Exchange Module Customization Guide

iPlanet Market Maker

Version 4.5

March 2002

Contents

List of Figures

List of Tables

About This Document

The Exchange Module Customization Guide provides a description of the basic functionality of the iPlanet Market Maker Exchange module, including instructions for customizing the exchange framework and for integrating the Exchange module with an external module.

This preface contains the following sections:

- Audience
- What's in This Document
- Documentation Conventions
- The Document Online
- Product Support

Audience

The audience for this document is the programmer who is customizing the iPlanet Market Maker Exchange module.

What's in This Document

The following table summarizes what each chapter covers.

Table 1 Chapter Summary

If you want to know about this	See this chapter
Description of contents of this guide; listing of documentation set; information on product support	"About This Document"
An overview of the Exchange module and how an exchange works	Chapter 1, "Introduction to Exchange"
Instructions for defining the attributes of the Exchange module	Chapter 2, "Defining the Exchange Attributes"
Instructions for defining the matching and trading rules of the Exchange module	Chapter 3, "Defining the Matching and Trading Rules"
Instructions for integrating the Exchange module with the OMS module.	Chapter 4, "Integrating Exchange with an External System"
Structure of the message oriented middleware (MOM) and Java message service (JMS) tags	Appendix A, "MOM Listener and JMS Tags"

Documentation Conventions

This document uses the following conventions:

- The `monospace` font is used for sample code and code listings, Application Program Interface (API) and language elements (such as method names and property names), file names, path names, directory names, Hypertext Markup Language (HTML) tags, and any text that must be typed on the screen.
- The *italic* font is used in code to represent placeholder parameters (variables) that should be replaced with an actual value.
- Brackets ([ ]) are used to enclose optional parameters.
- A slash (/) is used to separate directories in a path. (Windows NT supports both the slash and the backslash.)

The Document Online

You can find the Exchange Module Customization Guide online in Portable Document Format (PDF) and HTML formats at the following web site:

`http://docs.iplanet.com/docs/manuals/`

Product Support

If you have problems with your iPlanet Market Maker software, contact iPlanet customer support using one of the following mechanisms:

- iPlanet online support web site at:

  `http://www.iplanet.com/support/online/`

  From this location, the CaseTracker and CaseView tools are available for logging problems.

- The telephone dispatch number associated with your maintenance contract

So that the technical support staff can best assist you in resolving problems, please have the following information available when you contact support:

- Description of the problem, including the situation where the problem occurs and its impact on your operation.
- Machine type, operating system version, and product version, including any patches and other software that might be affecting the problem.
- Detailed steps on the methods you have used to reproduce the problem.
- Any error logs or core dumps.

Chapter 1

Introduction to Exchange

The iPlanet Market Maker Exchange module is a customizable framework that allows companies to create electronic trades.

This chapter contains the following sections:

- Functionality
- Customization
- Exchange Schema

Functionality

An *exchange* is a mechanism by which *traders* (both buyers and sellers) asynchronously trade [the symbols of] homogenous commodities using a bid / ask methodology. This is similar to the way symbols are traded in a stock exchange. Symbols best suited to trade as exchanges represent near-commodity items that have a small number of similar attributes. Standard goods and services are also well suited for exchange trading. Exchanges are especially effective in markets where items are affected by volatile supply and demand. By managing excess supply and peak load demand, exchanges can facilitate quick liquidity.

The methodology for setting up an exchange depends on:

- The types of symbols to be traded.
- The rules required to match buy orders and sell orders.
- The rules required to finalize trades.

Unlike a stock exchange, where price and quantity are usually the basis for defining a trade, an iPlanet MarketMaker exchange might have additional factors to consider before a trade can take place. Price may or may not be a factor in defining a trade.

A common example is a *transport exchange*, which is a type of exchange that involves a means / method of transport from one place to another. This could be any type of transportation or container, such as cargo space on an airline. In addition to price and quantity factors, the following factors might also be part of the trade specification:

- A departure airport
- An arrival airport
- A departure date
- An arrival date

Features

The iPlanet MarketMaker Exchange module features include:

- Real-time or batch mode matching of buy orders and sell orders.
- Real-time, market-wide quoting (and price determination, if price is an exchange attribute).
- Order splitting and aggregation.Support for market-wide member black lists, exclusive lists, and preferred lists.
- Built-in transaction audit trail.
- Comprehensive exchange management, including membership, administrative order manipulation, scheduled market open and close, symbol suspension, and so on.
- Automated matching.
- Manual matching, also known as *lifting* (with or without reference).

Users

There are three users involved in exchanges:

- Trader

- Company administrator
- iPlanet Market Maker administrator

Trader

A *trader* is the person or company doing either the buying or selling in an exchange. The basic tasks performed by a trader are:

- Placing orders
- Viewing / browsing orders
- Cancelling orders
- Performing extended (fuzzy) searches
- Manually matching orders—lifting with or without reference
- Viewing audit logs

Company Administrator

The *company administrator* is the person assigned by the iPlanet Market Maker administrator to perform the following Exchange module tasks:

- Setting up and administering trader accounts
- Restricting trading partners using the black list

iPlanet Market Maker Administrator

The *iPlanet Market Maker administrator* (also known as immhost) owns the iPlanet Market Maker software and sets up the digital marketplace. This user has the authority to create and approve member companies, edit company profiles, and manage users for companies. The specific tasks that relate to the Exchange module are:

- Setting up exchanges
- Administering exchange membership and status
- Managing symbol status

Customization

Because the requirements of a trade in an iPlanet Market Maker exchange can have so many variables, the Exchange module provides very little out-of-box business functionality. Instead, it provides a framework of services that you can extend or customize to create your own exchanges. You can specify the matching and trading rules that determine how to match orders and settle trades. You can also specify whether to create buy orders and sell orders interactively or in batch mode.

Two sample exchange implementations are provided with the iPlanet Market Maker software. For more information about these sample exchanges, refer to Chapter 2, “Defining the Exchange Attributes.”

Exchange Schema

Each exchange has a schema that defines its particular characteristics. By identifying exactly what behavior you want the exchange to implement, you can tailor the exchange to your precise needs. Each exchange uses a separate schema, which allows you to create and host multiple exchanges in a single installation.

Schema Processes

To set up the schema for an exchange, you will define attributes associated with the following processes:

- Pre-match processing
- Match processing
- Post-match processing

Pre-match Processing

Pre-match processing is determined by setting the exchange rules that define the following:

- Life span of orders (for example, Good Until Cancelled, or Expiration Date)
- Sequence in which to process orders
- Flatten function—Mechanism to convert the values defined in the user interface to a set of values that the Exchange module can process

Details about the user interface and pre-match processing are contained in Chapter 2, “Defining the Exchange Attributes.”

Match Processing

Match processing is determined by setting the exchange rules that define how you want to match buy orders and sell orders. Details about match processing are contained in Chapter 3, “Defining the Matching and Trading Rules.”

Post-match Processing

Post-match processing is determined by setting the exchange rules that define how to convert matched orders into traded orders. Details about post-match processing are contained in Chapter 3, “Defining the Matching and Trading Rules.”

Schema Services

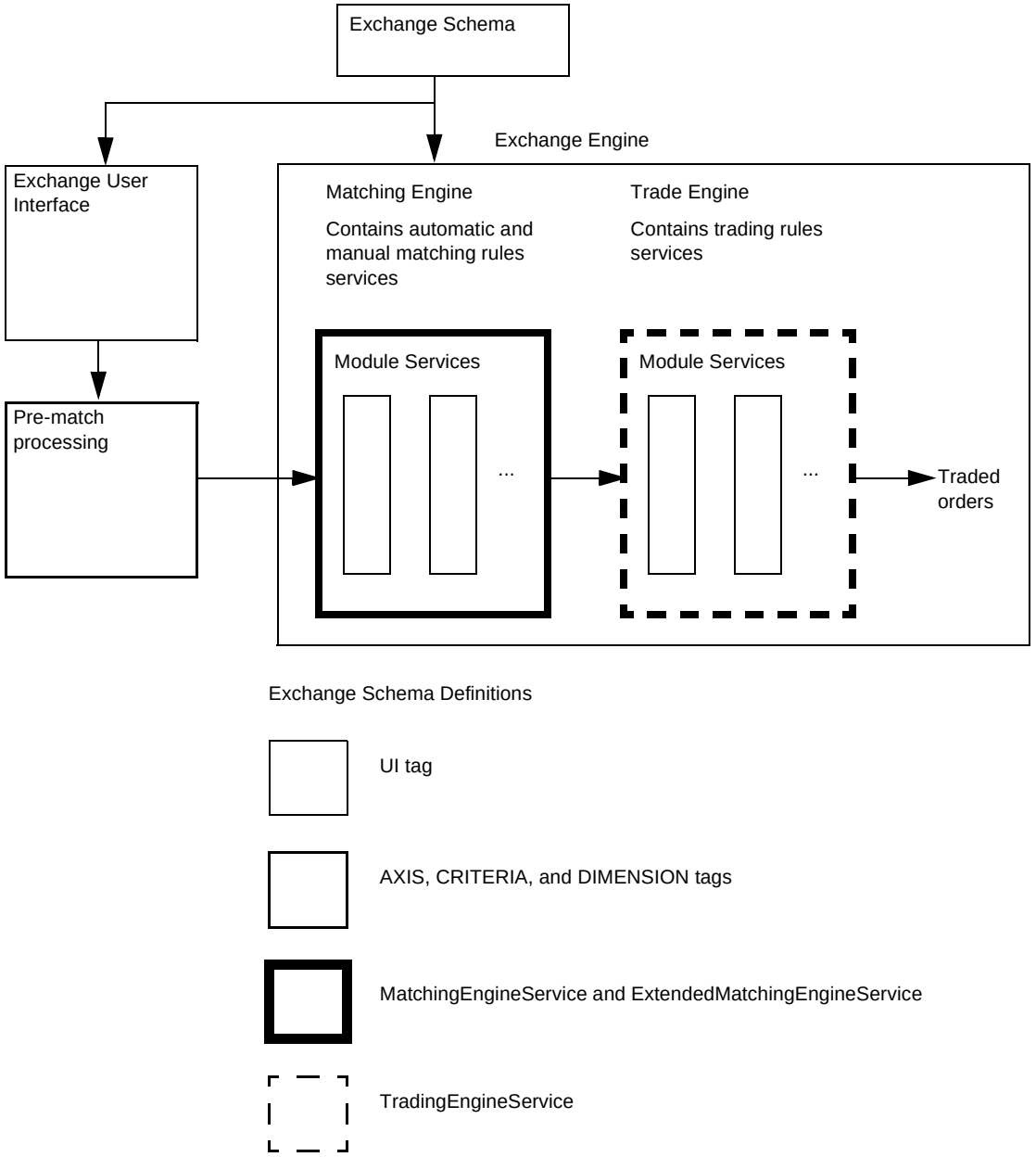
Within each exchange schema, there are two types of services:

- System services
- Module services

System services provide basic Exchange module functionality and usually do not need to be customized. *Module services* are typically meant to be customized to implement your requirements for a particular exchange.

