables you to specify the iProcess fields that contain the data you want to extract and map them to the order line elements that contain the data you want to update. To do this:

1. Specify the location of the order line parameter in the XML code where you want the status to change. This is achieved using XPath queries. For more information about using XPath, refer to the W3C web site at *http://www.w3.org*.

   Use an XPath expression to locate the value in the XML that you want to update. XPath is the W3C standard for selecting an element in a document. An example XPath expression is:

   ```
   /:order/:lines/:status[../:orderLine/:lineNumber = "ABC"]
   ```

   this points to the line number of an order line.

   You can also use an iProcess field rather than specifying an explicit value. For example:

   ```
   /:order/:lines/:status[../:orderLine/:lineNumber = "%LINENO%"]
   ```

   In this example `%LINENO%` is an iProcess field with the value ABC.

2. Specify the order line parameter that you want to update and map it to the iProcess field that contains the data you want to extract. See Adding a Field Mapping on page 101.

You can delete any mappings you have defined at any time. See Deleting a Field Mapping on page 101.

Defining Order Request Fulfillment Elements

This tab enables you to extract data from the fulfillment process associated with order requests or amendments based on the value supplied in the SWO_AMENDINDEX field. The value is `0` for an order request and, for order amendments, `1`..*n* where *n* is the unique number for each order amendment.

You can specify the iProcess fields that contain the data you want to extract and map to the order request fulfillment elements that contain the data you want to update. For example, you may want iProcess Conductor Order Management to use a different event step for receiving amendments or you may want the fulfillment process to use a different iProcess Suite process. See Adding a Field Mapping on page 101.

You can delete an order element at any time. See Deleting a Field Mapping on page 101.

Specify Output Mappings

The Outputs tab enables you to specify the data from the order header or order line you want to extract and map it to an iProcess field. This means that the data can now be processed in the iProcess Suite process. For example, you may want to extract a delivery address of a product so that it can be displayed in a work item.

The following tabs enable you to do this:

- Order Content
- Order Header Elements
- Order Line Elements
- Order Request Fulfillment Elements

Defining Order Content

This tab enables you to specify parameters from the main XML content of the Order details that contains the data you want to extract and map it to the iProcess field that you want to update with the extracted data. See Adding a Field Mapping on page 101.

You can delete any mappings you have defined at any time. See Deleting a Field Mapping on page 101.

Defining Order Header Elements

This tab enables you to specify the parameters from the order header that contains the data you want to extract and map to the iProcess field that you want to update with the extracted data. See Adding a Field Mapping on page 101.

You can delete any mappings you have defined at any time. See Deleting a Field Mapping on page 101.

Defining Order Line Elements

This tab enables you to specify the parameter from the order line that contains the data you want to extract and map it to the iProcess field that you want to update with the extracted data. To do this:

1. Specify the location of the order line parameter in the XML code that you want to extract. This is achieved using XPath queries. For more information about using XPath, refer to the W3C web site at *http://www.w3.org*.

   Use an XPath expression to locate the value in the XML that you want to extract. XPath is the W3C standard for selecting an element in a document. An example XPath expression is:

   ```
   /:order/:lines/:status[../:orderLine/:lineNumber = "ABC"]
   ```

   this points to the line number of an order line.

   You can also use an iProcess field rather than specifying an explicit value. For example:

   ```
   /:order/:lines/:status[../:orderLine/:lineNumber = "%LINENO%"]
   ```

   In this example `%LINENO%` is an iProcess field with the value ABC.

2. Specify the order line parameter that you want to extract and map it to the iProcess field that you want to update with the extracted data. See Adding a Field Mapping on page 101.

You can delete any mappings you have defined at any time. See Deleting a Field Mapping on page 101.

Defining Order Request Fulfillment Elements

This tab enables you to extract data from the fulfillment process associated with order requests or amendments based on the value supplied in the SWO_AMENDINDEX field. The value is `0` for an order request and, for order amendments, `1`..*n* where *n* is the unique number for each order amendment.

You can specify the data you want to extract from the fulfillment element and map it to the iProcess field that you want to update with the extracted data. See Adding a Field Mapping on page 101.

You can delete an order element at any time. See Deleting a Field Mapping on page 101.

Adding a Field Mapping

To map an order element to an iProcess field, do the following:

1. Click to add a field to the Field Mapping table.
2. From the Description column, select the order element from the drop-down list of available order elements that you want to map to an iProcess field.
3. From the Field column, select the iProcess field from the drop-down list of available iProcess fields that you want the order element to map to.

Deleting a Field Mapping

To delete a field mapping, do the following:

1. From the Field Mapping table, select the field mapping you want to delete by clicking the box for the field mapping from the Del.? column. You can select as many field mappings as you need.
2. Click . The field mappings are deleted.

Summary

This chapter described how to incorporate EAI Order steps into iProcess procedures.

The next chapter examines EAI Transform steps in greater detail.

Chapter 9 Using the EAI Transform Step

This chapter describes how EAI Transform steps can be used, how they work and how they integrate with iProcess.

Topics

- Why Use an EAI Transform Step?, page 104
- Understanding the EAI Transform Step Process, page 105
- Creating an EAI Transform Step, page 106
- Define Basic EAITransform Step Information, page 106
- Define the EAI Transform Step Input Parameters, page 107
- XSLT Transformation, page 111
- Define the EAI Transform Step Output Parameters, page 112

Why Use an EAI Transform Step?

EAI Transform steps provide the facility to transform XML to/from iProcess field data and/or a designated URL. For example, the EAI Transform step could be used to take XML data from an iProcess memo field, apply a transformation to the data, and pass the result to another memo field. Equally, XML data can be parsed and mapped onto discrete iProcess fields.

The EAI Transform step enables you to:

- populate iProcess fields with XML
- map input iProcess fields to XML
- extract XML content and structure
- map XML to output iProcess fields

The data that is passed in and out of iProcess is called an XML payload.

EAI Transform steps only support synchronous/immediate invocation.

Understanding the EAI Transform Step Process

When you install the EAI Transform plug-in, the EAI Transform Server plug-in is registered with the TIBCO iProcess Engine.

The EAI Transform step accepts an XML document and applies XSLT to give another XML document the possibility of mapping iProcess fields in and out.

Creating an EAI Transform Step

The following is an overview of the steps involved to create an EAI Transformation step in your procedure:

1. Define Basic EAITransform Step Information - step name, description and type. Optionally, step deadline and audit trail information can also be defined.
2. Define the EAI Transformation step depending on your requirements. You need to specify:

   — the source of the XML.

   — which fields from the iProcess Conductor to pass to the XML source, and vice versa.

   — whether the structure of the XML needs to be changed

   When you have done this, the EAI Transformation step is defined and the following icon is displayed: X-FRM

Task A Define Basic EAITransform Step Information

To define an EAI Transformation step, do the following:

1. Start the TIBCO iProcess Modeler, click **EAI Step tool** and click in the window where you want to place the EAI step.
2. In the EAI Step Definition dialog, enter the Name and Description for the step.
3. In the EAI Type drop-down list, select **EAI_TRANSFORM - EAI Transform plug-in**. You must enter this when you first create the step; it cannot be changed later. The list box displays EAI step types that have been installed as client EAI plug-ins.
4. Check the **Ignore Case Suspend** checkbox if you want the step to still be processed as normal while a case is suspended by an SPO or SAL application.

   If **Ignore Case Suspend** is not checked (the default option), the step is not processed while the case is suspended.

Cases can only be suspended and re-activated from an SPO or SAL application. Audit trail messages indicate whether a case is active or suspended. Refer to *TIBCO iProcess Objects: Programmer Guide* for more information about suspending cases.

5. (Optional) Select the **Don't delete work items on withdraw** option. If this option is selected, and the deadline on an outstanding step expires or it is withdrawn as an action (release or deadline expire) of another step:
   — the deadline actions are processed.
   — the step remains outstanding (the step remains in the workqueue or the sub-procedure case is not purged).
   — when the step is released (or the sub-procedure case completes) the normal release actions are not processed but the case field data associated with the release step (e.g. the field values set in a normal step whilst in a work queue or the output parameters of a sub-case) is applied to the main case data.
6. Click the **Audit Trail** tab to define custom audit trail entry expressions. This enables you to define text expressions that are evaluated when the step is processed and inserted (as the value of the %USER variable) to identify the user in the audit trail entries:
   — In the Call-out Initiated field, enter a valid text expression that will replace the %USER value in the audit trail when the call out is initiated.
   — In the Call-out Complete field, enter a valid text expression that will replace the %USER value in the audit trail when the call out is complete.
7. The Delayed Release tab is disabled because EAI Transform steps only support synchronous/immediate invocation.
8. Click **Deadlines** if you want to enter deadline information for this step. Refer to "Using Deadlines" in *TIBCO iProcess Modeler - Basic Design* guide for an explanation of defining deadlines.
9. Click the **General** tab, then click **EAI Call-Out Definition**. The EAI Transform Step Definition dialog is displayed.

Task B Define the EAI Transform Step Input Parameters

You need to specify what iProcess data you want to send out and how the data is formed into XML. For example, you might be performing the XML transformation to transform an input XML document from one format to another or to match a specific target XML schema. The following tabs enable you to specify this:

- XML input
- Input Data Mapping

Example Scenario

Previous iProcess steps have captured data about a customer order in XML format and this is located in an iProcess field. You want to pass this data to the TIBCO iProcess Conductor so that it can be processed in an execution plan. You require the resulting data to be sent back to iProcess.

In this example, you need to specify that the XML source is an iProcess field and then enter the field that contains the XML. If the XML data does not match the expected XML schema for the TIBCO iProcess Conductor, you need to define some XSLT code to transform the data into a valid XML payload.

Complete the following steps to define the source XML data you want to pass to the TIBCO iProcess Conductor.

1. Click the XML input tab. This tab enables you to specify the XML source that you want to pass to the iProcess Conductor.

Remember that the final XML source that iProcess sends out must match the XML schema or format that it is expecting. You can use the XSLT tab to manipulate your XML in order to make it validate against the XML schema or format.

2. Choose one of the following options to define how you want to capture your XML data:

Option	Description
Static XML	Define an XML code fragment that you want to send to the iProcess Conductor. You can then paste the XML in the XML Data/Location text box.
iProcess Field	Specify that you want the XML code to be read from an iProcess field.
iProcess Expression	Specify that you want the XML code to be derived from an iProcess expression.
URL	Define that you want the XML code to be derived from a specific URL. You need to enter the exact URL in the XML Data/Location text box. For example, http://.../doc.xml or file://.../doc.xml.

3. Specify the XML data or the location of the data using the following options:

Option	Description
XML Data / Location	Enter or paste the XML data into the XML Data/Location text box. If you chose the URL option, you need to enter the URL that points to the XML location.
Import URL Data	Only use this option if you have specified to use XML from a URL. Choose when the XML data from the URL you have specified is imported. Choose one of the following options: • Design-time - the XML data is immediately retrieved from the URL when the step is saved. • Run-time - the XML data is imported from the URL when a case of the procedure is run.

Task C Define Data Mapping for Input Parameters

This tab enables you to specify how data in the XML source code is mapped to iProcess data. This means that you can substitute XML data with an iProcess field, array or expression so that the data is captured at run-time from the iProcess field, expression or array and inserted into the XML code.

Example Scenario

If you are sourcing the XML from an iProcess field and this has been captured from an external database, you can use an XPath expression to substitute one or more data values with iProcess field(s). For example, if the source XML code is:

```
<root>
 <employee id="1">
  <name first="Mark" last="Stanley" />
  <custid>2xn45</custid>
 </employee>
</root>
```

and you want to replace the `custid` data value with an iProcess field, you would enter the following XPath expression:

```
/root/employee[@id='1']/custid/text()
```

For more information about using XPath, refer to the W3C web site at *http://www.w3.org*.

Complete the following steps to define how iProcess fields are mapped to the source XML data.

1. Define the location in the XML code where you want to replace an element with iProcess data. This is achieved using XPath queries. For more information about using XPath, refer to the W3C web site at *http://www.w3.org*.

   Use an XPath expression to locate the value in the XML that you want to substitute. XPath is W3C standard for selecting an element in a document. An example XPATH expression is:

   ```
   /request/field@name="id"
   ```

   this points to the name attribute of the field element.

2. From iProcess Type, choose the type of iProcess data element that you want to substitute the XML parameter with: Field or Expression. If you want to specify an iProcess array field select **Field** and then select the **Array** checkbox to indicate that the field you choose in the iProcess Data drop-down list is an array.

3. From the iProcess Data drop-down list, select the iProcess data element that you want to insert into the XML source at the point of the XPath expression. If you have selected **Field** as your iProcess Type, the drop-down lists all the available iProcess fields.

4. Click **Add Mapping**. The mapping is then displayed in the Configured Mappings box. You can then add further mappings if required.

If you start adding a mapping but need to clear the contents of the fields, you can click **Reset**. This clears the contents of the Xpath Expression and the iProcess Data fields.

Edit an Existing Mapping

If you want to edit a mapping, select the mapping from the Configured Mappings box, make the changes in the dialog and then click **Update Mapping**.

Remove a Mapping

To remove a mapping, select it from the Configured Mappings list and click **Remove Mapping**.

Task D XSLT Transformation

This tab is used to transform the XML source code that you have specified in the XML Input tab so that it can be validated against the XML schema or required format for the iProcess Conductor. You can specify how to transform XML code using the XSLT (eXtensible Stylesheet Language Transformation) programming language, which is specifically designed for creating a new XML document from an existing one.

For more information about using XSLT, refer to the W3C web site at *http://www.w3.org*.

Example Scenario

For example, you could convert all of the <ProdInfo> tags to <Info> tags. For a more complex example, you could convert every other <para> tag to <p font=“Arial”>.

You could also use XSLT to convert documents from some other data model to your own data model, for example:

If your supplier provides technical marketing information in XML but they use a different data model than you do. You would use XSLT to transform their tags and structure to match yours. For example, they could use a structure that allows <productinfo> to be within <productinfo> where you require an embedded <productinfo> to be a <subsection> instead.

