Customization Guide

iPlanet Market Maker

Version 4.5

March 2002

Contents

List of Figures

List of Tables

About This Document

The Customization Guide provides guidelines and instructions for customizing iPlanet Market Maker.

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 iPlanet Market Maker.

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"

Table 1 Chapter Summary *(Continued)*

If you want to know about this	See this chapter
How to customize the look and feel of the pages	Chapter 1, “Customizing Web Pages”
How the contents of the iPlanet Market Maker portal page is derived	Chapter 2, “Customizing Display Profiles”
How to customize the functionality of the modules	Chapter 3, “Customizing Modules”
Sample code used to customize the Auction module	Chapter 4, “Customizing the Auction Module”
Sample code used to customize the OMS module	Chapter 5, “Customizing the OMS Module”
How to use iPlanet Market Maker module pluggability	Chapter 6, “Module Pluggability”
iPlanet Market Maker properties that you can configure	Appendix A, “Configurable Property Names”

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 this document 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

Customizing Web Pages

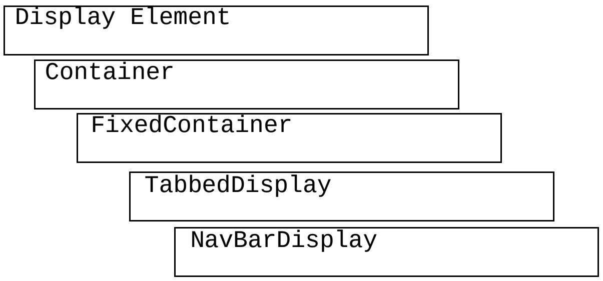
This chapter contains the following sections:

- Vertical Navbar
- Conclusion
- Customizing the Appearance of Web Pages

Vertical Navbar

The vertical navbar is an iPlanet Market Maker display element. Its class is `NavBarDisplay`. It is a specialization of `TabbedDisplay`. Normally, `TabbedDisplay` shows its tabs as links at the top of the display. `NavBarDisplay` instead shows a hierarchical menu at the left of the content, plus an optional breadcrumb on top.

The hierarchy of classes is:



In iPlanet Market Maker, the main container for the application is a `TabbedDisplay`, with the setting `notabs` set to *true* in the `DisplayProfile`, so that the tabs are not generated.

Instead, the main jsp, `imm.jsp`, instantiates the `TabbedDisplay` and generates a DHTML/javascript menu.

NavBarDisplay can be used instead of TabbedDisplay to achieve a different presentation for the main page.

Architecture

Classes

To use the NavBarDisplay, a few classes are necessary:

The package `com.iplanet.ecommerce.vortex.display` contains the following classes:

- NavBarDisplay is the display element

  To produce the HTML for the pbreadcrumb (PathBar) and the navbar, it uses:

  - SimpleTable
  - SideNavigationProducer
  - PathBarProducer

- `ActioncodeMap` manages the action code mapping feature:

  - JumptoLinkGen
  - PostAuthenticationHook

    These are not used for now.

These classes then use a few properties files for configuration. Contrary to the other properties files read in iPlanet Market Maker, the navbar properties files use a new enhanced ResourceManager which consists of the following classes in the `com.iplanet.ecommerce.vortex.util package`:

```
XmlDocumentInstance.java
ResourceManagerLauncher.java
ResourceManager.java
ResourceInstance.java
PropertyHelper.java
PropertiesInstance.java
```

This ResourceManager is an improvement on the old resources, in that when a config file is modified, you do not have to restart the server in order for it to be picked up.

JSPs

There are three JSPs in `infrastructureD\www\jsp`:

```
immnav.jsp
NavBarHeader.jsp
NavBarFooter.jsp
```

In order for the iMM installation to have both: the `TabbedDisplay` and the `NavBarDisplay` available, the `NavBarDisplay` container is used within another jsp:

`immnav.jsp`.

This jsp can co-exist with `imm.jsp`.

It is only an example, and it can be used as a base for your own main JSP.

If you customize it though, be sure to keep the two includes:

`NavBarHeader.jsp`

and

`NavBarFooter.jsp`

These are important, because they open and close tables that are assumed by the `NavBarDisplay` class.

Doing this allows us to remove one level of table embedding, which is critical for rendering performances on Netscape 4.7.