Figure 1-1 illustrates the exchange flow.

Figure 1-1 Flow of an Exchange



Chapter 2

Defining the Exchange Attributes

This chapter provides instructions for setting up attributes to identify order matching criteria for the iPlanet Market Maker Exchange module.

This chapter contains the following sections:

- Overview
- Implementing the Exchange Attributes

Overview

To be able to match buy and sell orders, the Exchange module must first know what criteria to use to identify whether a match does, or does not, exist. This is accomplished by defining a *common set of attributes*, also known as the *matching criteria*. This set of attributes, such as price and quantity, is the basis for comparing orders in an exchange. After you have defined these attributes, the Exchange module compares these attributes to identify potential matches between buy orders and sell orders.

Symbols in an exchange are similar and share common matching attributes. To set up an exchange, you need to specify the symbols you want to trade and the attributes associated with those symbols. Because of the flexible nature of the iPlanet Market Maker Exchange module, you can define whatever attributes are necessary to compare orders.

An exchange has the following types of attributes:

- Criteria attributes
- Dimensions attributes
- Axes attributes

Criteria

Criteria define all the information required to place an order in an exchange. Each criteria has a set of criteria fields, which you enter in the Exchange module user interface. The criteria fields specify the quantity you want to buy or sell, at what price, and so on. These are the front-end, or human-entered, or specified, matching attributes in an exchange.

Dimension

Axes and criteria are based on *dimensions*, which define their internal value types.

Axes

Axes provide a set of values to allow comparisons between orders. They are the back-end matching engine-readable representations of matching criteria.

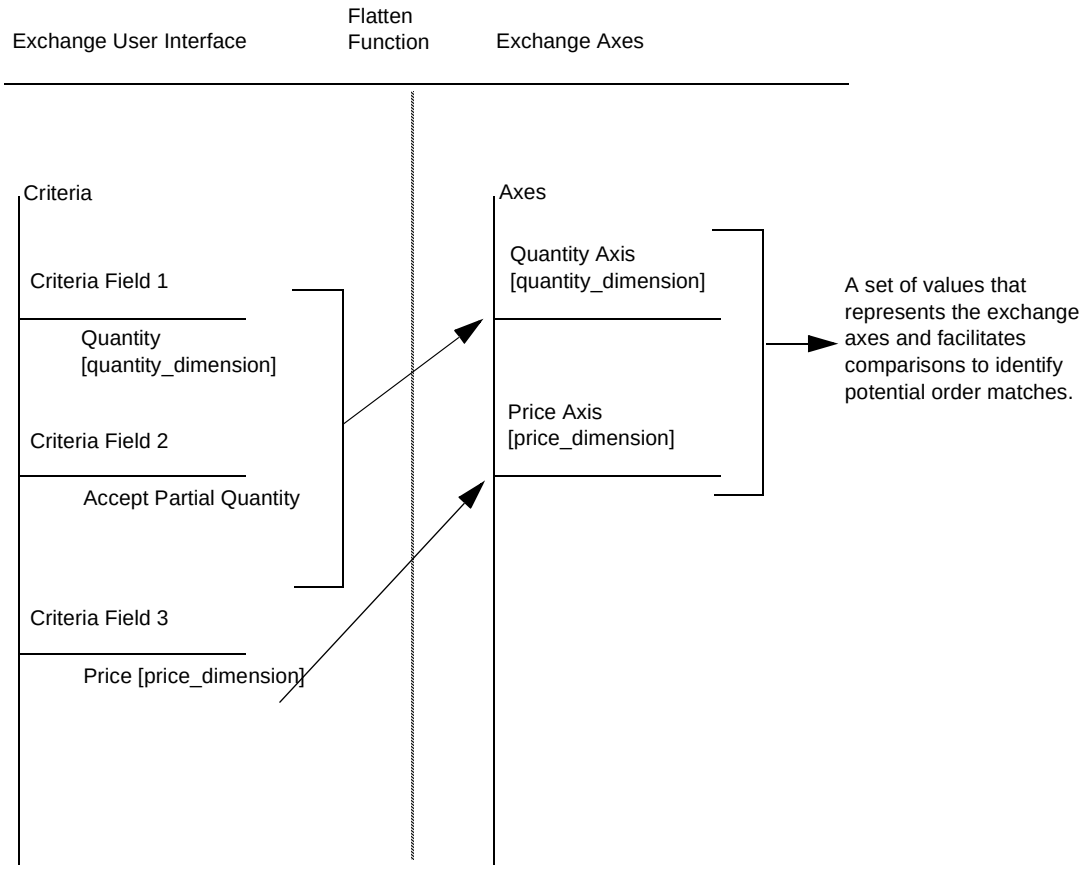
Flattening

To convert the criteria to a set of matching attribute values that the Exchange module can interpret, you define a *flatten* function, which is a Java class that translates the criteria to the values of the exchange axes. The Exchange module processes the axes values to identify potential matches.

Figure 2-1 shows an example of the relationship between axes and criteria. The left side shows the defined criteria and criteria fields that appear in an Exchange module user interface. In this example, the Order Criteria of Market means that the buyer or seller accepts the current market price, so no explicit price is specified. Other criteria types might require you to specify a price.

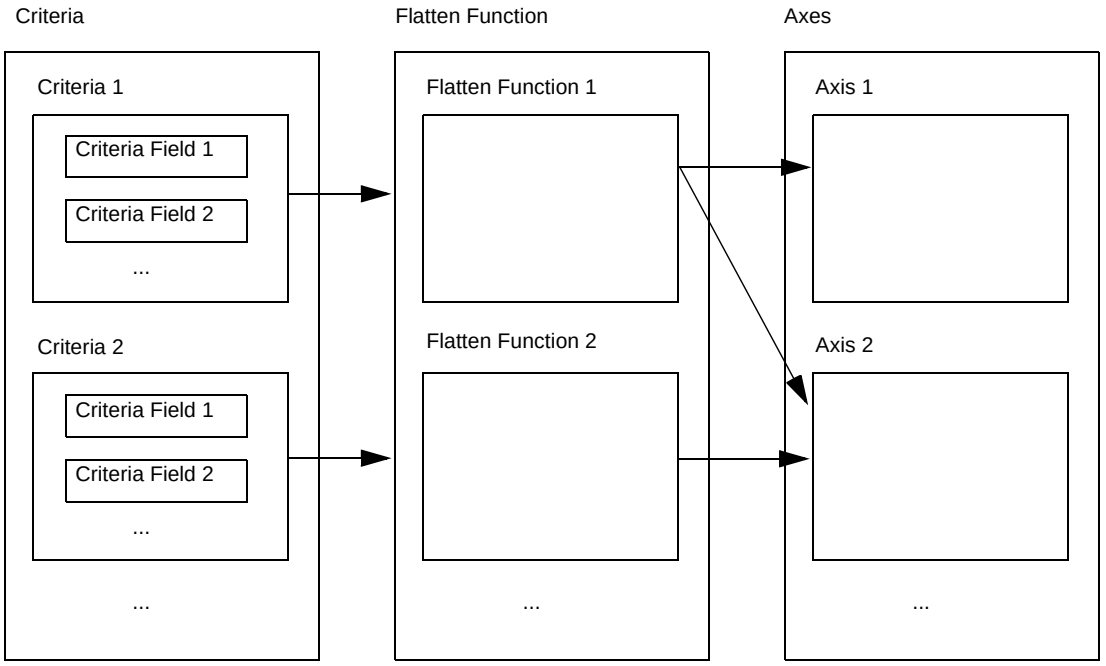
The right side of Figure 2-1 represents the exchange processing that takes place after the flatten function converts the criteria fields to a value of the axis that the Exchange module uses to identify potential order matches.

Figure 2-1 Criteria and Axes in an Exchange



There is a flatten function for each criteria. It is possible for a flatten function to convert more than one criteria field to the same axis, as shown in Figure 2-1. Figure 2-2 shows another representation of how a flatten function converts criteria to axes.