Complete the following steps to specify where the XSLT template is sourced.

1. Choose where the XSLT information will be derived from:

Option	Description
None	Specify that you do not want to use XSLT transformation on your XML data. This assumes that the XML data specified in the XML Input tab is in the expected format for the iProcess Conductor.
Static XSLT	Specify that you have an XSLT template you want to use. You can paste the XSLT contents into the XML Data/Location text box.

Option	Description
iProcess Field	Define if your XSLT transformation code will be derived from the contents of an iProcess field.
iProcess Expression	Choose this option if your XSLT transformation code will be derived from the result of an iProcess expression.
URL	Choose this option if your XSLT transformation code will be derived from a URL. You must then enter the URL of the XSLT template in the XML Data/Location text box.

2. Specify the contents of the XSLT template or the URL location of the XSLT template.

   In the XML Data/Location text area, enter the XSLT code if you have chosen the **Static XSLT** option or enter a URL to specify where the XSLT template is located.

3. Choose if you want to import the XSLT template from the URL now (design time) or when a case is run (runtime).

   — **Design time** - the XSLT template will be derived from the provided URL when the step is saved.

   — **Runtime** - the XSLT code will be derived only when the case is run. You may want to use this if you know that the XSLT is dynamic and you always want to use the latest code.

Task E Define the EAI Transform Step Output Parameters

You need to specify the data that is returned to iProcess. The following tabs enable you to specify this:

- XML output
- Output Data Mapping

The XML output tab enables you to specify the iProcess data that you want to pass to the XML destination. Remember that the final XML source that iProcess sends out must match the XML schema or format that it is expecting. You can use the XSLT tab to manipulate your source XML in order to match the XML schema or required format.

Example Scenario

Data about a customer order has been captured in an iProcess field and you want to pass this data out to your billing system. You want this data to be passed to the billing system so the customer's bill can be produced.

In this example, you need to specify that the XML output is an iProcess field and then select the field that contains the XML. If the XML data does not match the expected XML schema for the iProcess Conductor, you need to define some XSLT code to transform the data into a valid XML payload.

To define the data that you want to pass to the XML destination, you need to specify an iProcess field. To do this, from iProcess Field, select the iProcess field from the drop-down list of available iProcess fields.

Task F Define Data Mapping for Output Parameters

This tab enables you to specify how data from the iProcess Conductor is mapped to the destination XML code. This means that you can substitute XML data with an iProcess field or array so that the data is captured at run-time from the iProcess field or array and inserted into the XML code.

Complete the following steps to define how iProcess fields are mapped to the destination XML data.

1. Define the location in the XML code where you want to replace an element with iProcess data. This is achieved using XPath queries.

   Use an XPATH expression to locate the value in the XML that you want to substitute. XPath is W3C standard for selecting an element in a document. For example, you can specify an XPath expression that points to the third paragraph in the second chapter. For more information about using XPath, refer to the W3C web site at *http://www.w3.org*.

2. From the drop-down list of available iProcess fields, select the iProcess field or array that you want to insert into the XML source at the point of the XPath expression.

3. If the iProcess field/array you want to substitute the XML Parameter with is an array field, select **Array**.

4. Click **Add Mapping**. The mapping is then displayed in the Configured Mappings box. You can then add further mappings if required.

Editing a Mapping

If you want to edit a mapping, select the mapping from the Configured Mappings box, make the changes in the dialog and then click **Update Mapping**.

Remove a Mapping

To remove a mapping, select it from the Configured Mappings list and click **Remove Mapping**.

Appendix A Using iProcess Conductor with TIBCO EMS

This appendix describes how to make iProcess Conductor work with TIBCO Enterprise Message Service™ version 5.0.

Topics

- Introduction, page 116
- Using iProcess Conductor in a Cluster with EMS on page 117
- Using iProcess Conductor under JBoss 4.2.1 with EMS, page 118
- Using iProcess Conductor under JBoss EAP 4.3 with EMS, page 125
- Using iProcess Conductor under BEA WebLogic with EMS, page 132
- Securing the iProcess Conductor JMS Gateway with EMS, page 146

Introduction

TIBCO Enterprise Message Service (EMS) software lets application programs send and receive messages according to the Java Message Service (JMS) protocol. It also integrates with TIBCO Rendezvous and TIBCO SmartSockets message products.

For the iProcess Conductor clustering, EMS is the required as the JMS provider. For the standalone iProcess Conductor, the default installation uses either BEA WebLogic JMS or JBoss MQ/Messaging as the JMS server. It is possible to decouple the JMS server functionality from JBoss/WebLogic and use TIBCO EMS instead.

The process needed to enable iProcess Conductor to operate with EMS varies depending on the application server that iProcess Conductor uses, and on whether you are using a standalone or a clustered iProcess Conductor environment.

Using iProcess Conductor in a Cluster with EMS

To configure an iProcess Conductor clustered installation, using either JBoss or WebLogic, to work with EMS, carry out the following steps:

1. Set up the EMS Fault Tolerant environment. See the TIBCO EMS documentation for details.
2. To change the EAI Orchestration server plug-in to work with EMS, modify the `eaiorch.jndi.factory`, `eaiorch.jms.factory` and `eaiorch.jndi.url` values in *$SWDIR*`\eaijava\properties\eaiorch\eaiorch.properties` to specify TIBCO EMS values as follows:

```
# TIBCO iProcess Conductor Orchestration Server Plug-in
eaiorch.jndi.factory=com.tibco.tibjms.naming.TibjmsInitialContextFactory
eaiorch.jndi.url=tibjmsnaming://host_name:port_number
eaiorch.jms.factory=XAQueueConnectionFactory
eaiorch.jms.principal=comEAIStepUser
eaiorch.jms.credentials=staffware
```

And for WebLogic:

```
eaiorch.authconf.location=
```

For JBoss:

```
eaiorch.authconf.location=JBOSS_HOME/client/auth.conf
```

3. Enter the EMS Fault Tolerant context url in the EMS Configuration dialog when you install iProcess Conductor.

Using iProcess Conductor under JBoss 4.2.1 with EMS

To configure an iProcess Conductor non-clustered installation working under JBoss version 4.2.1 to work with EMS, carry out the following steps:

TIBCO EMS Configuration

1. Create the following queues, topics and ConnectionFactories in TIBCO EMS:

   — `factories.conf`:

   ```
   [XAQueueConnectionFactory]
   type = xaqueue
   url = tcp://port_number

   [XATopicConnectionFactory]
   type = xatopic
   url = tcp://port_number
   ```

   — `queues.conf`

   ```
   queue/COMOrchExceptionQueue
   queue/COMOrderExceptionQueue
   queue/COMRequestInjectorQueue
   queue/COMProcessQueue
   queue/COMOrderRequestInjectorQueue
   queue/AOPDProcessQueue
   queue/AOPDResponseQueue
   ```

   queue/COMProcessQueueInner

   queue/DLQ

   queue/JMSFulFillOrderQueue

   — `topics.conf`

   ```
   topic/COMOrchestratorTopic
   topic/IPCJeopardyTopic
   topic/COMOrderManagerTopic
   ```

   where *port_number* is the port number that your EMS server is configured to use. If this value is not specified, it defaults to `7222`.

2. Make a backup copy (in a separate directory) of the configuration files that will be changed:

   — *JBOSS_HOME*`\server\default\deploy\jms\jms-ds.xml`

   — *JBOSS_HOME*`\server\default\conf\jndi.properties`

   — *JBOSS_HOME*`\server\default\conf\standardjboss.xml`

3. Reconfigure the `JMSProviderLoader` mbean to load TIBCO Enterprise Message Service instead of loading JBoss MQ.

   Add the following lines in *JBOSS_HOME*`\server\default\deploy\jms\jms-ds.xml` to cause the `JMSProviderLoader` mbean to load TIBCO Enterprise Message Service:

```
<!-- The JMS provider loader -->
<mbean code="org.jboss.jms.jndi.JMSProviderLoader"
name=":service=JMSProviderLoader,name=TibjmsProvider">
<attribute name="ProviderName">TIBCOJMSProvider
</attribute>
<attribute name="ProviderAdapterClass">org.jboss.jms.jndi.JNDIProviderAdapter
</attribute>
<attribute name="QueueFactoryRef">XAQueueConnectionFactory
</attribute>
<attribute name="TopicFactoryRef">XATopicConnectionFactory
</attribute>
<attribute name="Properties">
java.naming.security.principal=ipcuser
java.naming.security.credentials=ipcuser
java.naming.factory.initial=com.tibco.tibjms.naming.TibjmsInitialContextFactory
java.naming.factory.url.pkgs=com.tibco.tibjms.naming
java.naming.provider.url=tibjmsnaming://host_name:port_number
</attribute>
</mbean>
```

4. Add the following lines in *JBOSS_HOME*`\server\default\deploy\jms\jms-ds.xml` to provide mapping between local JNDI names and those present in TIBCO EMS:

```
<!-- Redirect QueueConnectionFactory to TIBCO Enterprise Message Service -->
<mbean code="org.jboss.naming.NamingAlias"
name="DefaultDomain:service=NamingAlias,fromName=TIBCO.QueueConnectionFactory">
<attribute name="ToName">tibjmsnaming://host_name:port_number/XAQueueConnectionFactory
</attribute>
<attribute name="FromName">TIBCO.QueueConnectionFactory
</attribute>
</mbean>

<!-- Redirect TopicConnectionFactory to TIBCO Enterprise Message Service -->
<mbean code="org.jboss.naming.NamingAlias"
name="DefaultDomain:service=NamingAlias,fromName=TIBCO.TopicConnectionFactory">
<attribute name="ToName">tibjmsnaming://host_name:port_number/XATopicConnectionFactory
</attribute>
<attribute name="FromName">TIBCO.TopicConnectionFactory
</attribute>
```

```
</mbean>

<!-- Redirect COMRequestInjectorQueue to TIBCO Enterprise Message Service -->
<mbean code="org.jboss.naming.NamingAlias"
name="DefaultDomain:service=NamingAlias,fromName=queue/COMRequestInjectorQueue">
<attribute
name="ToName">tibjmsnaming://host_name:port_number/queue/COMRequestInjectorQueue
</attribute>
<attribute name="FromName">queue/COMRequestInjectorQueue
</attribute>
</mbean>

<!-- Redirect COMOrderRequestInjectorQueue to TIBCO Enterprise Message Service -->
<mbean code="org.jboss.naming.NamingAlias"
name="DefaultDomain:service=NamingAlias,fromName=queue/COMOrderRequestInjectorQueue
">
<attribute
name="ToName">tibjmsnaming://host_name:port_number/queue/COMOrderRequestInjectorQueue
</attribute>
<attribute name="FromName">queue/COMOrderRequestInjectorQueue
</attribute>
</mbean>

<!-- Redirect COMOrchExceptionQueue to TIBCO Enterprise Message Service -->
<mbean code="org.jboss.naming.NamingAlias"
name="DefaultDomain:service=NamingAlias,fromName=queue/COMOrchExceptionQueue">
<attribute
name="ToName">tibjmsnaming://host_name:port_number/queue/COMOrchExceptionQueue
</attribute>
<attribute name="FromName">queue/COMOrchExceptionQueue
</attribute>
</mbean>

<!-- Redirect COMOrderExceptionQueue to TIBCO Enterprise Message Service -->
<mbean code="org.jboss.naming.NamingAlias"
name="DefaultDomain:service=NamingAlias,fromName=queue/COMOrderExceptionQueue">
<attribute
name="ToName">tibjmsnaming://host_name:port_number/queue/COMOrderExceptionQueue
</attribute>
<attribute name="FromName">queue/COMOrderExceptionQueue
</attribute>
</mbean>

<!-- Redirect COMProcessQueue to TIBCO Enterprise Message Service -->
<mbean code="org.jboss.naming.NamingAlias"
name="DefaultDomain:service=NamingAlias,fromName=queue/COMProcessQueue">
<attribute name="ToName">tibjmsnaming://host_name:port_number/queue/COMProcessQueue
</attribute>
<attribute name="FromName">queue/COMProcessQueue
</attribute>
</mbean>

<!-- Redirect COMProcessQueueInner to TIBCO Enterprise Message Service -->
<mbean code="org.jboss.naming.NamingAlias"
name="DefaultDomain:service=NamingAlias,fromName=queue/COMProcessQueueInner ">
<attribute
name="ToName">tibjmsnaming://host_name:port_number/queue/COMProcessQueueInner
```

```
</attribute>
<attribute name="FromName">queue/COMProcessQueueInner
</attribute>
</mbean>

<!-- Redirect AOPDProcessQueue to TIBCO Enterprise Message Service -->
<mbean code="org.jboss.naming.NamingAlias"
name="DefaultDomain:service=NamingAlias,fromName=queue/AOPDProcessQueue">
<attribute name="ToName">tibjmsnaming://host_name:port_number/queue/AOPDProcessQueue
</attribute>
<attribute name="FromName">queue/AOPDProcessQueue
</attribute>
</mbean>

<!-- Redirect AOPDResponseQueue to TIBCO Enterprise Message Service -->
<mbean code="org.jboss.naming.NamingAlias"
name="DefaultDomain:service=NamingAlias,fromName=queue/AOPDResponseQueue">
<attribute name="ToName">tibjmsnaming://host_name:port_number/queue/AOPDResponseQueue
</attribute>
<attribute name="FromName">queue/AOPDResponseQueue
</attribute>
</mbean>

<!-- Redirect JMSFulfillOrderQueue to TIBCO Enterprise Message Service -->
<mbean code="org.jboss.naming.NamingAlias"
name="DefaultDomain:service=NamingAlias,fromName=queue/JMSFulfillOrderQueue">
<attribute
name="ToName">tibjmsnaming://host_name:port_number/queue/JMSFulfillOrderQueue
</attribute>
<attribute name="FromName">queue/JMSFulfillOrderQueue
</attribute>
</mbean>

<!-- Redirect IPCJeopardyTopic to TIBCO Enterprise Message Service -->
<mbean code="org.jboss.naming.NamingAlias"
name="DefaultDomain:service=NamingAlias,fromName=topic/IPCJeopardyTopic">
<attribute name="ToName">tibjmsnaming://host_name:port_number/topic/IPCJeopardyTopic
</attribute>
<attribute name="FromName">topic/IPCJeopardyTopic
</attribute>
</mbean>

<!-- Redirect COMOrchestratorTopic to TIBCO Enterprise Message Service -->
<mbean code="org.jboss.naming.NamingAlias"
name="DefaultDomain:service=NamingAlias,fromName=topic/COMOrchestratorTopic">
<attribute
name="ToName">tibjmsnaming://host_name:port_number/topic/COMOrchestratorTopic
</attribute>
<attribute name="FromName">topic/COMOrchestratorTopic
</attribute>
</mbean>

<!-- Redirect COMOrderManagerTopicto TIBCO Enterprise Message Service -->
<mbean code="org.jboss.naming.NamingAlias"
name="DefaultDomain:service=NamingAlias,fromName=topic/COMOrderManagerTopic">
<attribute
name="ToName">tibjmsnaming://host_name:port_number/topic/COMOrderManagerTopic
```

```
</attribute>
<attribute name="FromName">topic/COMOrderManagerTopic
</attribute>
</mbean>
```