Display Profile

There must be a new *PAGE* element in the imm display profile:

<PAGE NAME=`/immnav.jsp`>

This element contains the same sub-elements as the `imm.jsp` page element, but it has different settings of which the `NavBarDisplay` can take advantage.

This display profile is found in `tools/etc/navbarload`. It is called `newdp.xml` and is based on the iPlanet Market Maker 4.0 SP1 display profile.

If you have customized your display profile, reapply your customizations to `newdp.xml`, both under the `imm.jsp` page element and under the `immnav.jsp` page element.

Then load it using `./install`.

NOTE The only different aspects of the DP structure in the `imm.jsp` page and in the `immnav.jsp` page are the Neg elements.

For each main menu title <*Module*>, Auction, Community, and so forth, there is a `Neg`<*Module*> avail item, and an associated Member item.

In the `imm.jsp` page, these avail items have the contentmask

`!Anonymous,!mod_`<*Module*>

This means that they are never rendered, because we do not need these when we generate the DHTML / Javascript menubar. The reason for this is that the top level menus never target real channels. However, they do target javascript functions that render the corresponding sublevel menu.

When we use the navbardisplay, we do need these top level menus to be rendered. These avail elements for `Neg`<*Module*> define `FileContent` elements, and the corresponding Members define the root and filename for these files.

By default these are `imm`/<*module*>`.html`.

However, because we need these channels to be rendered, we have to modify the contentmask to

`!Anonymous,mod_`<*Module*>

That is the only difference in the `immnav.jsp profile`.

```
<AVAILITEM NAME="NegAuctions" DESCRIPTION_KEY="0"
CLASS="com.iplanet.ecommerce.vortex.display.FileContent"
DEFCODE="-1" DEFINDEX="0" CONTENTMASK="!Anonymous,mod_auction" />
```

To target a page other than one of the default pages, modify the corresponding MEMBER elements in the `immnav.jsp` page of the Display Profile:

```
<CONTENT ITEMNAME="NegOMS" IDNUMBER="100" COORD="1,0,0">
        <SETTINGS>
          <BOOLEAN NAME="NoMenuBar" VALUE="true" />
          <STRING NAME="ContentFile"
VALUE="your_file_name_here.html" />
          <STRING NAME="FileRoot" VALUE="imm" />
        </SETTINGS>
```

Configuration Files

In `navbar\resources` there are a few configuration files that are relevant to configure the navbar:

- `navbar.xml` is used to configure the content of the navbar, based on the display profile.
- `actioncodemap.xml` is used to configure an advanced feature: the navbar elements can appear conditionnaly based on what is currently displayed in the container.
- `NavBarDisplayStrings.properties` is used to customize the look and feel of the navbar.
- `ResourceManager.xml` is used to configure the resourcemanager.
- The DTDs for the files `navbar.xml` and `actioncodemap.xml` are:

  `navbar.dtd`

  `actioncodemap.dtd`

The default values in the files as shipped should be enough to test the navbar. To customize it further, see the customization section.

Installation

In tools/etc/navbarload, there is a new display profile, which contains the entries necessary for the navbar. If you cd to that directory and run install, this new display profile will be loaded. Once it is complete, you can go to `immnav.jsp` in your browser and play with the new navbar. If you want to go back to the iMM default display profile, without the navbar features, run uninstall in tools/etc/navbarload, and the default display profile will be installed.

Test it!

To see if the product is installed correctly, point your browser to:

`http://<host>:<port>/imm/imm.jsp`

If everything is okay, then point the browser to:

`http://<host>:<port>/imm/immnav.jsp`

You should get the new container with the navbar on the left and the breadcrumb.

Login as `immhost/root/root` and check that the navbar behaves as expected.

Navbar Customization

Configure the ResourceManager

On the first load of the NavbarDisplay, a Class called `ResourceManagerLauncher` starts a thread that reads in and preloads all the Resource files specified within the `ResourceManager.xml` file. If any changes are made to any of these property files at run time, the thread launched by `ResourceManagerLauncher` reads in the changes and reloads the changed version of the Resource file used by the navbar. This is how we can change, for example, the look and feel of the Navbar on the fly!