Figure 2-2 Flattening More Than One Criteria Field to the Same Axis



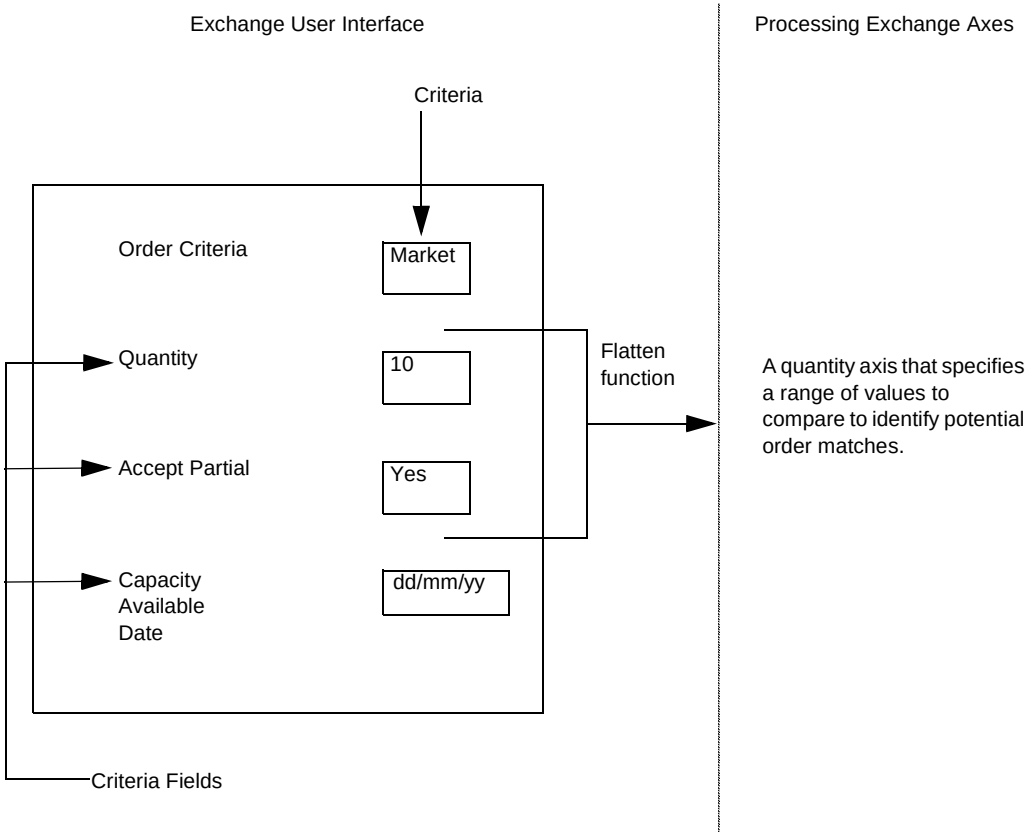
Often there is a one-to-one correspondence between axes and criteria, but not always. In an Exchange module user interface you might specify a quantity criteria field as an amount, and also have an option to accept a partial quantity.

Example

Suppose that Figure 2-3 is a buy order, willing to accept a partial quantity, for 10 widgets. There is a previous buy order in the exchange for 10 widgets only, and a sell order for 7 widgets. Because there are orders to buy 20 widgets and sell 7 widgets, the Exchange module needs a method for comparing 20 to 7 to determine if there is a potential match. In this case, quantity is not a single value as specified in the user interface, but is a range of values.

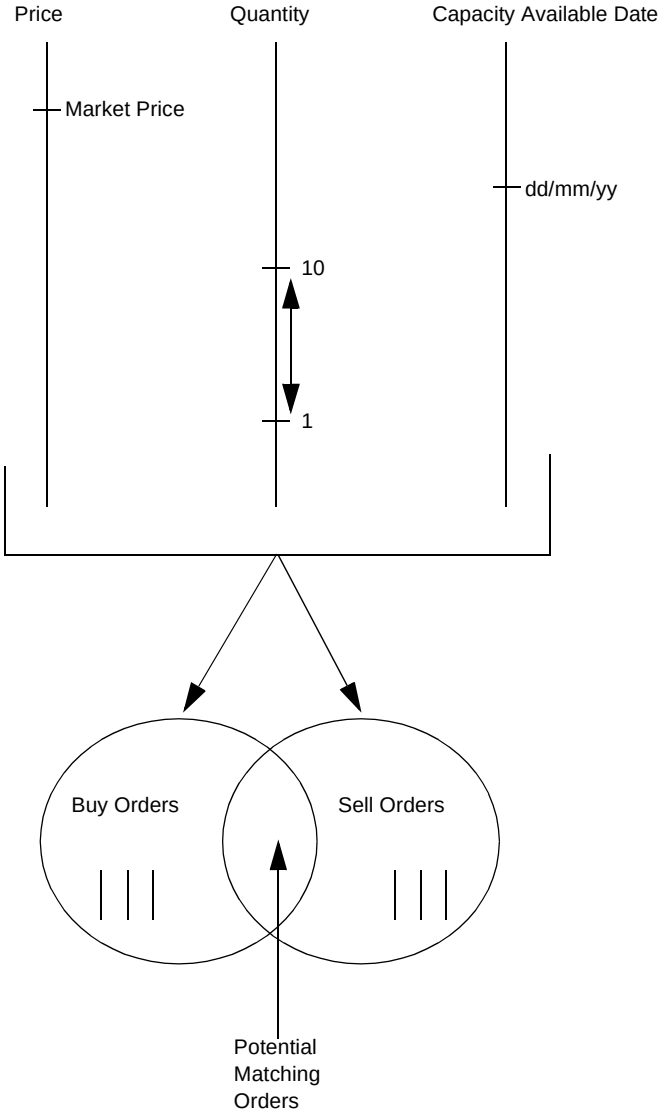
In the example, the buy order for 10 widgets willing to accept a partial quantity is a potential match with the sell order for 7 widgets. The other buy order for 10 widgets not willing to accept a partial quantity is not a potential match with the order to sell 7 widgets.

Figure 2-3 Buy Order Accepting a Partial Quantity



To determine if there are potential matches, the Exchange module compares all the axes of all the buy and sell orders. If there is any non-zero overlap in all the axes of a buy and sell order, there is a potential match. Figure 2-4 shows a representation of the axes associated with the criteria fields in Figure 2-3.

Figure 2-4 Exchange Axes for Matching Buy and Sell Orders



Example

Suppose, given the criteria and axes in Figure 2-3 and Figure 2-4, there is a buy order for 10 symbols and a sell order for 7 symbols. The Exchange module compares the buy and sell order axes values to determine if there is a potential match.

Table 2-1 shows another interpretation of the axes in Figure 2-4. A potential match requires a Yes in the Potential Match column for *all* axes in a buy and sell order. Because the order is for the market price, and not for a specific price or range of prices, the Price axis is irrelevant in the comparison. If both the buy and sell orders accept the market price, the Price axis has an overlap by default. Note that this potential match need not be exact, as is the case with the Quantity axis.

Table 2-1 Matching Potential Buy and Sell Orders

Axis	Order Number	Buy Order	Sell Order	Potential Match
Quantity	Order 1	10, will accept a partial quantity	7	Yes
	Order 2	10 only	7	No
Capacity Available Date	Order 1	03/30/01	03/30/01	Yes
	Order 2	03/30/01	06/01/01	No

Implementing the Exchange Attributes

The requirements that define the schema for an exchange depend on the symbols you want to trade. The exchange symbols, criteria, axes, and the user interface in which to enter the criteria are defined using an XML file.

The iPlanet Market Maker software comes with two sample exchange implementations. The schema files for these exchanges are in this directory:

<*ias_inst_dir*>/ias/APPS/imm40/imm40/WEB-INF/classes/exchange

where <*ias_inst_dir*> is where you installed your iPlanet Application Server software.

The sample schema files are:

- BasicSchema.xml file
- TransportSchema.xml file

You can copy these files and use them as templates to edit and set up the schema for your exchange. The ExchangeList.xml file in the same directory lists the exchanges in a marketplace. This file lets you define multiple exchanges on a single host.

The following sections explain the tasks of implementing the symbols, criteria, and axes for a sample exchange (this sample is for trading natural gas energy services):

- Defining the Symbols to be Traded
- Defining Axes
- Defining Criteria
- Defining Dimensions
- Defining the Life Spans for Orders
- Defining an Exchange User Interface

Defining the Symbols to be Traded

The names of the symbols to be traded in the sample energy exchange are.

- Arg Gas Phy (Buenos Aires)
- Arg Gaos Phy (GBA)
- Arg Gas Phy (Litoral)
- BEL Gas Phy Fwd
- BEL Gas Phy (ZEE HUB)
- US Gas Swap (Nymex)
- US Gas Phy Index (IF Transco Z6NY)
- US Gas Basis (NGI Chicago)
- CAN Gas Phy (NIT)
- CAN Gas Basis (GD/D Dawn)
- UK Gas Phy (NBP)
- NL Gas Phy Fwd (Oude/BEB H-Gas)

By default, the symbol catalog data store is located in the same directory as the schema file. This can be changed by modifying the service property FILE for the SymbolCatalogService.

Language-specific symbol stores can be added in the standard Java fashion. For example, a French version of the <*exchange_name*>Symbol.properties file should be named <*exchange_name*>`Symbol_fr.properties`; it can be referenced in the exchange schema descriptor file with its path.

To locate the file containing the definitions of symbols for an exchange, search for SymbolCatalogService in the schema descriptor file of the exchange. The definition of Symbol Catalog Service in the XML schema descriptor file, and the syntax, are shown in Table 2-2 and Table 2-4 respectively.

Table 2-2 Sample Symbol Catalog Service Definition

```
<SERVICE name="SymbolCatalogService" impl_class
="com.reference.energyb.service.PropertyBasedSymbolCatalogServiceImpl">
      <SERVICE_PROPERTIES>
              <SERVICE_PROPERTY name="FILE" value="exchange/EnergyBSymbols"/>
      </SERVICE_PROPERTIES>
</SERVICE>
```

The Table 2-3 below shows the contents of the EnergyBsymbol.xml file which defines the symbols for the exchange EnergyB. This file defines the order in which the exchange symbols are loaded and displayed in the user interface. To localize the exchange, language-specific symbols, for example, French, can be stored in the ExchangeBSymbol_fr.properties file. This file can then be referenced in the exchange XML schema descriptor EnergyBSchema.xml file

NOTE The symbol GUID should not be localized.