5. When the JBoss server invokes JNDI and encounters the `tibjmsnaming` scheme, the server must be able to find the TIBCO Enterprise Message Service URLConnectionFactory. Therefore, modify the file *JBOSS_HOME*`\server\default\conf\jndi.properties` as follows:

   Change:

   ```
   java.naming.factory.url.pkgs=org.jboss.naming:org.jnp.interface
   s
   ```

   To:

   ```
   java.naming.factory.url.pkgs=org.jboss.naming:org.jnp.
   interfaces:com.tibco.tibjms.naming
   ```

6. In the *JBOSS_HOME*`\server\default\conf\standardjboss.xml` file, navigate to the XML node:

   ```
   <invoker-proxy-binding>
   <name>message-driven-bean</name>
   ```

   And modify the following line. Change:

   ```
   <JMSProviderAdapterJNDI>DefaultJMSProvider
   </JMSProviderAdapterJNDI>
   ```

   To:

   ```
   <JMSProviderAdapterJNDI>TIBCOJMSProvider
   </JMSProviderAdapterJNDI>
   ```

   This change sets "TIBCOJMSProvider" as the JMS Provider Adapter JNDI name.

7. In the *JBOSS_HOME*`\server\default\conf\standardjboss.xml` file, navigate to the XML node:

   ```
   <invoker-proxy-binding>
   <name>message-driven-bean</name>
   ```

8. And modify the following line. Change:

   ```
   <DestinationQueue>queue/DLQ</DestinationQueue>
   ```

   To:

   `<DestinationQueue>`
   `tibjmsnaming://`*host_name:port_number*`/queue/DLQ`
   `</DestinationQueue>`

   This change specifies the TIBCO Enterprise Message Service JNDI name for the MDB Dead Letter Queue (DLQ) `queue/DLQ`.

9. Move the following files out of the *JBOSS_HOME*`\server\default\deploy` directory. These files are not needed when using TIBCO Enterprise Message Service and therefore must not be deployed:

   JBOSS_HOME`\server\default\deploy\jms\jbossmq-destinations-service.xml`

   JBOSS_HOME`\server\default\deploy\jms\COM-service.xml`

10. Copy `tibjms.jar` to the *JBOSS_HOME*`\server\default\lib` folder.

iProcess Conductor Configuration

11. Open *appFrameworks*/`AppConfig.xml`. Change the values of `QUEUE_CONNECTION_FACTORY` and `TOPIC_CONNECTION_FACTORY` to *TIBCO.QueueConnectionFactory* and *TIBCO.TopicConnectionFactory* respectively.

```
<setting>
<name>QUEUE_CONNECTION_FACTORY</name>
<value>TIBCO.QueueConnectionFactory</value>
</setting>
<setting>
<name>TOPIC_CONNECTION_FACTORY</name>
<value>TIBCO.TopicConnectionFactory</value>
</setting>
```

12. The security section of this file needs to be enabled if queues and topics are to be made secure. See Securing the iProcess Conductor JMS Gateway with EMS on page 146 for further details.

```
<!--==================== JMS settings (START)====================-->
<!-- Enable this section if an external JMS Server is used
<setting>
<name>INTERNAL_QUEUE_USER</name>
<value>ipcuser</value>
</setting>
<setting>
<name>INTERNAL_QUEUE_USER_PASSWORD</name>
<value>ipcuser</value>
</setting>
<setting>
<name>INTERNAL_TOPIC_USER</name>
<value>ipcuser</value>
</setting>
<setting>
<name>INTERNAL_TOPIC_USER_PASSWORD</name>
<value>ipcuser</value>
</setting>
-->
<!--====================  JMS settings (END)====================-->
```

EAI Orchestrator Configuration

13. To change the EAI Orchestration server plug-in to work with EMS, modify the `eaiorch.jndi.factory`, `eaiorch.jndi.url` and `eaiorch.jms.factory` values in *$SWDIR*`\eaijava\properties\eaiorch\eaiorch.properties` to specify TIBCO EMS values.

```
# TIBCO iProcess Conductor Orchestration Server Plug-in
                    eaiorch.jndi.factory=com.tibco.tibjms.naming.TibjmsInitialContextFactory
eaiorch.jndi.url=tibjmsnaming://host_name:port_number
eaiorch.jms.factory=XAQueueConnectionFactory
eaiorch.jms.principal=comEAIStepUser
eaiorch.jms.credentials=staffware
eaiorch.authconf.location=JBOSS_HOME/client/auth.conf
```

14. If it is not already present, copy `tibjms.jar` to the *$SWDIR*`\jmslib\ems` directory. `Tibjms.jar` can be found in the *EMS_HOME*`\ems\5.0\lib` folder.

Restart

15. Restart iProcess Engine.
16. Restart JBoss.

Using iProcess Conductor under JBoss EAP 4.3 with EMS

To configure an iProcess Conductor non-clustered installation working under JBoss EAP 4.3 to work with EMS, carry out the following steps:

TIBCO EMS Configuration

1. Create the following queues, topics and ConnectionFactories in TIBCO EMS:

   — `factories.conf`:

   ```
   [XAQueueConnectionFactory]
   type = xaqueue
   url = tcp://port_number

   [XATopicConnectionFactory]
   type = xatopic
   url = tcp://port_number
   ```

   — `queues.conf`

   ```
   queue/COMOrchExceptionQueue
   queue/COMOrderExceptionQueue
   queue/COMRequestInjectorQueue
   queue/COMProcessQueue
   queue/COMProcessQueueInner
   queue/COMOrderRequestInjectorQueue
   queue/AOPDProcessQueue
   queue/AOPDResponseQueue
   queue/JMSFulFillOrderQueue
   queue/DLQ
   ```

   — `topics.conf`

   ```
   topic/COMOrchestratorTopic
   topic/IPCJeopardyTopic
   topic/COMOrderManagerTopic
   ```

   where *port_number* is the port number that your EMS server is configured to use. If this value is not specified, it defaults to `7222`.

2. Make a backup copy (in a separate directory) of the configuration files that will be changed:

   — *JBOSS_HOME*`\server\`*server_type*`\deploy\hajndijms-ds.xml`

   — *JBOSS_HOME*`\server\`*server_type*`\conf\jndi.properties`

   — *JBOSS_HOME*`\server\`*server_type*`\conf\standardjboss.xml`

   where *server_type* is one of default, all or production, as described in *TIBCO iProcess Conductor Installation*.

3. Reconfigure the `JMSProviderLoader` mbean to load TIBCO Enterprise Message Service instead of loading JBoss MQ.

   Add the following lines in *JBOSS_HOME*`\server\`*server_type*`\deploy\hajndi-jms-ds.xml` to cause the `JMSProviderLoader` mbean to load TIBCO Enterprise Message Service:

```
<!-- The JMS provider loader -->
<mbean code="org.jboss.jms.jndi.JMSProviderLoader"
name=":service=JMSProviderLoader,name=TibjmsProvider">
<attribute name="ProviderName">TIBCOJMSProvider
</attribute>
<attribute name="ProviderAdapterClass">org.jboss.jms.jndi.JNDIProviderAdapter
</attribute>
<attribute name="QueueFactoryRef">XAQueueConnectionFactory
</attribute>
<attribute name="TopicFactoryRef">XATopicConnectionFactory
</attribute>
<attribute name="Properties">
java.naming.security.principal=ipcuser
java.naming.security.credentials=ipcuser
java.naming.factory.initial=com.tibco.tibjms.naming.TibjmsInitialContextFactory
java.naming.factory.url.pkgs=com.tibco.tibjms.naming
java.naming.provider.url=tibjmsnaming://host_name:port_number
</attribute>
</mbean>
```

4. Add the following lines in *JBOSS_HOME*`\server\`*server_type*`\deploy\hajndi-jms-ds.xml` to provide mapping between local JNDI names and those present in TIBCO EMS:

```
<!-- Redirect QueueConnectionFactory to TIBCO Enterprise Message Service -->
<mbean code="org.jboss.naming.NamingAlias"
name="DefaultDomain:service=NamingAlias,fromName=TIBCO.QueueConnectionFactory">
<attribute name="ToName">tibjmsnaming://host_name:port_number/XAQueueConnectionFactory
</attribute>
<attribute name="FromName">TIBCO.QueueConnectionFactory
</attribute>
</mbean>

<!-- Redirect TopicConnectionFactory to TIBCO Enterprise Message Service -->
<mbean code="org.jboss.naming.NamingAlias"
name="DefaultDomain:service=NamingAlias,fromName=TIBCO.TopicConnectionFactory">
<attribute name="ToName">tibjmsnaming://host_name:port_number/XATopicConnectionFactory
```

```
</attribute>
<attribute name="FromName">TIBCO.TopicConnectionFactory
</attribute>
</mbean>

<!-- Redirect COMRequestInjectorQueue to TIBCO Enterprise Message Service -->
<mbean code="org.jboss.naming.NamingAlias"
name="DefaultDomain:service=NamingAlias,fromName=queue/COMRequestInjectorQueue">
<attribute
name="ToName">tibjmsnaming://host_name:port_number/queue/COMRequestInjectorQueue
</attribute>
<attribute name="FromName">queue/COMRequestInjectorQueue
</attribute>
</mbean>

<!-- Redirect COMOrderRequestInjectorQueue to TIBCO Enterprise Message Service -->
<mbean code="org.jboss.naming.NamingAlias"
name="DefaultDomain:service=NamingAlias,fromName=queue/COMOrderRequestInjectorQueue
">
<attribute name="ToName">
tibjmsnaming://host_name:port_number/queue/COMOrderRequestInjectorQueue
</attribute>
<attribute name="FromName">queue/COMOrderRequestInjectorQueue
</attribute>
</mbean>

<!-- Redirect COMOrchExceptionQueue to TIBCO Enterprise Message Service -->
<mbean code="org.jboss.naming.NamingAlias"
name="DefaultDomain:service=NamingAlias,fromName=queue/COMOrchExceptionQueue">
<attribute
name="ToName">tibjmsnaming://host_name:port_number/queue/COMOrchExceptionQueue
</attribute>
<attribute name="FromName">queue/COMOrchExceptionQueue
</attribute>
</mbean>

<!-- Redirect COMOrderExceptionQueue to TIBCO Enterprise Message Service -->
<mbean code="org.jboss.naming.NamingAlias"
name="DefaultDomain:service=NamingAlias,fromName=queue/COMOrderExceptionQueue">
<attribute
name="ToName">tibjmsnaming://host_name:port_number/queue/COMOrderExceptionQueue
</attribute>
<attribute name="FromName">queue/COMOrderExceptionQueue
</attribute>
</mbean>

<!-- Redirect COMProcessQueue to TIBCO Enterprise Message Service -->
<mbean code="org.jboss.naming.NamingAlias"
name="DefaultDomain:service=NamingAlias,fromName=queue/COMProcessQueue">
<attribute name="ToName">tibjmsnaming://host_name:port_number/queue/COMProcessQueue
</attribute>
<attribute name="FromName">queue/COMProcessQueue
</attribute>
</mbean>

<!-- Redirect COMProcessQueueInner to TIBCO Enterprise Message Service -->
<mbean code="org.jboss.naming.NamingAlias"
```

```
name="DefaultDomain:service=NamingAlias,fromName=queue/COMProcessQueueInner ">
<attribute
name="ToName">tibjmsnaming://host_name:port_number/queue/COMProcessQueueInner
</attribute>
<attribute name="FromName">queue/COMProcessQueueInner
</attribute>
</mbean>

<!-- Redirect AOPDProcessQueue to TIBCO Enterprise Message Service -->
<mbean code="org.jboss.naming.NamingAlias"
name="DefaultDomain:service=NamingAlias,fromName=queue/AOPDProcessQueue">
<attribute name="ToName">tibjmsnaming://host_name:port_number/queue/AOPDProcessQueue
</attribute>
<attribute name="FromName">queue/AOPDProcessQueue
</attribute>
</mbean>

<!-- Redirect AOPDResponseQueue to TIBCO Enterprise Message Service -->
<mbean code="org.jboss.naming.NamingAlias"
name="DefaultDomain:service=NamingAlias,fromName=queue/AOPDResponseQueue">
<attribute name="ToName">tibjmsnaming://host_name:port_number/queue/AOPDResponseQueue
</attribute>
<attribute name="FromName">queue/AOPDResponseQueue
</attribute>
</mbean>

<!-- Redirect JMSFulfillOrderQueue to TIBCO Enterprise Message Service -->
<mbean code="org.jboss.naming.NamingAlias"
name="DefaultDomain:service=NamingAlias,fromName=queue/JMSFulfillOrderQueue">
<attribute
name="ToName">tibjmsnaming://host_name:port_number/queue/JMSFulfillOrderQueue
</attribute>
<attribute name="FromName">queue/JMSFulfillOrderQueue
</attribute>
</mbean>

<!-- Redirect IPCJeopardyTopic to TIBCO Enterprise Message Service -->
<mbean code="org.jboss.naming.NamingAlias"
name="DefaultDomain:service=NamingAlias,fromName=topic/IPCJeopardyTopic">
<attribute name="ToName">tibjmsnaming://host_name:port_number/topic/IPCJeopardyTopic
</attribute>
<attribute name="FromName">topic/IPCJeopardyTopic
</attribute>
</mbean>

<!-- Redirect COMOrchestratorTopic to TIBCO Enterprise Message Service -->
<mbean code="org.jboss.naming.NamingAlias"
name="DefaultDomain:service=NamingAlias,fromName=topic/COMOrchestratorTopic">
<attribute
name="ToName">tibjmsnaming://host_name:port_number/topic/COMOrchestratorTopic
</attribute>
<attribute name="FromName">topic/COMOrchestratorTopic
</attribute>
</mbean>

<!-- Redirect COMOrderManagerTopicto TIBCO Enterprise Message Service -->
<mbean code="org.jboss.naming.NamingAlias"
```

```
name="DefaultDomain:service=NamingAlias,fromName=topic/COMOrderManagerTopic">
<attribute
name="ToName">tibjmsnaming://host_name:port_number/topic/COMOrderManagerTopic
</attribute>
<attribute name="FromName">topic/COMOrderManagerTopic
</attribute>
</mbean>
```