NOTE Currently the sleep interval of the thread which reads the property files (if changed) is set to one second. There is no configurable property to set the sleep to some other value

The `ResourceManager.xml` has a main tag <*ResourceManager*> which contains one or more <*resource*> elements.

A resource looks like this:

```
<resource name="navbar"
            class="com.iplanet.ecommerce.vortex.util.XmlDocumentInstance"
   file="navbar.xml"
/>
```

The name attribute is the String used in the java class that uses this resource to identify it. The navbar component uses three resources:

- navbar

- actioncodemap
- navbardisplay

The class attribute is the class used to load the resource. There are three of them in the util package now, but you can add your own if you use this facility in your own class:

```
XmlDocumentInstance for XML documents
ResourceInstance for resourcebundles
PropertiesInstance for property files
```

The file attribute is the name of the file to be loaded. The file must be in the classpath of the application.

Configure the Navbar Content

`navbar.xml` allows you tp configure the content of the navbar. It follows the `navbar.dtd` DTD. Refer to it for the exact syntax. `navbar.xml` is the example that ships with the navbar.

`sunbidnavbar.xml` is a more complex example that shows a lot of the possibilities of the syntax.

The <*navbar*> element can contain the different elements that constitute the navbar.

Most of these elements allow the optional attributes: DisplayIf and ContentMask.

These attributes enable an additional level of customization and filtering to be applied to the navbar.

Filtering attributes

The DisplayIf Attribute

When a navbar element has the attribute DisplayIf set to a String, this String is looked up in `actioncodemap.xml` where a list of action codes is associated to that name. Then the navbar element is only displayed if the container displays the page associated to one of these action codes.

This can be used to produce context-sensitive elements in the navbar, that appear depending on the page you are viewing.

This very powerful mechanism is not used in the default `navbar.xml` that we ship with the component, but you can use it when you customize your navbar.

Example

```
<vortexSection idNumber="10004"
displayIf="Auction.Initiator.View"/>
```

The section will be displayed only when the page Auction.Survey is displayed in the container. Auction.Survey is a list of action codes defined in `actioncodemap.xml`.

The ContentMask Attribute

When a navbar element has the attribute ContentMask set to a String, the regular iPlanet Market Maker display profile ContentMask mechanism is applied to the navbar element. For example, only the users having the credentials required by the ContentMask will see this attribute.

This is useless for elements that come from the DisplayProfile where the ContentMask is already applied, but can be useful for the other types of elements (spacers, relatedSection, seeAlsoSection).

Example

```
<bigSpacer contentMask="buyer|seller|marketmaker"/>
```

The spacer is displayed only for buyer, seller and marketmaker, but not, for example, for anonymous user.

Spacers

There are four types of spacers. Use them to vertically separate out your navbar sections.

```
spacer
bigSpacer
lastSpacer
twoColumnSpacer
```

RelatedSection and seeAlsoSection

The content of these navbars will appear in special *Related* and *See Also* sections at the end of the navbar. This is a common layout used by many sites. However, you are not obliged to use it.

ImageRow

This displays an image on the whole row of the navbar.

Example

```
<imageRow
image="/@IMM_DOCROOT@/images/AdBanners/poweredbyiplanet.gif"
              alt="More About iPlanet Market Maker"

link="http://www.iplanet.com/products/iplanet_market/home_2_1_1a
.html"
              newWin="pimm"/>
```

Row

This displays text with a link.

- link attribute is the link.
- newWin is the name of the popup window to be created if you want the link to generate a popup.

Example

```
<row
link="/@IMM_DOCROOT@/immnav.jsp?VDSP_authenticationAction=logout"
     contentMask="buyer|seller|marketmaker">Logout</row>
<row link="http://store.sun.com">Sun Store (U.S. Only)</row>
<row link="http://www.sun.com/auctions/sunbid/faq-rules/"
newWin="sbr">Sun Bid
```

vortexSection

This is the most important element of the navbar. It displays a section of the iPlanet Market Maker DisplayProfile. The attribute is either idNumber or series.

- *idNumber:* the DisplayElement with the specified IDNUMBER is looked up in the DisplayProfile and displayed in the navbar.
- *series:* the series of idnumbers in this 100 series will be displayed, for example, the whole menu.