Table 2-3 Sample Symbols Definition

```
SYMBOL0=Arg_Gas_Phy_Buenos_Aires;Arg Gas Phy (Buenos Aires);Arg Gas Phy (Buenos
Aires)

SYMBOL1=Arg_Gas_Phy_GBA;Arg Gas Phy (GBA);Arg Gas Phy (GBA)

SYMBOL2=Arg_Gas_Phy_Litoral;Arg Gas Phy (Litoral);Arg Gas Phy (Litoral)

SYMBOL3=BEL_Gas_Phy_Fwd;BEL Gas Phy Fwd;BEL Gas Phy Fwd

SYMBOL4=BEL_Gas_Phy_ZEE_HUB;BEL Gas Phy (ZEE HUB);BEL Gas Phy (ZEE HUB)

SYMBOL5=US_Gas_Swap_Nymex;US Gas Swap (Nymex);US Gas Swap (Nymex)

SYMBOL6=US_Gas_Phy_Index_IF_Transco_Z6NY;US Gas Phy Index (IF Transco Z6NY);US
Gas Phy Index (IF Transco Z6NY)

SYMBOL7=US_Gas_Basis_NGI_Chicago;US Gas Basis (NGI Chicago);US Gas Basis (NGI
Chicago)

SYMBOL8=CAN_Gas_Phy_NIT;CAN Gas Phy (NIT);CAN Gas Phy (NIT)

SYMBOL9=CAN_Gas_Basis_GD/D_Dawn;CAN Gas Basis (GD/D Dawn);CAN Gas Basis (GD/D
Dawn)

SYMBOL10=UK_Gas_Phy_NBP;UK Gas Phy (NBP);UK Gas Phy (NBP)

SYMBOL11=NL_Gas_Phy_Fwd_Oude/BEB_H-Gas;NL Gas Phy Fwd (Oude/BEB H-Gas);NL Gas
Phy Fwd (Oude/BEB H-Gas)
```

Table 2-4 SERVICE and SERVICE_PROPERTY Tag Arguments

Symbol Argument	Value Type	Description
SERVICE name	SymbolCatalogService	Name of the service.
`impl_class`	STRING	Name of the Java class file that reads the symbol definitions in the *<exchange_name>*`Symbol.properties` file.
`SERVICE_PROPERTY`		
`name`	FILE	Indicates that the symbols are contained in a file.
`value`	STRING	Points to the file containing the symbols.

Defining Axes

To locate the definitions for the axes in one of the sample XML files, search for >AXIS_CONFIG. The syntax and definitions for the quantity axis in this exchange are shown in Table 2-6.

Table 2-5 Sample Quantity Axis Definition

```
<AXIS_CONFIG guid="energy_quantityAxis_guid"
             name="QuantityAxis"
             index="1"
             dimension="energy_quantity_guid"
             is_aggregation="true"/>
```

The arguments for the AXIS_CONFIG tag are described Table 2-6.

Table 2-6 AXIS_CONFIG Tag Arguments

Axis Argument	Value Type	Description
guid	guidid	Unique identifier for the axis.
name	STRING	A name that identifies the axis.
index	INTEGER	Specifies the sequence number for the axis (1, 2, 3, and so on).
dimension	STRING	Specifies the GUID of the dimension on which the axis is based.
is_aggregation	BOOLEAN	Specifies whether to add (aggregate) the axes values for orders. Set this value to True for only one axis that contains the values that need to be combined, usually for the quantity axis. For all the other axes, set this value to False.

Defining Criteria

To locate the definitions for the criteria in one of the sample XML files, search for <CRITERIA>.

NOTE You specify the flatten function file name in the flatten argument for the CRITERIA tag. This class contains the logic that converts the data defined in each CRITERIA_FIELD tag to an axis.

In an exchange XML file, there are two types of Java class files:

- Classes prefaced with com.iplanet—These are exchange system classes that you generally do not have to customize to create your own exchange.
- Classes prefaced with com.reference—These are classes that you might need to customize or replace, depending on the needs of your exchange. The flatten class file is an example.

If you want to introduce a new criteria, you must write a flattening class function and specify the function name for the flatten argument. You can also extend the flattening class of one criteria and implement the flattening logic for another criteria.

The syntax and definitions for the market order criteria in the energy exchange is shown in Table 2-8.

Table 2-7 Sample Market Criteria Definition

```
<CRITERIA guid="energy_market_guid"
          name="Market"
          action="ANY"
          flatten="com.reference.energy.flatten.Market">
            <CRITERIA_FIELDS>
                <CRITERIA_FIELD name="Quantity"
                                dimension="energy_quantity_guid"/>
                <CRITERIA_FIELD name="Partials"
                                dimension="energy_boolean_guid"/>
                <CRITERIA_FIELD name="CapacityAvailableDate"
                                dimension="energy_date_guid"/>
             </CRITERIA_FIELDS>
</CRITERIA>
```

The arguments for the CRITERIA and CRITERIA_FIELD tags are listed in Table 2-8.

Table 2-8 CRITERIA Tag Arguments

Criteria Argument	Value Type	Description
`CRITERIA guid`	guidid	Unique identifier for the criteria.
`name`	STRING	A string that identifies the criteria.
`action`	BUY	BUY specifies that the criteria applies to a buy order, SELL to a sell order, and ANY to both.
	SELL	
	ANY	
`flatten`	Java class	Specifies the Java class file used to flatten the criteria to a set of axis values that the Exchange module can use to identify potential matching buy and sell orders.
`CRITERIA_FIELD`		
	name	Name of the criteria field that appears in the Exchange module user interface.
	dimension	GUID for the dimension associated with the criteria field.

Defining Dimensions

To locate the definitions for the dimensions in one of the sample XML files, search for <DIMENSION>. The syntax and definitions for the market order criteria in this exchange is shown in and Table 2-10.

Table 2-9 Sample Quantity Dimension Definition

```
<DIMENSION guid="energy_quantity_guid"
           name="Quantity"
           type="INT"
           compare_function="DEFAULT"
           distance_function="DEFAULT"
           aggregation_function="DEFAULT"
           boundary_min="1">
</DIMENSION>
```

The arguments for the DIMENSION tag are described in Table 2-10.

Table 2-10 DIMENSION Tag Arguments

Dimension Argument	Value Type	Description
`guid`	guidid	Unique identifier for the dimension.
`name`	STRING	A string that identifies the dimension
`type`		Defines the type of the dimension: int, REAL, TIME, SET, FREEFORM.
	int	A positive whole number that maps to a Java primitive `int`.
	REAL	An irrational or rational number that maps to a Java primitive `double`.
	TIME	A date and time that maps to a Java type `java.util.Date`.
	SET	A collection of values that maps to a Java type `java.util.Map`.
	FREEFORM	A value expressed as a string. This attribute type is used to match values that are not known in advance, such as the member ID for a preferred customer. `FREEFORM` dimensions do not have an associated Order or Distance function.

Table 2-10 DIMENSION Tag Arguments *(Continued)*

Dimension Argument	Value Type	Description
compare_function		Specifies if the dimension values are ordered and how they can be compared based on that order. This attribute lets you define a non-alphanumeric type of comparison. You usually do not have to change this attribute and can leave it set to DEFAULT.
	DEFAULT	
distance_function		Allows for the computation of the distance between two values. For example, you can specify if the dimension values have a distance between them, such as the distance between two airports. You usually do not have to change this attribute and can leave it set to DEFAULT.
	DEFAULT	
aggregation_function		
	DEFAULT	Specifies if the dimension allows its values to be combined (aggregated). You can specify an aggregation attribute (DEFAULT) for only one dimension, usually the quantity dimension of the symbol to be traded in the exchange. For the other dimensions, specify NONE.
	NONE	
boundary_min	Null or number	Value that represents the minimum value for the dimension. Null represents infinity. This attribute is optional.
boundary_max	Null or number	Value that represents the minimum value for the dimension. Null represents infinity. This attribute is optional.
DIMENSION_SET	Set	Defines a set of dimension values, such as True and False to define a Boolean set.

Defining the Life Spans for Orders

To control how long orders are eligible to be matched in an exchange, specify the information for the LIFESPANS tag. Search for <LIFESPANS> in one of the sample files.

Table 2-11 Sample Life Span Definitions

```
<LIFESPANS>
    <LIFESPAN guid="energyb_goodUntilCancel_guid"
              name="GoodUntilCancel"
              action="ANY"
              calculator="com.reference.energyb.lifespan.GoodUntilCancel">
                <LIFESPAN_FIELDS>
                </LIFESPAN_FIELDS>
    </LIFESPAN>
    <LIFESPAN guid="energyb_immediateOrCancel_guid"
              name="ImmediateOrCancel"
              action="ANY"
              calculator="com.reference.energyb.lifespan.ImmediateOrCancel">
                <LIFESPAN_FIELDS>
                </LIFESPAN_FIELDS>
    </LIFESPAN>
    <LIFESPAN guid="energyb_goodUntilDate_guid"
              name="GoodUntilDate"
              action="ANY"
              calculator="com.reference.energyb.lifespan.GoodUntilDate">
                <LIFESPAN_FIELDS>
                <LIFESPAN_FIELD name="ExpireDate"
                                dimension="energyb_date_guid"/>
                 </LIFESPAN_FIELDS>
                 </LIFESPAN>
</LIFESPANS>
```

The syntax and definitions for life spans in this exchange is shown in Table 2-12.

Table 2-12 LIFEPANS> Tag Arguments

Life Span Argument	Value Type	Description
LIFESPAN guid	guidid	Unique identifier for the life span.
name	STRING	A string that identifies the life span.
action	BUY	BUY specifies that the criteria applies to a buy order, SELL to a sell order, and ANY to both.
	SELL	
	ANY	
calculator	Java class	Specifies the Java class file used to calculate the life span of an order.
LIFESPAN_FIELD		

Table 2-12 LIFEPANS> Tag Arguments *(Continued)*

Life Span Argument	Value Type	Description
	name	Name of the life span field that appears in the exchange user interface.
	dimension	GUID for the dimension associated with the life span field.