5. When the JBoss server invokes JNDI and encounters the `tibjmsnaming` scheme, the server must be able to find the TIBCO Enterprise Message Service URLConnectionFactory. Therefore, modify the file *JBOSS_HOME*`\server\`*server_type*`\conf\jndi.properties` as follows:

   Change:

   ```
   java.naming.factory.url.pkgs=org.jboss.naming:org.jnp.
   interfaces
   ```

   To:

   ```
   java.naming.factory.url.pkgs=org.jboss.naming:org.jnp.
   interfaces:com.tibco.tibjms.naming
   ```

6. In the *JBOSS_HOME*`\server\`*server_type*`\conf\standardjboss.xml` file, navigate to the XML node:

   ```
   <invoker-proxy-binding>
   <name>message-driven-bean</name>
   ```

   And modify the following line. Change:

   ```
   <JMSProviderAdapterJNDI>DefaultJMSProvider
   </JMSProviderAdapterJNDI>
   ```

   To:

   ```
    <JMSProviderAdapterJNDI>TIBCOJMSProvider
   </JMSProviderAdapterJNDI>
   ```

   This change sets "TIBCOJMSProvider" as the JMS Provider Adapter JNDI name.

7. In the *JBOSS_HOME*`\server\`*server_type*`\conf\standardjboss.xml` file, navigate to the XML node:

   ```
   <invoker-proxy-binding>
   <name>message-driven-bean</name>
   ```

8. And modify the following line. Change:

   ```
   <DestinationQueue>queue/DLQ</DestinationQueue>
   ```

   To:

   ```
   <DestinationQueue>
   tibjmsnaming://host_name:port_number/queue/DLQ
   </DestinationQueue>
   ```

   This change specifies the TIBCO Enterprise Message Service JNDI name for the MDB Dead Letter Queue (DLQ) `queue/DLQ`.

9. Move the following files out of the *JBOSS_HOME*`\server\server_type\deploy` directory. These files are not needed when using TIBCO Enterprise Message Service and therefore must not be deployed:

   JBOSS_HOME`\server\`*server_type*`\deploy\jboss-messaging.sar\destinations-service.xml`

   JBOSS_HOME`\server\`*server_type*`\deploy\jboss-messaging.sar\COM-service.xml`

10. Copy `tibjms.jar` to the *JBOSS_HOME*`\server\`*server_type*`\lib` folder.

iProcess Conductor Configuration

11. Open *appFrameworks*/**`AppConfig.xml`**. Change the values of `QUEUE_CONNECTION_FACTORY` and `TOPIC_CONNECTION_FACTORY` to *TIBCO.QueueConnectionFactory* and *TIBCO.TopicConnectionFactory* respectively.

```
<setting>
<name>QUEUE_CONNECTION_FACTORY</name>
<value>TIBCO.QueueConnectionFactory</value>
</setting>
<setting>
<name>TOPIC_CONNECTION_FACTORY</name>
<value>TIBCO.TopicConnectionFactory</value>
</setting>
```

12. The of this file needs to be enabled if queues and topics are to be made secure. See Securing the iProcess Conductor JMS Gateway with EMS on page 146 for further details.

```
<!--==================== JMS settings (START)====================-->
<!-- Enable this section if an external JMS Server is used
<setting>
<name>INTERNAL_QUEUE_USER</name>
<value>ipcuser</value>
</setting>
<setting>
<name>INTERNAL_QUEUE_USER_PASSWORD</name>
<value>ipcuser</value>
</setting>
<setting>
<name>INTERNAL_TOPIC_USER</name>
<value>ipcuser</value>
</setting>
<setting>
<name>INTERNAL_TOPIC_USER_PASSWORD</name>
<value>ipcuser</value>
</setting>
-->
<!--====================  JMS settings (END)====================-->
```

EAI Orchestrator Configuration

13. To change the EAI Orchestration server plug-in to work with EMS, modify the `eaiorch.jndi.factory`, `eaiorch.jndi.url` and `eaiorch.jms.factory` values in *$SWDIR*`\eaijava\properties\eaiorch\eaiorch.properties` to specify TIBCO EMS values.

```
# TIBCO iProcess Conductor Orchestration Server Plug-in
                  eaiorch.jndi.factory=com.tibco.tibjms.naming.TibjmsInitialContextFactory
eaiorch.jndi.url=tibjmsnaming://host_name:port_number
eaiorch.jms.factory=XAQueueConnectionFactory
eaiorch.jms.principal=comEAIStepUser
eaiorch.jms.credentials=staffware
eaiorch.authconf.location=JBOSS_HOME/client/auth.conf
```

14. If it is not already present, copy `tibjms.jar` to the *$SWDIR*`\jmslib\ems` directory. `Tibjms.jar` can be found in the *EMS_HOME*`\ems5.0\lib` folder.

Restart

15. Restart iProcess Engine.
16. Restart JBoss.

Using iProcess Conductor under BEA WebLogic with EMS

You can configure an iProcess Conductor non-clustered installation working under any currently supported version of BEA WebLogic to work with EMS.

Carry out the following steps:

TIBCO EMS Configuration

1. Create the following queues, topics and ConnectionFactories in TIBCO EMS:

   — `factories.conf`:

   ```
   [XAQueueConnectionFactory]
   type = xaqueue
   url = tcp://port_number

   [XATopicConnectionFactory]
   type = xatopic
   url = tcp://port_number
   ```

   — `queues.conf`

   ```
   queue/COMOrchExceptionQueue
   queue/COMOrderExceptionQueue
   queue/COMRequestInjectorQueue secure
   queue/COMProcessQueue
   queue/COMOrderRequestInjectorQueue
   queue/AOPDProcessQueue
   queue/AOPDResponseQueue
   queue/COMProcessQueueInner
   ```

   queue/JMSFulFillOrderQueue

   — `topics.conf`

   ```
   topic/COMOrchestratorTopic
   topic/IPCJeopardyTopic
   topic/COMOrderManagerTopic
   ```

iProcess Conductor Configuration

2. Open *appFrameworks*/`AppConfig.xml`. Change the values of `QUEUE_CONNECTION_FACTORY` and `TOPIC_CONNECTION_FACTORY` to *TIBCO.QueueConnectionFactory* and *TIBCO.TopicConnectionFactory* respectively.

```
<setting>
<name>QUEUE_CONNECTION_FACTORY</name>
<value>TIBCO.QueueConnectionFactory</value>
```

```
</setting>
<setting>
<name>TOPIC_CONNECTION_FACTORY</name>
<value>TIBCO.TopicConnectionFactory</value>
</setting>
```

3. This step is optional. Enable the security section of this file only if you want to use iProcess Conductor with TIBCO EMS in the secure mode. See Securing the iProcess Conductor JMS Gateway with EMS on page 146 for further details.

```
<!--==================== JMS settings (START)====================-->
<!-- Enable this section if an external JMS Server is used
<setting>
<name>INTERNAL_QUEUE_USER</name>
<value>ipcuser</value>
</setting>
<setting>
<name>INTERNAL_QUEUE_USER_PASSWORD</name>
<value>ipcuser</value>
</setting>
<setting>
<name>INTERNAL_TOPIC_USER</name>
<value>ipcuser</value>
</setting>
<setting>
<name>INTERNAL_TOPIC_USER_PASSWORD</name>
<value>ipcuser</value>
</setting>
-->
<!--====================  JMS settings (END)====================-->
```

EAI Orchestrator Configuration

4. To change the EAI Orchestration server plug-in to work with EMS, modify the `eaiorch.jndi.factory`, `eaiorch.jms.factory` and `eaiorch.jndi.url` values in *$SWDIR*`\eaijava\properties\eaiorch\eaiorch.properties` to specify TIBCO EMS values as follows:

```
# TIBCO iProcess Conductor Orchestration Server Plug-in
                eaiorch.jndi.factory=com.tibco.tibjms.naming.TibjmsInitialContextFactory
eaiorch.jndi.url=tibjmsnaming://host_name:port_number
eaiorch.jms.factory=XAQueueConnectionFactory
eaiorch.jms.principal=comEAIStepUser
eaiorch.jms.credentials=staffware
eaiorch.authconf.location=
```

5. If it is not already present, copy `tibjms.jar` to the *$SWDIR*`\eaijava\libs\repository\user` directory. `Tibjms.jar` can be found in the *EMS_HOME*`\ems5.0\lib` folder.

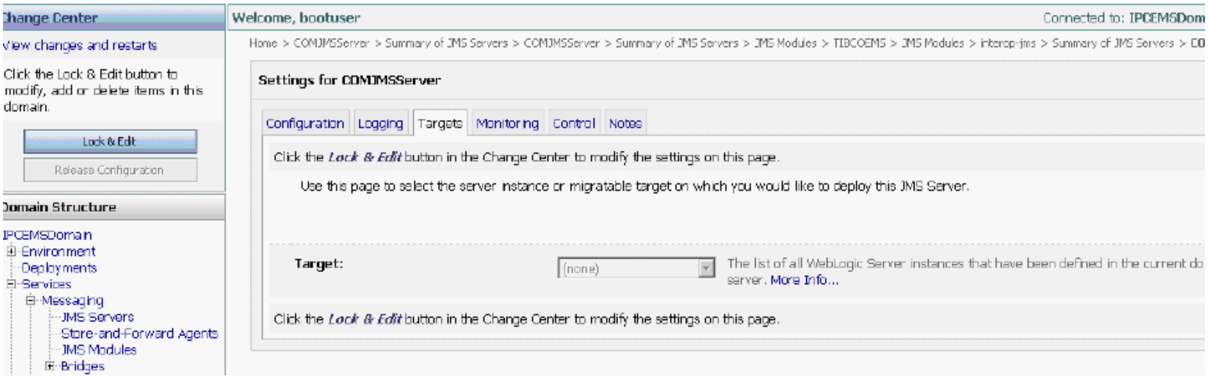
BEA WebLogic Configuration

6. Undeploy the existing JMS server. To do this:

   a. Using the WebLogic administration console, navigate to **Services > Messaging > JMS Servers > COMJMSServer**.

   b. Click the **Lock & Edit** button in the Change Center panel.

   c. Click the **Targets** tab, and in the Target field select **(none)** from the drop-down list. Then save.



Click the Lock & Edit button to modify, add or delete items in this domain.

Settings for COMJMSServer

Click the *Lock & Edit* button in the Change Center to modify the settings on this page.

Use this page to select the server instance or migratable target on which you would like to deploy this JMS Server.

Target: (none)

The list of all WebLogic Server instances that have been defined in the current do server.

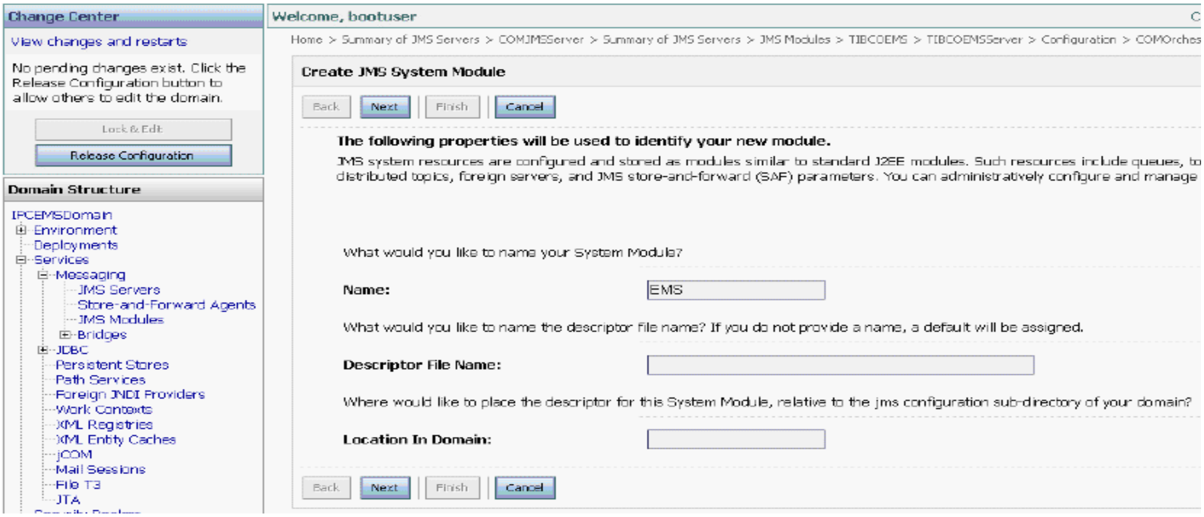
Click the *Lock & Edit* button in the Change Center to modify the settings on this page.

7. Create a new JMS Server module. To do this:

   a. Using the WebLogic administration console, navigate to **Services > Messaging > JMS Modules**.

   b. Click the **New** button to create a new JMS Module.

   c. Enter a meaningful name for the module (in the following illustration it is called `EMS`).