Example

```
<!-- Anonymous portal -->
<vortexSection idNumber="10003"/>
<spacer contentMask="Anonymous"/>
<!-- Market Place -->
<vortexSection idNumber="0"/>
<!-- OMS -->
<vortexSection series="100"/>
<!-- RFx -->
```

If you use the DisplayIf attribute in `navbar.xml`, the String value of this attribute must be defined in `actioncodemap.xml`. This file follows the `actioncodemap.dtd` DTD.

Each page element defines a logical name that can be used in `navbar.xml`. This logical name is associated to a list of action codes.

These symcodes are defined in java classes. A list of symcodemap elements lists all the classes that contain the definition for the action codes of the application. If you add your own action codes to iPlanet Market Maker, please add a symcodemap element referring to the class where your action codes are defined.

Example

```
<symcodemap
class="com.iplanet.ecommerce.vortex.arch.display.BaseDisplayModu
le"/>
<symcodemap
class="com.iplanet.ecommerce.vortex.auction.display.AucJSP"/>
    <page name="Auction.Initiator.View">
        <symcode value="ACT_INITIATOR_VIEW_BROWSE"/>
        <symcode value="ACT_INITIATOR_VIEW_SEARCH"/>
...
<symcode value="ACT_INITIATOR_VIEW_CANCELAUCTION"/>
</page>
```

Configure the Navbar Look and Feel

Finally, `NavBarDisplayStrings.properties` defines the look and feel of the navbar (for example, some pieces of HTML, with references to styles and images that will be used to generate it).

Refer to the comments in the file to see what these values mean.

Because it is instantiated by the ResourceManager, it is easy to test new look and feel by changing the file on the server directly and reloading your page to see the effect.

Configure the Settings of the Container in the Display Profile

One last way you can configure the navbar is through the settings element in the display profile for the top container in the `immnav.jsp` page element.

The important settings are:

- **NoTabs:** This should be false. If it is true the navbar will not be generated at all.
- **Pathbar:** If this is true, the breadcrumb on top of the navbar will be generated. Otherwise, it will not.

Example

```
<SETTINGS>

    <INTEGER NAME="ContentSpace" VALUE="0"/>
    <INTEGER NAME="ContentPad" VALUE="0"/>
    <INTEGER NAME="ContentBorder" VALUE="0"/>
    <STRING NAME="ContentColor" VALUE="#FFFFFF"/>
    <INTEGER NAME="ContainerBorder" VALUE="0"/>
    <STRING NAME="ContainerStyle" VALUE="BACKGROUND: #FFFFFF"/>
    <BOOLEAN NAME="NoMenuBar" VALUE="false"/>
    <STRING NAME="MenuBarColor" VALUE="#666699"/>
    <STRING NAME="MenuBarFontOpen" VALUE="&lt;FONT SIZE=2
COLOR=&apos;#FFFFFF&apos;&gt;&lt;B&gt;"/>
    <STRING NAME="MenuBarFontClose"
VALUE="&lt;/B&gt;&lt;/FONT&gt;"/>
    <STRING NAME="Orientation" VALUE="left"/>
    <BOOLEAN NAME="Pathbar" VALUE="true"/>
    <BOOLEAN NAME="NoTabs" VALUE="false"/>
    <BOOLEAN NAME="NoWrapper" VALUE="true"/>
```

```
<SETTINGS>

</SETTINGS>
```

Conclusion

With this documentation, you should be able to configure the navbar to solve your business needs.

Happy NavBar!

Customizing the Appearance of Web Pages

A digital marketplace usually has style standards to maintain a consistent look and feel for its web pages. For example, it is common for text fonts and button shapes to be used consistently across pages. The default iPlanet Market Maker pages might use different standards than your existing pages. If you want these pages to conform to your style standards, you can customize them. This section covers the following topics.

- "Customizing Text and Tables"
- "Customizing Buttons and Tabs"

This chapter covers only how you can customize the appearance of the pages. For information about how to customize the functionality of the modules, see Chapter 3, "Customizing Modules."

Customizing Text and Tables

To customize the appearance of text and tables, you edit the iPlanet Market Maker style sheets in the `@IMM_DOCROOT@/imm40/iMM` directory. Style sheets control the appearance of text, tables characteristics, and colors. Table 1-1 describes the style sheets.