Defining an Exchange User Interface

To locate the definitions for the Exchange module user interface, search for <UI_CRITERIAS>.

Table 2-13 Sample User Interface Definitions

```
<UI_CRITERIAS guid="energy_ui_criterias_guid"
        widgettype="RADIOBUTTON"
        ask_default_guid="energy_ui_criteria_market_guid"
        bid_default_guid="energy_ui_criteria_market_guid">
        <UI_CRITERIA guid="energy_ui_criteria_market_guid"
            name="Market"
                descriptor_guid="energy_market_guid"
                action="ANY">
                <UI_CRITERIA_FIELDS>
                        <UI_CRITERIA_FIELD criteria_field_name="Quantity"
                                label="Quantity"
                                widgettype="TEXT"
                                row="1"
                                col="1"
                                screen="2">
                </UI_CRITERIA_FIELDS>
```

The arguments for the UI_CRITERIA tag are described in Table 2-14.

Table 2-14 UI_CRITERIA Tag Arguments

UI Criteria Argument	Value Type	Description
guid	guidid	
label	STRING	
widgettype	RADIOBUTTON, CHOICE, or LIST	
ask_default_guid	UI_CRITERIA.guid	

Table 2-14 UI_CRITERIA Tag Arguments *(Continued)*

UI Criteria Argument	Value Type	Description
bid_default_guid	UI_CRITERIA.guid	
UI_CRITERIA		
guid	guidid	
name	STRING	
descriptor_guid	CRITERIA guid	
action	BUY, SELL, or ANY	
UI_CRITERIA_FIELD		
criteria_field_name	STRING	
label	STRING	Specifies the default name for the i18n string.
widgettype	TEXT, PASSWRD, HIDDEN, STRING, RADIOBUTTON, CHECKBOX, CHOICE, LIST, MULTILIST, DATE	Specifies the HTML component type for the criteria field.
row	INTEGER	Specifies the row number in which the criteria field is displayed relative to the other criteria fields.
column	INTEGER	Specifies the column number in which the criteria field is displayed relative to the other criteria fields.
screen	INTEGER	Specifies the screen number in which the criteria field is displayed.
UI_CRITERIA_FIELD_ITEM		
description	STRING	Name in list of choices displayed for a criteria field.
value	STRING	Value of the description attribute.
default	INTEGER	Specifies the order of the entry in a list of choices displayed for a criteria field.

Chapter 3

Defining the Matching and Trading Rules

This chapter provides instructions for defining the matching and trading rules for the iPlanet Market Maker Exchange module.

This chapter contains the following sections:

- Overview
- Specifying Matching Rules Services
- Defining Trading Rules Services

Overview

After you have defined the axes, criteria, and dimensions for your exchange as explained in Chapter 2, “Defining the Exchange Attributes,” the Exchange module can convert the order criteria to a common set of values in the axes of the exchange to identify potential matching of buy and sell orders. The method for doing this is to specify the modules that contain the matching rules in the same XML file in which you defined your axes, criteria, and dimensions.

The next step is to define the matching rules that determine which potential matching buy and sell orders are actually considered matches in the exchange. There might be cases when you want to further refine what constitutes a match beyond a comparison of values for a common set of axes. For example, you can define a matching rule that identifies companies that belong to a *black list*, which means they are precluded from participating in the exchange. A potential matching buy and sell order that has a buyer or seller on the black list would not produce a match.

Specifying Matching Rules Services

There are two types of exchange matching services that specify the matching rules:

- System services
- Module services

System Services Classes

System service classes implement the logic for the core exchange matching functionality. You usually do not have to customize them. These class names have a suffix of EngineService. Their paths have the com.iplanet prefix.

There are two system service classes that specify matching rules:

- MatchingEngineServiceImpl
- ExtendedMatchingEngineServiceImpl

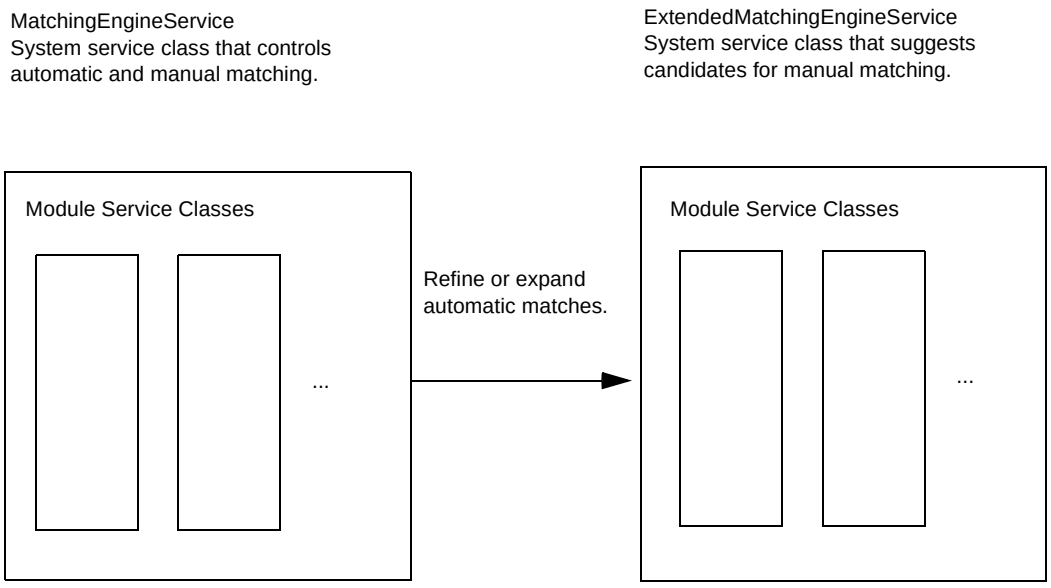
The MatchingEngineServiceImpl service class and its module service classes control automatic and manual matching of buy and sell orders in an exchange.

The ExtendedMatchingEngineServiceImpl service class and its module service classes let you extend matching to suggest potential matching buy and sell orders in the user interface. See Figure 3-1.

Example

Suppose you want users in your exchange user interface to be able to match buy and sell orders that are a close, but not exact, match. This capability is called *extended matching*. Within the ExtendedMatchingEngineServiceImpl service class is a module class called BallonServiceImpl (see the last highlighted lines in Table 3-3) that specifies the logic to expand, or *balloon*, the axes of your exchange. This results in order matches that are typically beyond the number of exact matches.

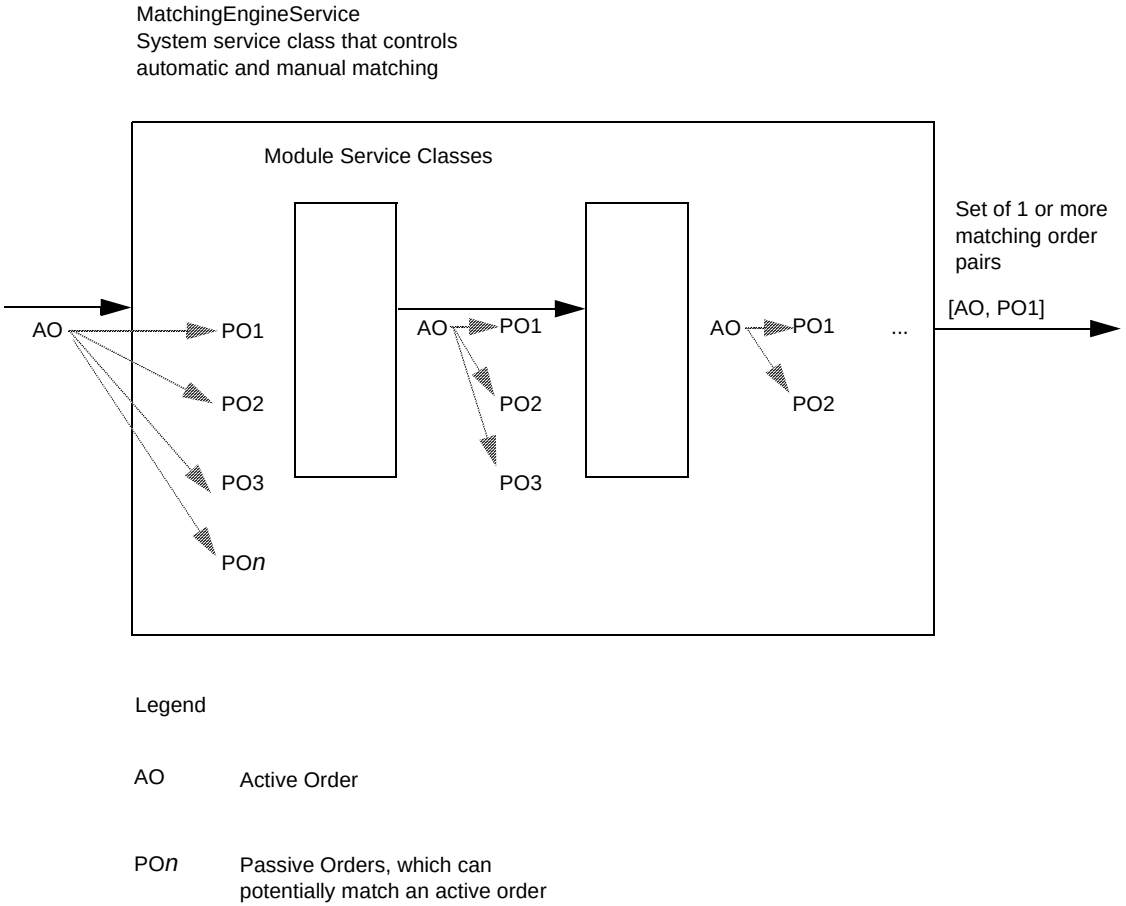
Figure 3-1 System and Module Service Classes



An active order entered in the Exchange module user interface is compared against a set of previously entered *passive orders* (order that are not yet matched). The matching engine compares each active order to the passive orders, one active order at time. For example, if there is an active order for the symbol X, that order is compared against the passive orders first before another active order for X can enter the matching engine for comparison.