No pending changes exist. Click the Release Configuration button to allow others to edit the domain.

Create JMS System Module

The following properties will be used to identify your new module.

JMS system resources are configured and stored as modules similar to standard J2EE modules. Such resources include queues, to distributed topics, foreign servers, and JMS store-and-forward (SAF) parameters. You can administratively configure and manage

What would you like to name your System Module?

Name: EMS

What would you like to name the descriptor file name? If you do not provide a name, a default will be assigned.

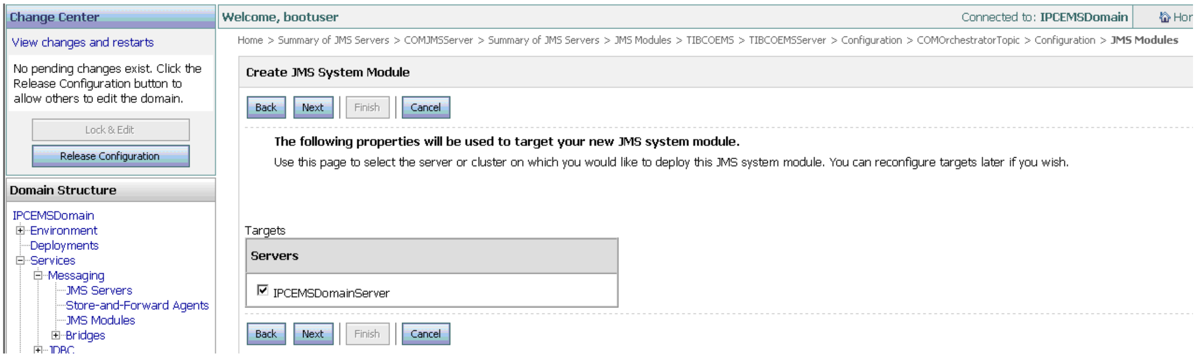
Descriptor File Name:

Where would like to place the descriptor for this System Module, relative to the jms configuration sub-directory of your domain?

Location In Domain:

d. Click the **Next** button.

e. Choose the server to which the new module needs to be deployed.


Change Center
View changes and restarts
Lock & Edit
Release Configuration
Domain Structure
IPCEMSDomain
Environment
Deployments
Services
Messaging
JMS Servers
Store-and-Forward Agents
JMS Modules
Bridges
JDBC
Welcome, bootuser
Connected to: IPCEMSDomain
Hor
Home > Summary of JMS Servers > COMJMSServer > Summary of JMS Servers > JMS Modules > TIBCOEMS > TIBCOEMSServer > Configuration > COMOrchestratorTopic > Configuration > JMS Modules
Back
Next
Finish
Cancel

No pending changes exist. Click the Release Configuration button to allow others to edit the domain.

Create JMS System Module

The following properties will be used to target your new JMS system module.

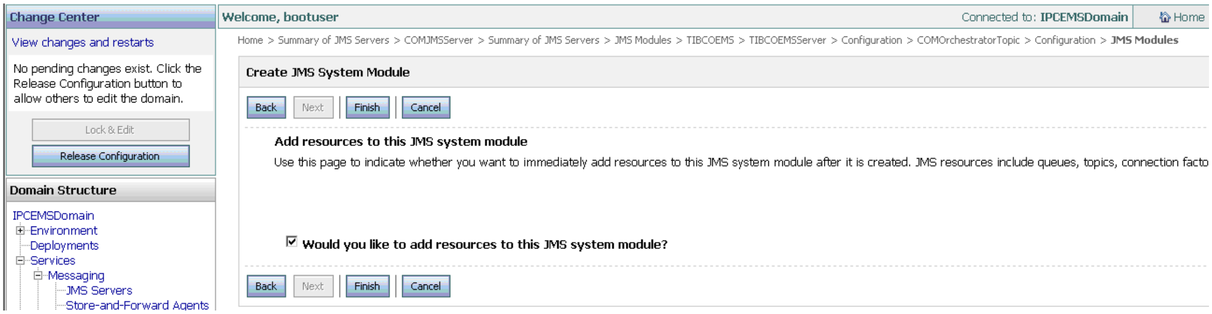
Use this page to select the server or cluster on which you would like to deploy this JMS system module. You can reconfigure targets later if you wish.

Targets

Servers
☑ IPCEMSDomainServer

f. Click the **Next** button.

g. Check the **Would you like to add resources to this JMS System module?** checkbox.



No pending changes exist. Click the Release Configuration button to allow others to edit the domain.

Home > Summary of JMS Servers > COMJMSServer > Summary of JMS Servers > JMS Modules > TIBCOEMS > TIBCOEMSServer > Configuration > COMOrchestratorTopic > Configuration > **JMS Modules**

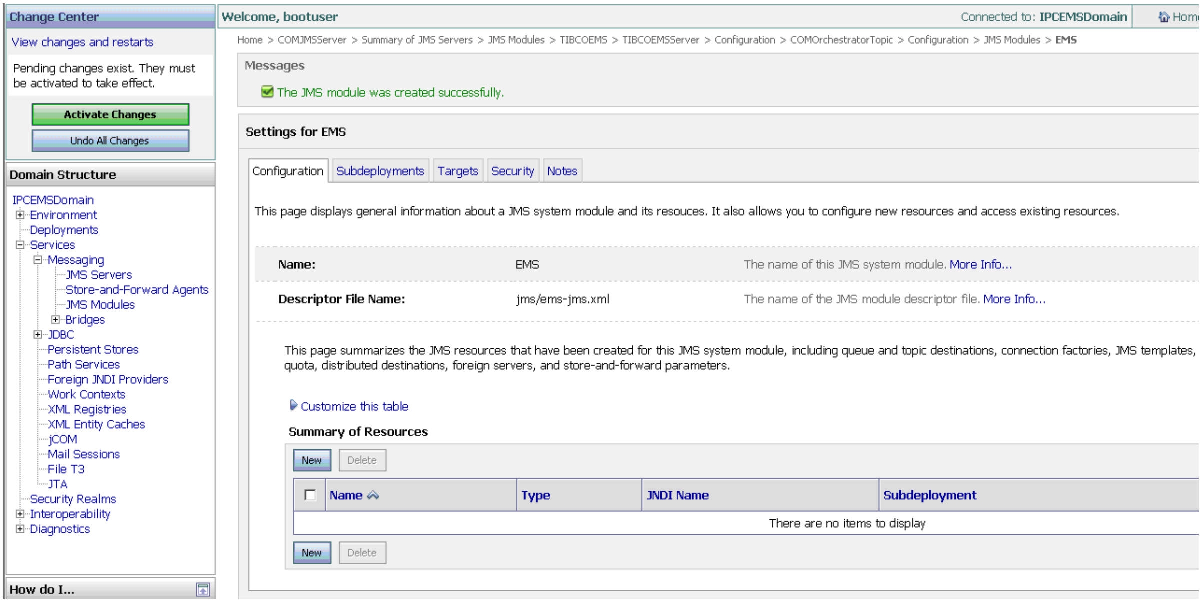
Create JMS System Module

Add resources to this JMS system module

Use this page to indicate whether you want to immediately add resources to this JMS system module after it is created. JMS resources include queues, topics, connection facto

☑ **Would you like to add resources to this JMS system module?**

h. Click the **Finish** button. The JMS module is successfully created.



Pending changes exist. They must be activated to take effect.

Home > COMJMSServer > Summary of JMS Servers > JMS Modules > TIBCOEMS > TIBCOEMSServer > Configuration > COMOrchestratorTopic > Configuration > JMS Modules > **EMS**

Messages

✓ The JMS module was created successfully.

Settings for EMS

This page displays general information about a JMS system module and its resouces. It also allows you to configure new resources and access existing resources.

Name:	EMS	The name of this JMS system module. More Info...
Descriptor File Name:	jms/ems-jms.xml	The name of the JMS module descriptor file. More Info...

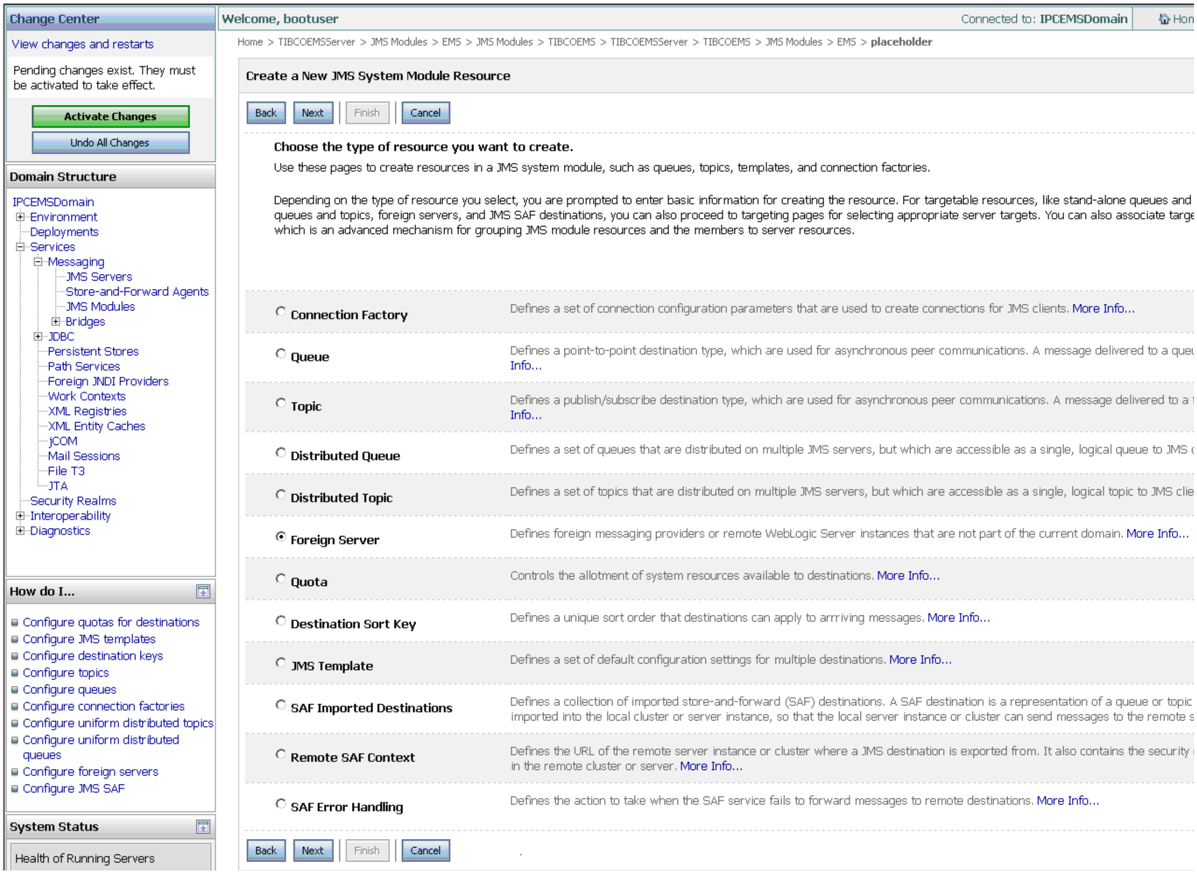
This page summarizes the JMS resources that have been created for this JMS system module, including queue and topic destinations, connection factories, JMS templates, quota, distributed destinations, foreign servers, and store-and-forward parameters.

Summary of Resources

□	Name	Type	JNDI Name	Subdeployment
	There are no items to display			

i. Click the **New** button to add a new resource. This will be a foreign JMS Server.

j. Select the **Foreign Server** radio button from the list displayed, and click the **Next** button.



Change Center

View changes and restarts

Pending changes exist. They must be activated to take effect.

Domain Structure

- IPCEMSDomain
  - Environment
  - Deployments
  - Services
    - Messaging
      - JMS Servers
      - Store-and-Forward Agents
      - JMS Modules
      - Bridges
    - JDBC
    - Persistent Stores
    - Path Services
    - Foreign JNDI Providers
    - Work Contexts
    - XML Registries
    - XML Entity Caches
    - jCOM
    - Mail Sessions
    - File T3
    - JTA
  - Security Realms
  - Interoperability
  - Diagnostics

How do I...

- Configure quotas for destinations
- Configure JMS templates
- Configure destination keys
- Configure topics
- Configure queues
- Configure connection factories
- Configure uniform distributed topics
- Configure uniform distributed queues
- Configure foreign servers
- Configure JMS SAF

System Status

Health of Running Servers

Welcome, bootuser

Connected to: IPCEMSDomain

Home > TIBCOEMSServer > JMS Modules > EMS > JMS Modules > TIBCOEMS > TIBCOEMSServer > TIBCOEMS > JMS Modules > EMS > **placeholder**

Create a New JMS System Module Resource

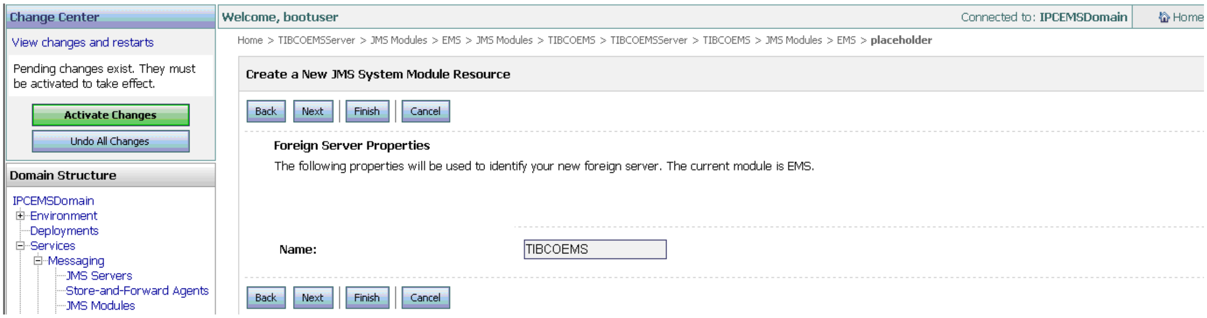
Choose the type of resource you want to create.