Table 1-1 Style Sheet File Names

Style Sheet File Name	Used to Define
`m_styles.css`	Text and tables in market maker pages.

Table 1-1 Style Sheet File Names

Style Sheet File Name	Used to Define
`b_styles.css`	Text and tables in buyer pages.
`s_styles.css`	Text and tables in seller pages.
`x_styles.css`	Text and tables for pages in which the user role is not yet known.

Each of the style sheets defines similar characteristics. The main difference is that the colors vary depending on whether the user role is market maker, buyer, or seller. The following example shows a section of the `m_style.css` style sheet file.

```
...
TH /*For text at the top of a table heading column*/
{
   BACKGROUND: #CCCCCC;
   COLOR: #000000;
   FONT-FAMILY: Arial,Helvetica,Geneva,Swiss,SunSans-Regular;
   FONT-SIZE: 10pt;
   FONT-STYLE: normal;
   FONT-WEIGHT: bold
}
TD /*For text within a table cell*/
{
   FONT-FAMILY: Arial,Helvetica,Geneva,Swiss,SunSans-Regular;
   FONT-SIZE: 9pt;
   FONT-STYLE: normal
}
TABLE /*For the table background and ruling*/
{
   BACKGROUND: #CCCCCC;
   PADDING-BOTTOM: 0px;
   PADDING-LEFT: 0px;
   PADDING-RIGHT: 0px;
   PADDING-TOP: 0px
}
.pagetitle /*For the top and bottom bars of a table*/
```

```
{
   BACKGROUND: #CC6600;
   COLOR: #000000;
   FONT-FAMILY: Arial,Helvetica,Geneva,Swiss,SunSans-Regular;
   FONT-SIZE: 11pt;
   FONT-STYLE: normal;
   FONT-WEIGHT: bold;
}
```

Figure 1-1 shows an example of how some of the styles are applied.

Figure 1-1 Styles Used on a Sample Page

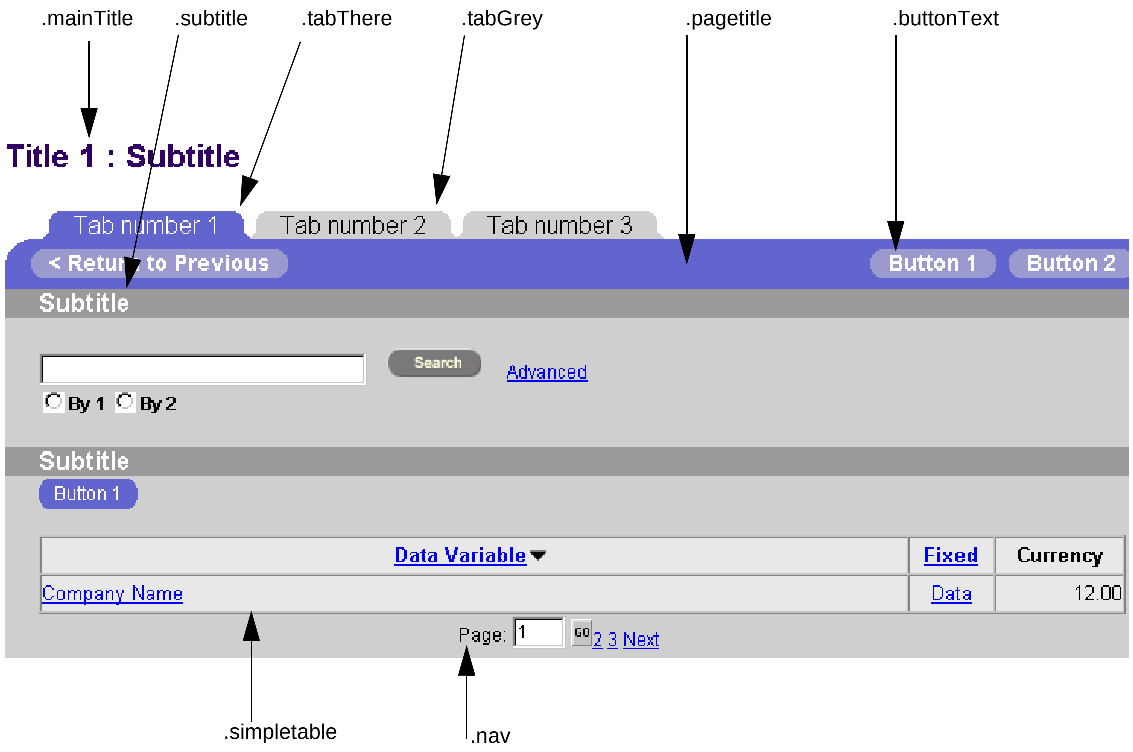


Figure 1-2 shows the same sample page with some style sheet changes.