You can use the module service classes to determine which potential matches become actual matches. These services let you refine your matching rules depending on your exchange requirements as shown Figure 3-2.

Figure 3-2 Refining Order Matches



Module Services Classes

Under each system matching service class you can specify module service classes. You can customize the module service classes to implement specific matching rules in your exchange. These class names have a suffix of ModuleService. Their paths have a prefix that is not com.iplanet. For example, in the `BasicSchema.xml` and the `TransportSchema.xml` sample exchange schema files, the module service classes have the `com.reference` prefix.

You can edit the sample module service classes in the iPlanet Market Maker Exchange module, or use them as templates to create new module service classes. You can find the `.java` files for the sample exchanges in the following directory:

<*imm_install_dir*>`/iMM/doc/exchange/sample`

where <*imm_install_dir*>is the directory where you installed your iPlanet Market Maker software.

Example

Suppose you want to specify a preferential trading arrangement for a list of preferred companies so that orders on this list are automatically matched before all other orders. The name of your module service class to implement this capability might be PreferredListMatchingModuleServiceImpl.java. Because you want rule to be applied for automated matching, you add this class under the MatchingEngineService system service class. See the highlighted text in Table 3-1.

Table 3-1 Specifying an Additional Module Class

```
<SERVICE name="MatchingEngineService"
impl_class="com.iplanet.ecommerce.vortex.exchange.service.impl.MatchingEngineS
erviceImpl">
    <SERVICE_PROPERTIES>
       <SERVICE_PROPERTY name="SystemMatchingEnabled" value="true"/>
        <SERVICE_PROPERTY name="SystemMatchingModuleServiceKey.0"
value="NVPMatchingModuleService"/>
        <SERVICE_PROPERTY name="SystemMatchingModuleServiceKey.1"
value="BlackListMatchingService"/>
        <SERVICE_PROPERTY name="SystemMatchingModuleServiceKey.2"
value="PreferredListMatchingModuleServiceImpl"/>
        <SERVICE_PROPERTY name="SystemAggregationAxesMatchingModuleServiceKey"
value="PruneAggregatedAxisMatchingModuleService"/>
       <SERVICE_PROPERTY name="ManualMatchingEnabled" value="true"/>
        <SERVICE_PROPERTY name="ManualMatchingModuleServiceKey.0"
value="BlackListMatchingService"/>
        <SERVICE_PROPERTY name="ManualMatchCalculatorImplClass"
value="com.reference.energyb.service.ManualMatchCalculatorImpl" />
        <SERVICE_PROPERTY name="message.topic.matches" value="matches"/>
    </SERVICE_PROPERTIES>
</SERVICE>
<SERVICE name="NVPMatchingModuleService"
impl_class="com.reference.energyb.service.NVPMatchingModuleServiceImpl">
        <SERVICE_PROPERTIES>
        <SERVICE_PROPERTY name="NVPAxisName.0" value="PriceAxis" />
        <SERVICE_PROPERTY name="NVPAxisName.1" value="QuantityAxis"/>
    <SERVICE_PROPERTY name="NVPAxisName.2" value="CapacityAvailableDateAxis"/>
    </SERVICE_PROPERTIES>
</SERVICE>
<SERVICE name="BlackListMatchingService"
impl_class="com.reference.energyb.service.BlackListMatchingModuleServiceImpl">
    <SERVICE_PROPERTIES>
    </SERVICE_PROPERTIES>
</SERVICE>
<SERVICE name="PreferredListMatchingModuleServiceImpl"
impl_class="com.reference.energyb.service.PreferredListMatchingModuleServiceIm
pl">
    <SERVICE_PROPERTIES>
    </SERVICE_PROPERTIES>
</SERVICE>
...
```

Table 3-1 lists the module service classes that define the logic for the matching rules.

Table 3-2 Module Service Classes

Module Class Name	Description
BallonImpl	Contains the logic to increase, or balloon, axes to enable extended matches for buy and sell orders with axes that are a close, but not exact, matches.
BlackListMatchingModuleServiceImpl	The logic to filter out matching orders from companies on a black list. Users on a black list are not allowed to trade with a particular company.
ManualMatchCalculatorImpl	Contains the logic to calculate the remaining quantity, if any, of manually matched buy and sell orders.
NVPMatchingModuleServiceImpl	Compares all the axes to determine which potential matching buy and sell orders are exact matches.
OrderSequenceServiceImpl	Lets you customize the priority order in which to process incoming orders. The sample implementation specifies that orders are passed to the matching engine according to their time stamp.
PruneAggregatedAxisMatchingModuleServiceImpl	Prunes a list of matching orders and returns a subset of orders that collectively satisfy an aggregated axis, which is usually the quantity axis. This class also computes the remaining value of the aggregated axis to enable matching partial orders.
TransportQuoteServiceImpl	Contains the logic to determine the current market price for a symbol.

Matching Orders in the Exchange User Interface

To determine actual matching orders from a set of potential matching orders, the ExtendedMatchingEngineServiceImpl service class uses a mechanism similar to that of the MatchingEngineServiceImpl class. But the ExtendedMatchingEngineServiceImpl service class does not produce actual matches. Instead, it and suggests identifies

potential matching orders to be displayed in the Exchange module user interface, allowing you to manually perform matching of the buy and sell orders in the user interface. This is referred to as *lifting* orders. You can further refine this lifting as either standalone lifting or reference lifting.

Standalone Lifting

In a *standalone lift*, passive orders are displayed in the Exchange module user interface. When you lift an order, the Exchange module automatically creates an opposing matching order. For example, if your lift is for a buy order, a matching sell order is created automatically.

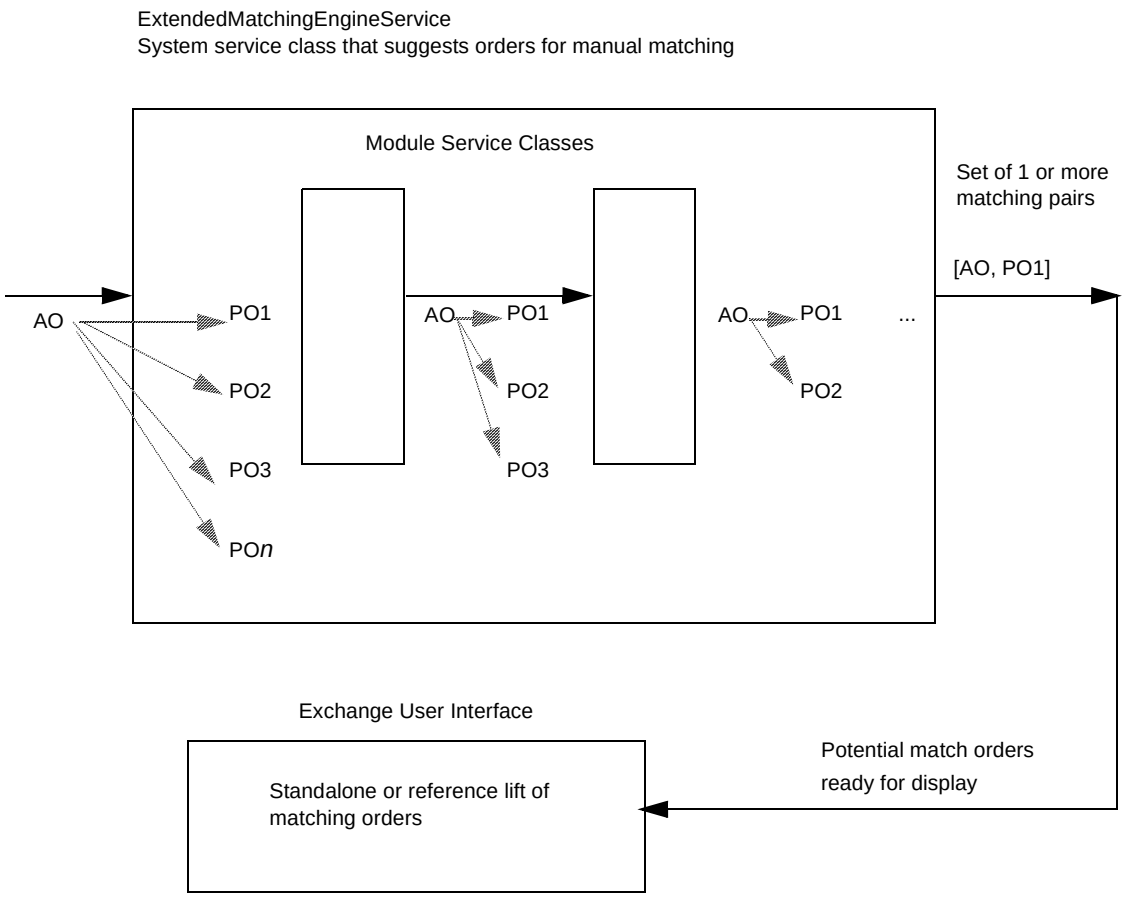
If you specify a quantity in your lift order that is less than the quantity of the symbols available in the exchange, the remaining quantity is returned to the exchange as a new active order with the original order number.

Reference Lifting

In a *reference lift*, passive orders are also displayed in the Exchange module user interface. However, the Exchange module does not automatically create an opposing matching order. Instead, you specify a *reference order* to lift in relation to a list of passive orders. For example, if your reference order is a buy order, you must lift a passive sell order.

A reference lift also lets you determine, per order, whether any quantity remaining from the lifted order is to be canceled or returned to the exchange as an active order. See Figure 3-3.