Use these pages to create resources in a JMS system module, such as queues, topics, templates, and connection factories.

Depending on the type of resource you select, you are prompted to enter basic information for creating the resource. For targetable resources, like stand-alone queues and queues and topics, foreign servers, and JMS SAF destinations, you can also proceed to targeting pages for selecting appropriate server targets. You can also associate targe which is an advanced mechanism for grouping JMS module resources and the members to server resources.

○ Connection Factory	Defines a set of connection configuration parameters that are used to create connections for JMS clients. More Info...
○ Queue	Defines a point-to-point destination type, which are used for asynchronous peer communications. A message delivered to a queu Info...
○ Topic	Defines a publish/subscribe destination type, which are used for asynchronous peer communications. A message delivered to a t Info...
○ Distributed Queue	Defines a set of queues that are distributed on multiple JMS servers, but which are accessible as a single, logical queue to JMS c
○ Distributed Topic	Defines a set of topics that are distributed on multiple JMS servers, but which are accessible as a single, logical topic to JMS clie
● Foreign Server	Defines foreign messaging providers or remote WebLogic Server instances that are not part of the current domain. More Info...
○ Quota	Controls the allotment of system resources available to destinations. More Info...
○ Destination Sort Key	Defines a unique sort order that destinations can apply to arrriving messages. More Info...
○ JMS Template	Defines a set of default configuration settings for multiple destinations. More Info...
○ SAF Imported Destinations	Defines a collection of imported store-and-forward (SAF) destinations. A SAF destination is a representation of a queue or topic imported into the local cluster or server instance, so that the local server instance or cluster can send messages to the remote s
○ Remote SAF Context	Defines the URL of the remote server instance or cluster where a JMS destination is exported from. It also contains the security in the remote cluster or server. More Info...
○ SAF Error Handling	Defines the action to take when the SAF service fails to forward messages to remote destinations. More Info...

k. Provide an appropriate name for the server. In this example, it is `TIBCOEMS`.



Pending changes exist. They must be activated to take effect.

Create a New JMS System Module Resource

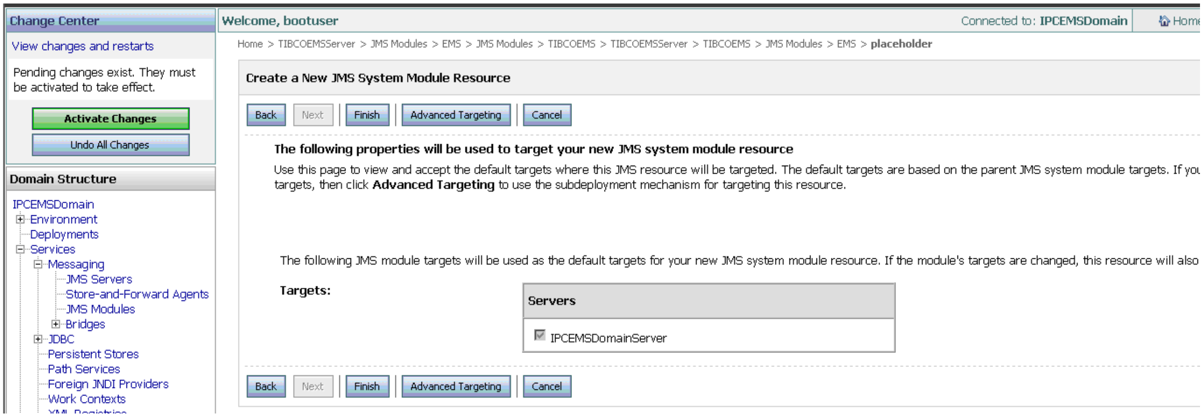
Foreign Server Properties

The following properties will be used to identify your new foreign server. The current module is EMS.

Name: TIBCOEMS

l. Click the **Next** button.

m. Default targets are displayed. Click the **Finish** button to accept them.



Pending changes exist. They must be activated to take effect.

Create a New JMS System Module Resource

The following properties will be used to target your new JMS system module resource

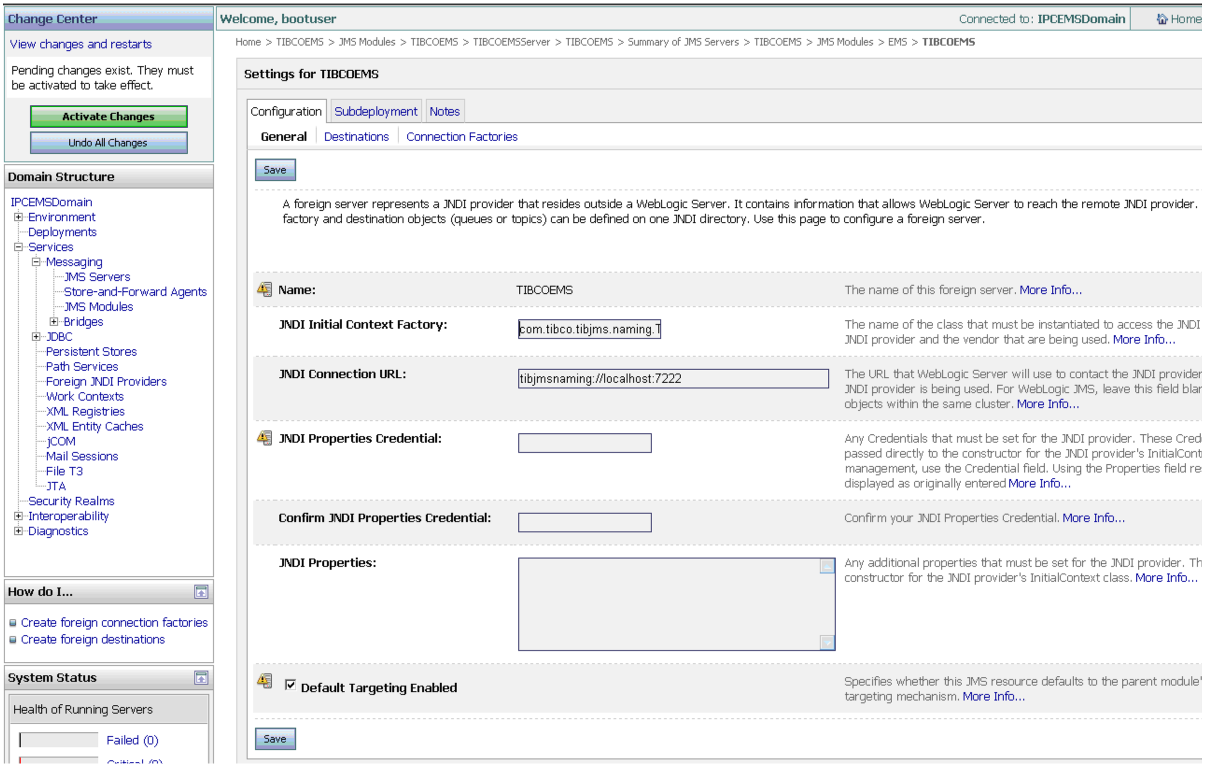
Use this page to view and accept the default targets where this JMS resource will be targeted. The default targets are based on the parent JMS system module targets. If you targets, then click **Advanced Targeting** to use the subdeployment mechanism for targeting this resource.

The following JMS module targets will be used as the default targets for your new JMS system module resource. If the module's targets are changed, this resource will also

Targets:

Servers
☑ IPCEMSDomainServer

n. Click the name of the newly-added server. The following Settings dialog appears.



Change Center

View changes and restarts

Pending changes exist. They must be activated to take effect.

Activate Changes

Undo All Changes

Domain Structure

IPCEMSDomain
⊞ Environment
Deployments
⊟ Services
⊟ Messaging
JMS Servers
Store-and-Forward Agents
JMS Modules
⊞ Bridges
⊞ JDBC
Persistent Stores
Path Services
Foreign JNDI Providers
Work Contexts
XML Registries
XML Entity Caches
jCOM
Mail Sessions
File T3
JTA
Security Realms
⊞ Interoperability
⊞ Diagnostics

How do I...

- Create foreign connection factories
- Create foreign destinations

System Status

Health of Running Servers

Failed (0)

Welcome, bootuser

Connected to: IPCEMSDomain

Home

Home > TIBCOEMS > JMS Modules > TIBCOEMS > TIBCOEMSServer > TIBCOEMS > Summary of JMS Servers > TIBCOEMS > JMS Modules > EMS > **TIBCOEMS**

Settings for TIBCOEMS

Configuration | Subdeployment | Notes

General | Destinations | Connection Factories

Save

A foreign server represents a JNDI provider that resides outside a WebLogic Server. It contains information that allows WebLogic Server to reach the remote JNDI provider. factory and destination objects (queues or topics) can be defined on one JNDI directory. Use this page to configure a foreign server.

Name:	TIBCOEMS	The name of this foreign server. More Info...
JNDI Initial Context Factory:	com.tibco.tibjms.naming.T	The name of the class that must be instantiated to access the JNDI JNDI provider and the vendor that are being used. More Info...
JNDI Connection URL:	tibjmsnaming://localhost:7222	The URL that WebLogic Server will use to contact the JNDI provider JNDI provider is being used. For WebLogic JMS, leave this field blar objects within the same cluster. More Info...
JNDI Properties Credential:		Any Credentials that must be set for the JNDI provider. These Cred passed directly to the constructor for the JNDI provider's InitialCont management, use the Credential field. Using the Properties field re displayed as originally entered More Info...
Confirm JNDI Properties Credential:		Confirm your JNDI Properties Credential. More Info...
JNDI Properties:		Any additional properties that must be set for the JNDI provider. Th constructor for the JNDI provider's InitialContext class. More Info...
☑ **Default Targeting Enabled**		Specifies whether this JMS resource defaults to the parent module' targeting mechanism. More Info...

Save

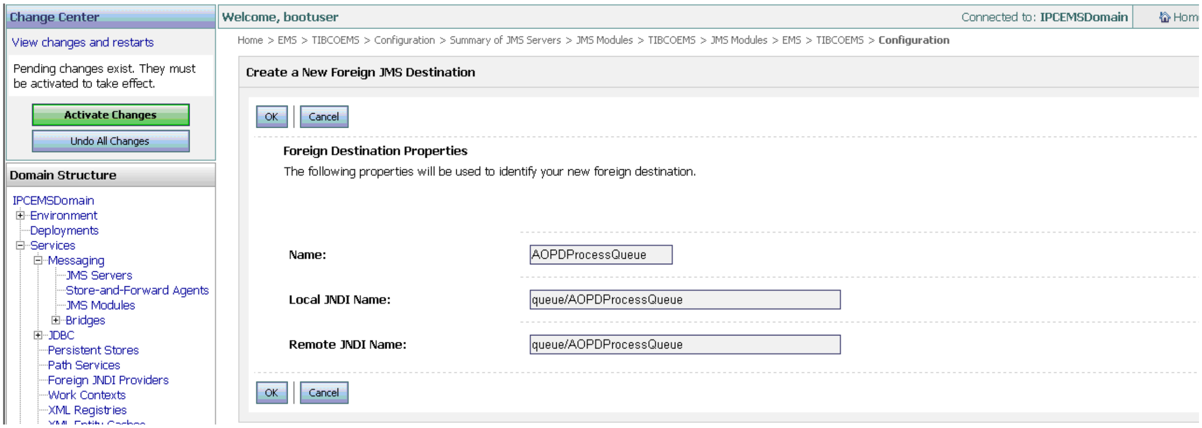
o. Provide appropriate values for the JNDI Initial Context Factory and JNDI Connection URL fields. Typical values are:

JNDI Initial Context Factory:
`com.tibco.tibjms.naming.TibjmsInitialContextFactory`

JNDI Connection URL: tibjmsnaming://*host_name*:*`port_number`*

p. Click the **Save** button.

q. Click the **Destinations** tab to create foreign destinations.



Create a New Foreign JMS Destination

Foreign Destination Properties

The following properties will be used to identify your new foreign destination.

Name: AOPDProcessQueue

Local JNDI Name: queue/AOPDProcessQueue

Remote JNDI Name: queue/AOPDProcessQueue

r. Create the following foreign JMS destinations:

Queues

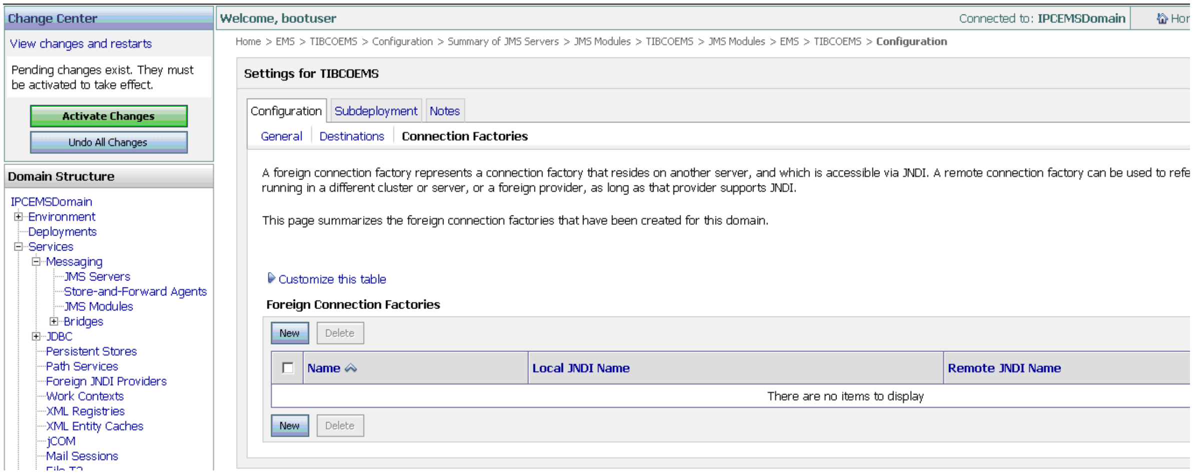
Name	Local JNDI Name	Remote JNDI Name
COMOrchExceptionQueue	queue/COMOrchExceptionQueue	queue/COMOrchExceptionQueue
COMOrderExceptionQueue	queue/COMOrderExceptionQueue	queue/COMOrderExceptionQueue
COMOrderRequestInjectorQueue	queue/COMOrderRequestInjectorQueue	queue/COMOrderRequestInjectorQueue
COMProcessQueue	queue/COMProcessQueue	queue/COMProcessQueue
COMRequestInjectorQueue	queue/COMRequestInjectorQueue	queue/COMRequestInjectorQueue
AOPDProcessQueue	queue/AOPDProcessQueue	queue/AOPDProcessQueue

Name	Local JNDI Name	Remote JNDI Name
AOPDResponseQueue	queue/AOPDResponseQueue	queue/AOPDResponseQueue
COMProcessQueueInner	queue/COMProcessQueueInner	queue/COMProcessQueueInner
JMSFulFillOrderQueue	queue/JMSFulFillOrderQueue	queue/JMSFulFillOrderQueue

Topics

Name	Local JNDI Name	Remote JNDI Name
IPCJeopardyTopic	topic/IPCJeopardyTopic	topic/IPCJeopardyTopic
COMOrchestratorTopic	topic/COMOrchestratorTopic	topic/COMOrchestratorTopic
COMOrderManagerTopic	topic/COMOrderManagerTopic	topic/COMOrderManagerTopic

s. Navigate to the Connection Factories tab so that Queue and Topic Connection Factories can be created.



A foreign connection factory represents a connection factory that resides on another server, and which is accessible via JNDI. A remote connection factory can be used to refe
running in a different cluster or server, or a foreign provider, as long as that provider supports JNDI.

This page summarizes the foreign connection factories that have been created for this domain.

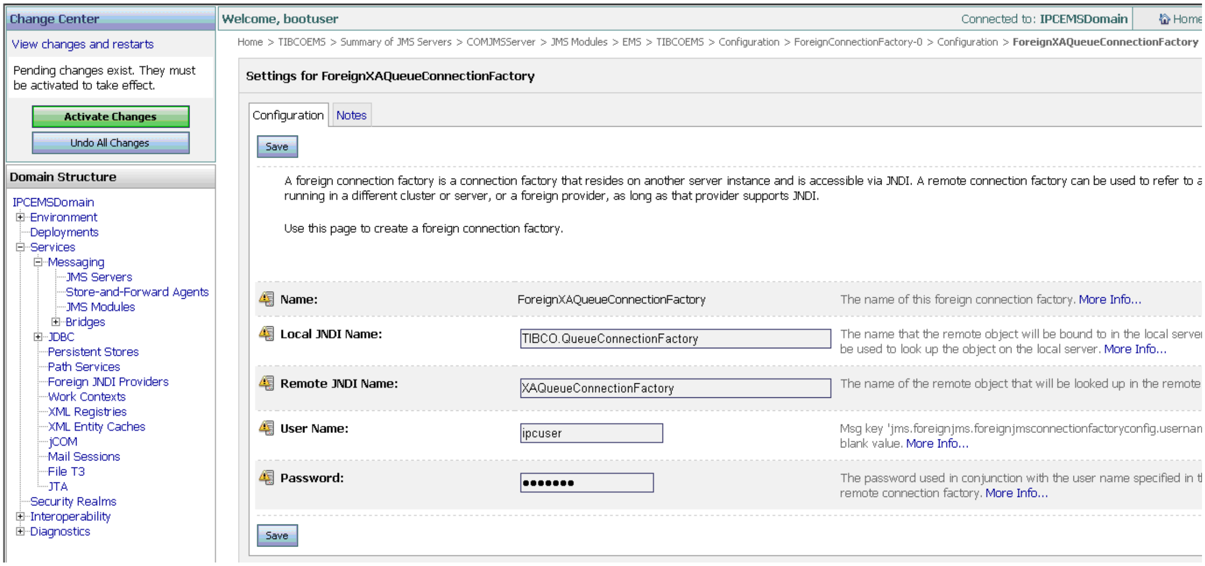
Foreign Connection Factories

□	Name	Local JNDI Name	Remote JNDI Name
	There are no items to display		

t. Create the following connection factories:

Name	Local JNDI Name	Remote JNDI Name
ForeignXAQueueConnectionFactory	TIBCO.QueueConnectionFactory	XAQueueConnectionFactory
ForeignXATopicConnectionFactory	TIBCO.TopicConnectionFactory	XATopicConnectionFactory

u. This step is optional; do this only if you want to use iProcess Conductor with TIBCO EMS in the secure mode. If so, then click on each of the COMConnectionFactories to specify `ipcuser` as the user name and provide its credentials for each of the ConnectionFactories created. See also Securing the iProcess Conductor JMS Gateway with EMS on page 146.



Pending changes exist. They must be activated to take effect.

Settings for ForeignXAQueueConnectionFactory

A foreign connection factory is a connection factory that resides on another server instance and is accessible via JNDI. A remote connection factory can be used to refer to a running in a different cluster or server, or a foreign provider, as long as that provider supports JNDI.

Use this page to create a foreign connection factory.

Name:	ForeignXAQueueConnectionFactory	The name of this foreign connection factory. More Info...
Local JNDI Name:	TIBCO.QueueConnectionFactory	The name that the remote object will be bound to in the local server be used to look up the object on the local server. More Info...
Remote JNDI Name:	XAQueueConnectionFactory	The name of the remote object that will be looked up in the remote
User Name:	ipcuser	Msg key 'jms.foreignjms.foreignjmsconnectionfactoryconfig.usernan blank value. More Info...
Password:	•••••••	The password used in conjunction with the user name specified in t remote connection factory. More Info...

The password that you specify here for `ipcuser` must be the same as that given when you created `ipcuser` in EMS - see step 2 on page 146. The same credentials are also mentioned in the `AppConfig.xml` file. By default, the password for `ipcuser` is '`ipcuser`', but you can change this when creating the user.

v. Click the **Activate Changes** button in the Change Center panel so that the changes can be applied.

8. Shut down both WebLogic and iProcess Engine.

9. Copy `tibjms.jar` to the *BEA_HOME*`\user_projects\domains\`*domain*`\lib` directory, where *domain* is the name of the domain where the EMS application is installed.

10. Modify `META-Inf/weblogic-ejb-jar.xml` in the `ORCH-ejb-2.0`, `ORDER-ejb-2.0 EXCEPTION-ejb-2.0` and `OPD-ejb-2.0` files with ConnectionFactory settings as in the following examples. These JAR files can be found in

BEA_HOME\user_projects\domains\domain\applications\COM-ORCH-2.0.ear.

In the examples the text in **bold** is the newly added xml fragment:

— ORCH-ejb:

```
<weblogic-enterprise-bean>
      <ejb-name>com/RequestInjectorMessageBean</ejb-name>
      <message-driven-descriptor>
         <pool>
            <max-beans-in-free-pool>1</max-beans-in-free-pool>
            <initial-beans-in-free-pool>1</initial-beans-in-free-pool>
         </pool>

<destination-jndi-name>queue/COMRequestInjectorQueue</destination-jndi-name>
 <connection-factory-jndi-name>TIBCO.QueueConnectionFactory</connection-factory-jndi-name>
      </message-driven-descriptor>
      <reference-descriptor>
      </reference-descriptor>
      <enable-call-by-reference>True</enable-call-by-reference>

<run-as-identity-principal>RequestInjectorMessageBeanUser</run-as-identity-principa
l>

<weblogic-enterprise-bean>
      <ejb-name>com/OrchestratorMessageBean</ejb-name>
      <message-driven-descriptor>
         <pool>
            <max-beans-in-free-pool>5</max-beans-in-free-pool>
            <initial-beans-in-free-pool>5</initial-beans-in-free-pool>
         </pool>
         <destination-jndi-name>queue/COMProcessQueue</destination-jndi-name>
 <connection-factory-jndi-name>TIBCO.QueueConnectionFactory</connection-factory-jndi-name>
      </message-driven-descriptor>
      <reference-descriptor>
      </reference-descriptor>
      <enable-call-by-reference>True</enable-call-by-reference>
<run-as-identity-principal>ProcessInjectorMessageBeanUser</run-as-identity-principa
l>

   </weblogic-enterprise-bean>

<weblogic-enterprise-bean>
      <ejb-name>com/PlanStateListener</ejb-name>
      <message-driven-descriptor>
         <pool>
            <initial-beans-in-free-pool>1</initial-beans-in-free-pool>
         </pool>
         <destination-jndi-name>topic/COMOrchestratorTopic</destination-jndi-name>
<connection-factory-jndi-name>TIBCO.TopicConnectionFactory</connection-factory-jndi-name>
</message-driven-descriptor>
      <reference-descriptor>
      </reference-descriptor>
      <enable-call-by-reference>True</enable-call-by-reference>
```

```
<run-as-identity-principal>ProcessInjectorMessageBeanUser</run-as-identity-principa
l>
 </weblogic-enterprise-bean>

   </weblogic-enterprise-bean>
```

— ORDER-ejb

```
   <weblogic-enterprise-bean>
      <ejb-name>com/OrderOrchestratorListenerMessageBean</ejb-name>
      <message-driven-descriptor>
         <pool>
            <max-beans-in-free-pool>1</max-beans-in-free-pool>
            <initial-beans-in-free-pool>1</initial-beans-in-free-pool>
         </pool>
         <destination-jndi-name>topic/COMOrchestratorTopic</destination-jndi-name>
 <connection-factory-jndi-name>TIBCO.TopicConnectionFactory</connection-factory-jndi-name>
      </message-driven-descriptor>
      <reference-descriptor>
      </reference-descriptor>
      <enable-call-by-reference>True</enable-call-by-reference>
        <run-as-identity-principal>OrderRequestInjectorMessageBeanUser</run-as-
identity-principal>

   </weblogic-enterprise-bean>

   <weblogic-enterprise-bean>
      <ejb-name>com/OrderRequestInjectorMessageBean</ejb-name>
      <message-driven-descriptor>
         <pool>
            <initial-beans-in-free-pool>25</initial-beans-in-free-pool>
         </pool>
         <destination-jndi-name>queue/COMOrderRequestInjectorQueue</destination
-jndi-name>
 <connection-factory-jndi-name>TIBCO.QueueConnectionFactory</connection-factory-jndi-name>
      </message-driven-descriptor>
      <reference-descriptor>
      </reference-descriptor>
      <enable-call-by-reference>True</enable-call-by-reference>
        <run-as-identity-principal>OrderRequestInjectorMessageBeanUser</run-as-
identity-principal>

   </weblogic-enterprise-bean>
```

— EXCEPTION-ejb

```
<weblogic-enterprise-bean>
      <ejb-name>com/OrchestratorExceptionMessageBean</ejb-name>
      <message-driven-descriptor>
         <pool>
            <max-beans-in-free-pool>1</max-beans-in-free-pool>
            <initial-beans-in-free-pool>1</initial-beans-in-free-pool>
         </pool>
         <destination-jndi-name>queue/COMOrchExceptionQueue</destination-jndi-name>
```

```
<connection-factory-jndi-name>TIBCO.QueueConnectionFactory</connection-factory-jndi-name>
      </message-driven-descriptor>
      <reference-descriptor>
      </reference-descriptor>
      <enable-call-by-reference>True</enable-call-by-reference>

<run-as-identity-principal>RequestInjectorMessageBeanUser</run-as-identity-principa
l>

   </weblogic-enterprise-bean>
   <weblogic-enterprise-bean>
      <ejb-name>com/ExceptionEventMessageBean</ejb-name>
      <message-driven-descriptor>
         <pool>
            <max-beans-in-free-pool>1</max-beans-in-free-pool>
            <initial-beans-in-free-pool>1</initial-beans-in-free-pool>
         </pool>
         <destination-jndi-name>topic/COMOrchestratorTopic</destination-jndi-name>
<connection-factory-jndi-name>TIBCO.TopicConnectionFactory</connection-factory-jndi-name>
      </message-driven-descriptor>
      <reference-descriptor>
      </reference-descriptor>
      <enable-call-by-reference>True</enable-call-by-reference>

<run-as-identity-principal>RequestInjectorMessageBeanUser</run-as-identity-principa
l>

   </weblogic-enterprise-bean>
```

— OPD-ejb

```
<weblogic-enterprise-bean>
      <ejb-name>com/OPD-ejb</ejb-name>
      <message-driven-descriptor>
         <pool>
            <max-beans-in-free-pool>1</max-beans-in-free-pool>
            <initial-beans-in-free-pool>1</initial-beans-in-free-pool>
         </pool>
         <destination-jndi-name>queue/AOPDProcessQueue</destination-jndi-name>
<connection-factory-jndi-name>TIBCO.QueueConnectionFactory</connection-factory-jndi-name>
      </message-driven-descriptor>
      <reference-descriptor>
      </reference-descriptor>
      <enable-call-by-reference>True</enable-call-by-reference>

   </weblogic-enterprise-bean>
```

11. Restart iProcess Engine.

12. Restart WebLogic.

Securing the iProcess Conductor JMS Gateway with EMS

You can decide to make the JMS Gateway used with iProcess Conductor secure. To do this, you need to carry out the following steps:

1. Set authorization enabled. For example:

   ```
   set server authorization=enabled
   ```

2. Create the following users:

   a. `anonymous` - the J2EE container tries to log in as this user at startup.

   b. `comEAIStepUser` - the EAI Orchestrator step will assume identity as this user when it sends messages. Set this user's password as 'staffware'.

   c. `ipcuser` - the iProcess Conductor internal user. Set this user's password as 'ipcuser'.

   d. `requestInjector` - EMS Clients can assume the identity of this user when it injects messages into iProcess Conductor. This name can be customized for a particular deployment. This user needs to be granted access rights for sending messages to `queue/COMRequestInjectorQueue` and `queue/COMOrderRequestInjectorQueue`. The password can be anything. The same credentials must be used by external clients for injecting EMS messages.