Figure 3-3 Potential Manual Order Matches



Legend

AO	Active Order
PO	Passive Orders, which can potentially match an active order

Table 3-3 shows an example of system and module service classes. To see a similar example, open the `BasicSchema.xml` or `TransportSchema.xml` file in the *<ias_inst_dir>*`/ias/APPS/imm40/imm40/WEB-INF/classes/exchange` directory (where *<ias_inst_dir>* is the location where you installed your iPlanet Applications Server software) in a text editor and search for MatchingEngineService.

NOTE In code, the ellipses (...) mean that some syntax is omitted.

In Table 3-3, MatchingEngineService is a system service class and NVPMatchingModuleService is a module class. ExtendedMatchingEngineService is a system service class and BallonServiceImp is a module service class. Each module and system service has an implementation class name and a set of service properties. Each service property has a name and value.

Table 3-3 Matching Engine Services

```
<SERVICE name="MatchingEngineService"
impl_class="com.iplanet.ecommerce.vortex.exchange.service.impl.MatchingEngineS
erviceImpl">
    <SERVICE_PROPERTIES>
       <SERVICE_PROPERTY name="SystemMatchingEnabled" value="true"/>
        <SERVICE_PROPERTY name="SystemMatchingModuleServiceKey.0"
value="NVPMatchingModuleService"/>
...
</SERVICE>
<SERVICE name="NVPMatchingModuleService"
impl_class="com.reference.energyb.service.NVPMatchingModuleServiceImpl">
        <SERVICE_PROPERTIES>
        <SERVICE_PROPERTY name="NVPAxisName.0" value="PriceAxis" />
        <SERVICE_PROPERTY name="NVPAxisName.1" value="QuantityAxis"/>
    <SERVICE_PROPERTY name="NVPAxisName.2" value="CapacityAvailableDateAxis"/>
    </SERVICE_PROPERTIES>
</SERVICE>
...
<SERVICE name="ExtendedMatchingEngineService"
impl_class="com.iplanet.ecommerce.vortex.exchange.service.impl.ExtendedMatchin
gEngineServiceImpl">
    <SERVICE_PROPERTIES>
        <SERVICE_PROPERTY name="MatchingModuleServiceKey.0"
value="NVPExtMatchingModuleService"/>
        <SERVICE_PROPERTY name="MatchingModuleServiceKey.1"
value="BlackListMatchingService"/>
        <SERVICE_PROPERTY name="BalloonServiceKey" value="BalloonService"/>
        <SERVICE_PROPERTY name="message.topic.extended_matches"
value="extended_matches"/>
    </SERVICE_PROPERTIES>
</SERVICE>
```

Table 3-3 Matching Engine Services *(Continued)*

```
...

<SERVICE name="BalloonService"

impl_class="com.iplanet.ecommerce.vortex.exchange.service.impl.BalloonServiceI
mpl">
    <SERVICE_PROPERTIES>
        <SERVICE_PROPERTY name="BalloonFunction"
value="com.reference.energyb.balloon.BalloonImpl" />
    </SERVICE_PROPERTIES>
</SERVICE>
```

Defining Trading Rules Services

If your exchange has a price axis, you must define trading rules to convert matched orders to traded orders. The TradeEngineService system service class implements the logic for the trading rules. You can use the sample PricingTradeModuleServiceImp module service class as a template, or create your own trading rules in a different class. See Table 3-4.

Table 3-4 Trade Engine Services

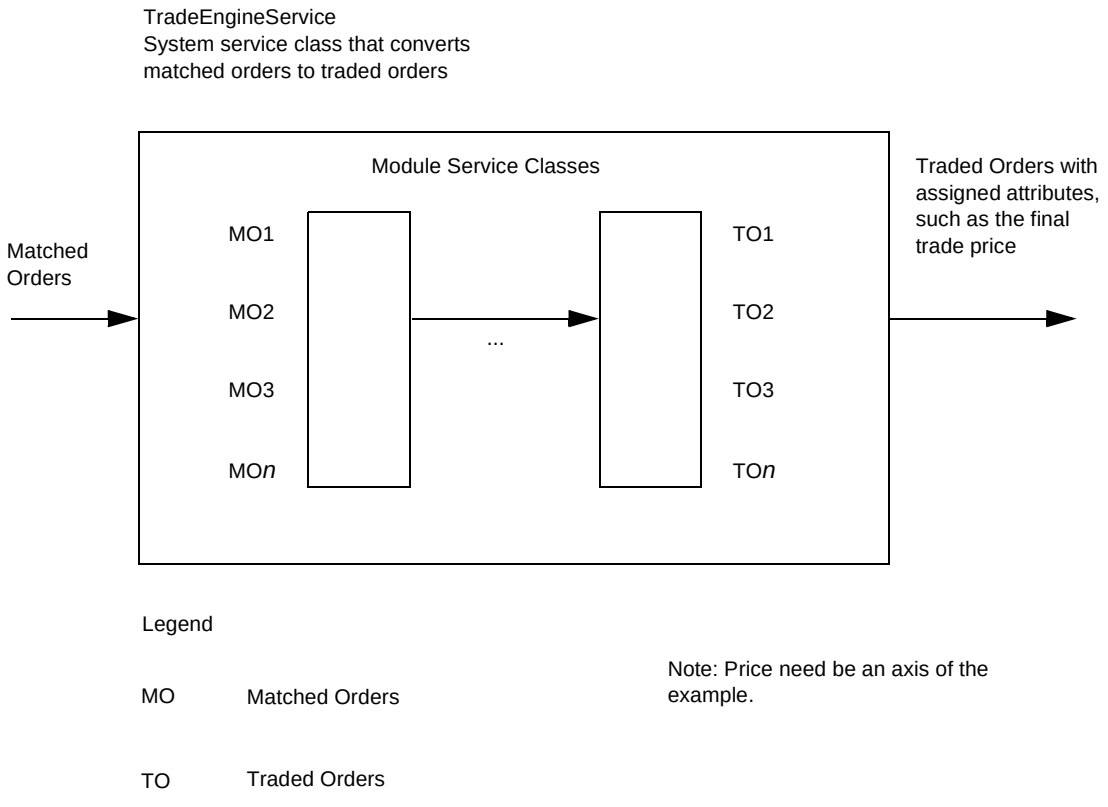
```
<SERVICE name="TradeEngineService"

impl_class="com.iplanet.ecommerce.vortex.exchange.service.impl.
TradeEngineServiceImpl">
    <SERVICE_PROPERTIES>
        <SERVICE_PROPERTY name="TradeModuleServiceKey.0"
value="PriceTradeModuleService"/>
        <SERVICE_PROPERTY name="message.topic.trades" value="trades" />
    </SERVICE_PROPERTIES>
</SERVICE>
...
</SERVICE>
<SERVICE name="PriceTradeModuleService"
impl_class="com.reference.transport.service.PricingTradeModuleServiceImpl">
    <SERVICE_PROPERTIES>
        <SERVICE_PROPERTY name="PricingAxisName" value="PriceAxis" />
    </SERVICE_PROPERTIES>
</SERVICE>
```

You specify trading rules to assign attributes to matched buy and sell orders. A final trade price is a typical type of attribute. The trading rules follow a model similar to that of the matching rules shown in Figure 3-2, except that there is no refining, or *pruning*, of matched orders. Every matched order becomes a traded order. See Figure 3-4. The difference is that traded orders have attributes that specify the basis for settling trades, such as price, or whatever is relevant to the requirement of the exchange.

Example

Suppose that as a result of the matching rules you defined, there is a matching order with a buy price of $8 and a sell price of $10. The Exchange module needs a trading rule to determine whether the final trade price is $8, $10, or some price in between the two.

Figure 3-4 Matched and Traded Orders



Chapter 4

Integrating Exchange with an External System

The iPlanet Market Maker Exchange module can be customized to integrate with an external module, such as a third-party application or another iPlanet Market Maker module. As an example, this chapter explains how to integrate the iPlanet Market Maker order management system (OMS) with the Exchange module.

This chapter contains the following sections:

- Overview
- Implementing the Integration
- Error Recovery

Overview

Rather than presenting general instructions for integrating the Exchange module with external applications, the instructions in this chapter will address a specific example: integrating the Exchange module with the order management system (OMS) of iPlanet Market Maker. By using the example instructions as a guideline, you will be able to implement the integration of the Exchange module with any downstream application such as trade settlement, portfolio management, or any legacy order management systems within the buyer and seller companies.

A trade consists of a single active order, matched with one or more (buy / sell) orders. This pairing of a buy order and a matching sell order is called a *trade pair*. A trade can contain one or more trade pairs. For each trade pair, a requisition (or multiple requisitions) is created in OMS that contains a single line item representing the trade, or purchase. The requisition is submitted to an approval

process within the buyer company. If the requisition is approved by the buyer company, the order is submitted to the seller company for further processing. All the buy-side information is collected from the buy order in the trade pair; all the sell-side information is collected from the sell order in the trade pair.

The order processes that occur after these trades happen are external to the Exchange module. These OMS processes will need to be integrated with the Exchange module. In the out-of-box iPlanet Market Maker software, two reference exchanges for directing the buy orders in a trade to the OMS have been implemented:

- Basic reference exchange
- Transport reference exchange

With the help of the iPlanet Messaging Queue, exchange events are published by MOMPublisherService for each exchange service. These events are defined as SERVICE_PROPERTYs. A MOM_LISTENER picks up these published messages under the topic *trades* and calls the API of the BasicExchangeAsyncManager. This API in turn invokes the corresponding service (in this case, the OMS service) and the service performs the business function.