   You can decide not to create the `requestInjector` user and its associated grants, and instead create a user of your choice who will have rights to inject orders into the system.

   e. `RequestInjectorMessageBeanUser` - an internal message bean user. Set this user's password as 'RequestInjectorMessageBeanUser'.

   f. `OrderRequestInjectorMessageBeanUser` - an internal message bean user. Set this user's password as 'OrderRequestInjectorMessageBeanUser'.

   g. `ProcessInjectorMessageBeanUser` - an internal message bean user. Set user's password as 'ProcessInjectorMessageBeanUser'.

   h. `OrderOrchestratorListenerMessageBeanUser` - set this user's password as 'OrderOrchestratorListenerMessageBeanUser'.

   i. `InternalUser` - an internal message bean user. Set this user's password as 'InternalPassword'.

   For example:

```
create user anonymous
create user comEAIStepUser password=staffware
create user ipcuser password=ipcuser
```

```
create user requestInjector password=requestInjector
create user RequestInjectorMessageBeanUser
password=RequestInjectorMessageBeanUser
create user OrderRequestInjectorMessageBeanUser
password=OrderRequestInjectorMessageBeanUser
create user ProcessInjectorMessageBeanUser
password=ProcessInjectorMessageBeanUser
create user OrderOrchestratorListenerMessageBeanUser
password=OrderOrchestratorListenerMessageBeanUser
create user InternalUser password=InternalPassword
```

3. Secure the following queues:

   `queue/COMOrchExceptionQueue`

   `queue/COMOrderExceptionQueue`

   `queue/COMOrderRequestInjectorQueue`

   `queue/COMProcessQueue`

   `queue/COMRequestInjectorQueue`

   `queue/AOPDProcessQueue`

   `queue/AOPDResponseQueue`

   `queue/COMProcessQueueInner`

   `queue/JMSFulFillOrderQueue`

   For example:

```
addprop queue queue/COMOrchExceptionQueue secure
addprop queue queue/COMOrderExceptionQueue secure
addprop queue queue/COMOrderRequestInjectorQueue secure
addprop queue queue/COMProcessQueue secure
addprop queue queue/COMRequestInjectorQueue secure
addprop queue queue/AOPDProcessQueue secure
addprop queue queue/AOPDResponseQueue secure
addprop queue queue/COMProcessQueueInner secure
addprop queue queue/JMSFulFillOrderQueue
```

4. Secure the following topics:

   `topic/COMOrchestratorTopic`

   `topic/COMOrderManagerTopic`

   `topic/IPCJeopardyTopic`

   For example:

```
addprop topic topic/COMOrchestratorTopic secure
addprop topic topic/COMOrderManagerTopic secure
addprop topic topic/IPCJeopardyTopic secure
```

5. Grant user access.

   You need the following queue level grants:

```
grant queue queue/COMOrchExceptionQueue ipcuser send,receive
grant queue queue/COMOrderExceptionQueue ipcuser send,receive
grant queue queue/COMProcessQueue ipcuser send,receive
grant queue queue/COMRequestInjectorQueue ipcuser send,receive
grant queue queue/COMOrderRequestInjectorQueue ipcuser send,receive
grant queue queue/AOPDProcessQueue  ipcuser receive
grant queue queue/AOPDResponseQueue  ipcuser send,receive
grant queue queue/COMProcessQueueInner ipcuser send,receive
grant queue queue/JMSFulFillOrderQueue ipcuser send,receive

grant queue queue/COMRequestInjectorQueue requestInjector send
grant queue queue/COMOrderRequestInjectorQueue requestInjector send
grant queue queue/COMOrderRequestInjectorQueue OrderRequestInjectorMessageBeanUser
receive
grant queue queue/COMRequestInjectorQueue RequestInjectorMessageBeanUser receive
grant queue queue/COMProcessQueue comEAIStepUser send
grant queue queue/COMProcessQueue ProcessInjectorMessageBeanUser receive
grant queue queue/COMOrchExceptionQueue RequestInjectorMessageBeanUser receive
grant queue queue/AOPDProcessQueue  InternalUser receive
grant queue queue/AOPDResponseQueue  InternalUser send,receive
```

You need the following topic level grants:

```
grant topic topic/IPCJeopardyTopic ipcuser subscribe,publish
grant topic topic/COMOrderManagerTopic ipcuser subscribe,publish
grant topic topic/COMOrchestratorTopic ipcuser subscribe,publish,durable
grant topic topic/COMOrchestratorTopic RequestInjectorMessageBeanUser
subscribe,durable
grant topic topic/COMOrchestratorTopic OrderRequestInjectorMessageBeanUser
subscribe,durable
grant topic topic/COMOrchestratorTopic OrderOrchestratorListenerMessageBeanUser
subscribe
```

Additional Customization

The access rights described in the previous section are required by iProcess Conductor. You may choose to create additional users and grant rights to them so that orders can be injected into the system.

For example, if userX needs to perform order injection, then the following steps need to be performed:

1. Create userX.
2. Provide userX with `send` rights to `queue/COMOrderRequestInjectorQueue`.

As another example, if userY needs to perform Automatic Order Plan Development (AOPD), then the following steps need to be performed:

1. Create userY.
2. Provide userY with `send` rights to `queue/AOPDProcessQueue`.
3. Provide userY with `receive` rights to `queue/AOPDResponseQueue`.

Changing Passwords

If you need to set or to change the passwords of any of the users listed in step 2, you need to ensure that the changed password is copied to the locations given in this section.

Under WebLogic

When iProcess Conductor is used under WebLogic, it requires the following username:

- `ipcuser`: if you change the password for `ipcuser`, you must:
  - — Revise the values of the INTERNAL_QUEUE_USER_PASSWORD and INTERNAL_TOPIC_USER_PASSWORD properties in the `AppConfig.xml` file.
  - — Log in to WebLogic Console, navigate to the foreign JMS Server definition EMS, and revise the `ipcuser` password for its connection factories.

Under JBoss

When iProcess Conductor is used under JBoss, it requires the following usernames:

- `ipcuser`: if you change the password for `ipcuser`, you must:
  - — Revise the values of the INTERNAL_QUEUE_USER_PASSWORD and INTERNAL_TOPIC_USER_PASSWORD properties in the `AppConfig.xml` file.
  - — Update the `ipcuser` password in *JBOSS_HOME*`\server\default\deploy\jms\jms-ds.xml`.
- `RequestInjectorMessageBeanUser`, `OrderRequestInjectorMessageBeanUser`, `ProcessInjectorMessageBeanUser`, `InternalUser`: if you change the password for iProcess Conductor Orchestration Plug-in and iProcess Conductor Orchestration Plug-in, any of these users, you must:

— Revise the value of `mdb-passwd` in the `jboss.xml` file that is located in the `ORDER-ejb-2.0.jar`, `EXCEPTION-ejb-2.0.jar`, `OPD-ejb-2.0.jar` and `ORCH-ejb-2.0.jar` files.

Note that the `mdb-user` for `com/PlanStateListener` defined in `jboss.xml` in `ORCH-ejb-2.0.jar` should be changed to `RequestInjectorMessageBeanUser`, as shown below:

Is this "should be changed" only if the password is changed; or should it be changed at setup? In which case it should probably go somewhere else.

```
    <message-driven>
         <ejb-name>com/PlanStateListener</ejb-name>

<destination-jndi-name>topic/COMOrchestratorTopic</destination-jnd
i-name>
         <mdb-user>RequestInjectorMessageBeanUser</mdb-user>
         <mdb-passwd>password</mdb-passwd>

<mdb-client-id>ProcessInjectorMessageBeanUserSubscriber</mdb-clien
t-id>

<mdb-subscription-id>ProcessInjectorMessageBeanUserSubscriber</mdb
-subscription-id>
    </message-driven>
```

Other EMS users

The passwords for the following user names may also be changed:

- `comEAIStepUser`: this is used by the iProcess Conductor Orchestration Plug-in. Its password is stored in the $SWDIR/`eaijava/eaiorch/eaiorch.properties` file.
- `requestInjector`: this is generally used by third-party products which inject orders to iProcess Conductor. If you change the password for `requestInjector` you must set the new password as the credential for JMS connection.

Appendix B Unsupported XPath Functions in XML Transform Plug-in

Some XPath functions are not supported by XML Transform Plug-in. This appendix describes the XPath functions that are not supported.

Topics

- Unsupported XPath Functions, page 152

Unsupported XPath Functions

Listed below are the XPath functions that are unsupported in the XML Transform Plug-in.

A

```
add-to-dateTime
add-to-time
avg
```

B

```
base-length
base64-to-hex
base64-to-string
```

C

```
compare-date
compare-dateTime
compare-time
concat-sequence
concat-sequence-format
conditionsiteration
create-date
create-dateTime
create-dateTime-timezone
create-time
current-date
current-dateTime
current-dateTime-timezone
```

D

```
datetime
```

E

```
empty
exists
```

F

```
for
format-date
format-dateTime
format-time
```

G

```
get-century-from-date
get-century-from-dateTime
get-day-from-date
get-day-from-dateTime
get-hours-from-DateTime
get-hours-from-time
```

get-minutes-from-dateTime
get-minutes-from-time
get-month-from-date
get-month-from-dateTime
get-seconds-from-dateTime
get-seconds-from-time
get-timezone-from-date
get-timezone-from-dateTime
get-timezone-from-time
get-year-from-date

H

hex-to-base64
hex-length
hex-to-string

I

if-absent
index-of

L

last-index-of
left
lower-case

M

max
min

P

pad
pad-and-limit
pad-front
parse-date
parse-dateTime
parse-time

R

render-xml
right
round-fraction

S

string-round-fraction
string-to-base64
string-to-hex
substring-after-last
substring-before-last

T

timestamp
tokenize

tokenize-allow-empty
translate-timezone
trim

U

upper-case

V

validate-dateTime

X

xor

Glossary

C

classification

A way of grouping managed objects in the iProcess Conductor in a way that is meaningful to your business (for example, by geographic location).

customer premises equipment (CPE)

Customer premises equipment (CPE) is equipment that belongs to a service provider, but is physically located on the customer's premises rather than on the provider's premises. For example, in the telecommunications industry, a Digital Subscriber Line router.

critical path

Represents the maximum length path through an execution plan. There are various types of critical paths that are calculated by the iProcess Conductor to determine whether an execution plan is "in jeopardy." See **jeopardy management**.

D

dependency

A relationship between tasks or phases of tasks in an execution plan in the TIBCO iProcess Conductor. For example, Task B cannot start until Task A completes.

E

execution plan

A collection of process components and their dependencies. Used by the iProcess Conductor Orchestrator to manage the fulfillment of a given order.

execution plan task

This provides the control structure for a single process component with an execution plan.

execution plan template

This is an example execution plan that is used as the basis for activating execution plan instances.

J

jeopardy condition

A circumstance that causes an execution plan, or part of a plan, not to complete on time; or which puts it at risk of not completing on time. See also jeopardy management.

jeopardy consequential action

An action taken by the TIBCO iProcess Conductor as a result of a jeopardy condition. It could be to send a JMS message, initiate an iProcess Engine case start, or suspend the execution plan.

jeopardy management

The process of managing execution plan task progress and risk of failure. This is done by monitoring the critical paths through execution plans. See **critical path**.

M

meta-data

The input and output data required by process components in the TIBCO iProcess Conductor. This data is represented in iProcess procedures as fields.

milestone

A significant scheduling point within the process components of an execution plan. Inter-plan task dependencies can be defined from and to Milestones. Milestones also provide points through which meta-data can pass to and from Process Components.

P

process component

A logical process controlled from a plan task within an execution plan. It maps to a single iProcess procedure at a particular date and time at run-time. At design time, a process component may refer to one or more process versions. Each version will, in turn, reference a single iProcess procedure and an effective date/time range.

process component section

The interval between two milestones in a process component. This may be the whole of a simple process component that has only start and end milestones, or may be a part of a larger process component.

process signature

In an iProcess procedure, the EAI Orchestration steps and the data contained in the TIBCO iProcess fields that are defined as input and outputs in the EAI Orchestration steps. The process signature is created by selecting an iProcess procedure when you create a process component.

S

supply chain management (SCM)

The oversight of materials, information, and finances as they move in a process from supplier to manufacturer to wholesaler to retailer to consumer

summary group

Collection of execution plan tasks that can be arranged in a hierarchical structure of dependencies.

X

XSLT

XML Stylesheet Language Transformation. XSL Transformations (XSLT) is a standard way to describe how to transform (change) the structure of an XML (Extensible Markup Language) document into an XML document with a different structure. XSLT is a recommendation of the World Wide Web Consortium.

Index

A

Action in a plan task 22
AppConfig.xml 72
Application.xml file 72

C

changes from the previous release viii
Changing an order's status 95
Classifying execution plan templates 30
customer support xiv

D

Defining order content 99
Defining order header elements 98, 99
Defining order line elements 98
Defining order request fulfillment elements 99, 100
Designing execution plans 25

E

EAI Orchestrator step
  creating 82
  messages 79
  transaction scope 80
EAI Order step
  creating 92
  transaction scope 91
EAI Transform step
  creating 106
Error handling 51
Execution plan
  creating 42
  designing 25, 25
  developing 39
  manual 41

F

Field mapping
  adding 101
  deleting 101
  in an EAI Order step 101
Fulfillment procedure 35, 35

I

installation directory
  BEA_HOME xii, xii
  iProcessConductorDir xi
  iProcessConductorDomain xi
  JAVA_HOME xi
  JBOSS_HOME xi
  ORACLE_HOME xi
  WL_HOME xii
iProcess procedure 56
  for fulfillment 35, 35
  to handle errors 51

J

Jeopardy conditions 23, 25

M

Manual execution plan development 41
Milestone 22

O

Order
  content 99
  feasibility 39
Order header
  defining 98, 99
Order line
  defining 98, 100
Order request 65, 69
Order request fulfillment
  defining 99, 100

P

Plan task
  action 22
Process component
  defining 58
  designing 22, 31
  identifying 13
  reusing 15
  updating 58
  version 56
Process signature 23, 31
  designing 23, 25

R

Rule
  for fulfillment procedure selection 69
  for template selection 70
  making available 71
rule
  statically defined 72

S

statically defined rules 72
support, contacting xiv

T

technical support xiv

U

Unsupported XPath functions 152