Implementing the Integration

To implement the Exchange module integration with OMS, the following tasks are required:

1. Creating a service (java class) that will invoke the appropriate OMS module API to create requisitions
2. Defining the new service as a service in the XML schema for the exchange you are implementing
3. Writing a MOM listener that will listen to the topic *trades*
4. Defining the listener as a MOM_LISTENER in the XML schema for the exchange you are implementing
5. Adding a method to the public class BasicExchangeAsyncManager and the remotable framework to which the listener would make calls

NOTE The above Listener / AsyncManager / Service pattern can be used in a generic manner to integrate the Exchange module with any external or third party application.

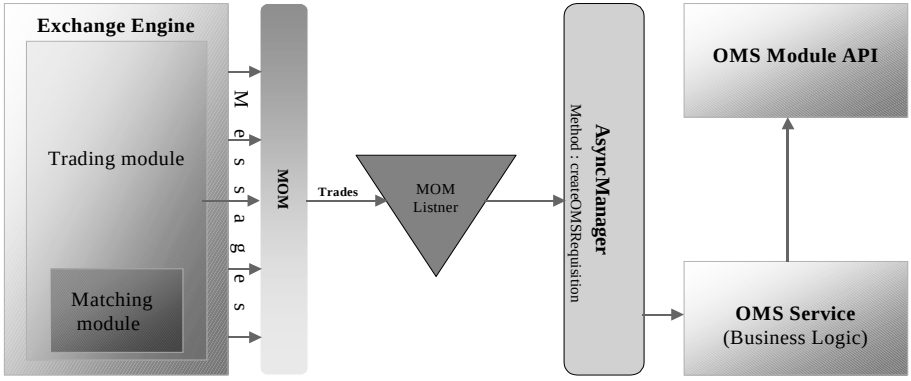
Basic Reference Exchange

The Basic reference exchange implements a listener for the *trades* topic in the Exchange module. Each time a trade takes place, the listener picks up the message corresponding to the *trades* topic and calls a method (business logic) that invokes the OMS module APIs to create a requisition.

For this purpose, a message oriented middleware (MOM) listener, OMSMOMListener, has been created and defined in the XML schema of the Basic reference exchange. The listener invokes the API of the BasicExchangeAsyncManager, which then delegates the call to the OMSService (OMSServiceImpl) where the required business functions are performed.

If the OMS module was implemented correctly, the requisition is submitted for approval according to the OMS module logic.

The following figure illustrates the integration of a Basic reference exchange with OMS.



Sample Code

Table 4-1 shows sample code for the OMS service. This service contains the business logic and calls the OMS module APIs to create a new requisition for each trade pair when the trade takes place. This service is defined in the XML schema descriptor file, BasicSchema.xml, as a SERVICE with quantity and price axes as the SERVICE_PROPERTYs.

Table 4-1 Sample OMSServiceImpl.java

```
..
requisition.setSourceGuid(buyOrder.getGuid());
requisition.setSourceType(OrderSource.EXCHANGE);
requisition.setSourceDisplayableId(buyOrder.getTrackingNumber();
LineItem item = omsManager.newRequisitionLineItem();
item.setVendorGuid(sellOrderRequest.getCompany());
item.setSourceGuid(symbolGuid);
item.setDescription(symbol.getDescription());
item.setQuantity((new Long(tradeUnitQuantity)).intValue());
item.setPrice(new Price(tradeUnitPrice));
item.setNegotiatedPrice(new Price(tradeUnitPrice));
item.setUnitOfMeasure("EA");
item.setDisplayableId(symbol.getName());
requisition.setOwningCompany(buyOrderRequest.getCompany());
requisition.setOwner(buyOrderRequest.getUser());
requisition.setCreator(buyOrderRequest.getUser());

requisition.setDescription("This requisition is for Trade
ID="+tradeID
+" ,Buy Order Tracking
Number="+buyOrder.getTrackingNumber()
+" ,Sell Order Tracking
Number="+sellOrder.getTrackingNumber());
requisition.addLineItem(item);
requisition.store();
boolean  isOMSRegistered =
ModuleList.isModuleRegistered(OMSModule.getModuleCode()) ;
               if(isOMSRegistered){
                        requisition.submitForApproval();
……
```

Table 4-2 shows sample code is for the OMS MOM listener. This listener is defined in the XML schema descriptor BasicSchema.xml as a MOM_LISTENER which would pick up all the messages in the *trades* topic of the Basic *reference* exchange.

Table 4-2 Sample OMSMOMListener.class

```
public class OMSMOMListener extends ExchangeMOMListener {
/**
     * Processes a TradeMOMMessage message to create an OMS
Requisition.
     * <P>
     * It calls the
BasicExchangeAsyncManager.createOMSRequisition(String tradeID)
     * method.
     * <P>
     *
     * @param msg the TradeMOMMessage to process.
     * @throws JMSException if message retrieval fails
     * @throws VortexException thrown if the processing of the
message fails.
     */
public void processMessage(Message message) throws
        JMSException, VortexException {
ObjectMessage objMessage = (ObjectMessage) message;
        TradeMOMMessage tradeMsg = (TradeMOMMessage)
MOMUtils.unwrapObject((byte[])objMessage.getObject());
Context ctx = TLContext.getImmContext();
        BasicExchangeAsyncManager asyncMgr =
BasicExchangeAsyncManager.getBasicExchangeAsyncManager(ctx);

 asyncMgr.createOMSRequisition(tradeMsg.getTradeId());
    }
}
```

Table 4-3 shows the sections of the XML schema descriptor file where the OMS service and the OMS MOM listener are defined.

Table 4-3 Sample BasicSchema.xml File

```
<SERVICE name="OMSService"
impl_class="com.reference.basic.service.OMSServiceImpl">
<SERVICE_PROPERTIES>
<SERVICE_PROPERTY name="PriceAxisName" value="PriceAxis" />
<SERVICE_PROPERTY name="QuantityAxisName" value="QuantityAxis" />
</SERVICE_PROPERTIES>
</SERVICE>

<MOM_LISTENER name="OMSMOMListener"
impl_class="com.reference.basic.mom.OMSMOMListener">
<MOM_LISTENER_PROPERTIES>
<MOM_LISTENER_PROPERTY name="active" value="false"/>
<MOM_LISTENER_PROPERTY name="debug" value="error"/>
<MOM_LISTENER_PROPERTY name="message.topic" value="trades" />
</MOM_LISTENER_PROPERTIES>
</MOM_LISTENER>
```

Error Recovery

If errors, such as lack of privileges, are encountered while creating the requisitions in the OMS module, the MOM message designated for the OMS listener is stored by the Exchange Module Recovery Mechanism, and a notification of the error is sent to the iPlanet Market Maker administrator as well as to all the account holders of the buy order. The iPlanet Market Maker administrator would then replay the message once the problem is resolved.

NOTE By default, the OMS listener is set to Off in the exchange XML schema. To view the requisitions that have been created in the OMS module, the company that owns the buy order needs to be assigned as a buyer company (or both, buyer and seller company) in the iPlanet Market Maker community, with appropriate OMS privileges.

Appendix A

MOM Listener and JMS Tags

The message oriented middleware (MOM) listener and the Java message service (JMS) tags are in the sample BasicSchema.xml and TransportSchema.xml schema files that come with the iPlanet Market Maker software.

MOM Listeners Tag

The MOM_LISTENERS tag specifies the list of registered JMS listeners in the Exchange module. The Exchange module uses the publish/subscribe JMS paradigm. Each registered listener subscribes to the topic of interest. By default, these listeners are internal exchange services, but you can configure additional listeners and integrate them to external systems as part of customizing the process for a new exchange.

Each listener entity has the following XML structure:

```
<MOM_LISTENER name="[name of the listener]" impl_class="[implementation class
name]">
     <MOM_LISTENER_PROPERTIES>
          <MOM_LISTENER_PROPERTY name="debug" value="[debug level to set for
the listener]"/>
          <MOM_LISTENER_PROPERTY name="message.topic" value="[name of the
topic to which the listener subscribes to]" />
     </MOM_LISTENER_PROPERTIES>
</MOM_LISTENER>
```

The MOM listener implementation class should extend the com.iplanet.ecommerce.vortex.exchange.mom.ExchangeMOMListener class provided by the Exchange module API and implement the processMessage method.

JMS Tag

The JMS tag specifies all the configuration information needed to enable the Exchange module for the JMS implementation being used, such as the iPlanet Message Queue. The ENCRYPT tag specifies the XML subtree that needs token replacement of values for elements such as user name, password, and so on.

```
   <JMS>
      <JMS_PROPERTIES>
         <JMS_PROPERTY name="jndi.context.factory"
value="com.sun.jndi.ldap.LdapCtxFactory"/>
         <JMS_PROPERTY name="jndi.url" value="ldap://localhost:389/o=imm"/>
      <ENCRYPT>
         <JMS_PROPERTY name="jmq.admin.user"     value="admin"/>
         <JMS_PROPERTY name="jmq.admin.password" value="admin"/>
         <JMS_PROPERTY name="jndi.authentication"  value="simple"/>
         <JMS_PROPERTY name="jndi.principal" value="cn=Directory Manager"/>
         <JMS_PROPERTY name="jndi.credentials" value="dirmanager"/>
         <JMS_PROPERTY name="jms.exchange.user" value="crs"/>
         <JMS_PROPERTY name="jms.exchange.password" value="crs"/>
      </ENCRYPT>
         <JMS_PROPERTY name="jms.host" value="localhost"/>
         <JMS_PROPERTY name="jms.port" value="7676"/>
         <JMS_PROPERTY name="routers.jvm.id" value="router"/>
         <!-- this should go into MOM_PROPERTIES -->
         <JMS_PROPERTY name="max.concurrent.listeners" value="5" />
         <!-- unack messages threshold for the router listeners -->
         <JMS_PROPERTY name="unack.messages.stop.limit" value="1000"/>
         <JMS_PROPERTY name="unack.messages.resume.limit" value="100"/>
      </JMS_PROPERTIES>
   </JMS>
```

Index

F

G

H

I

J

L

M

N

O

P

R

S

T

U

W

X