Figure 1-2 Styles Changed from Figure 1-1

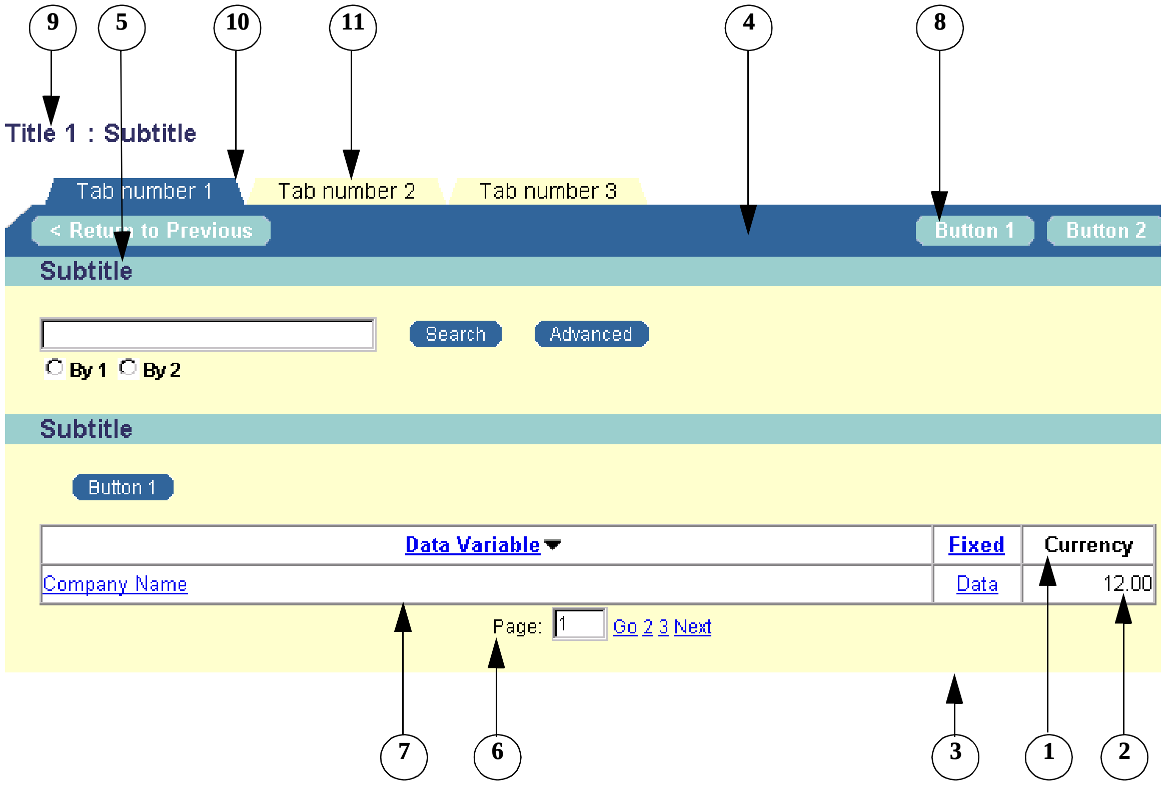


Table 1-2 shows the differences between the style sheets applied in Figure 1-1 and Figure 1-2. The numbers in the middle column of the table correspond to the numbers in Figure 1-2. The changes to the style sheet applied in Figure 1-2 have bullets to the left of the code in the right column.

Table 1-2 **Style Sheets used for Figure 2-1 and Figure 2-2**

Style Sheet for Figure 1-3		Style Sheet for Figure 1-4
`TH`		`TH`
`{`		`{`
`BACKGROUND: #CCCCCC;`	1	• `BACKGROUND: #FFFFCC;`
`COLOR: #000000;`		`COLOR: #000000;`

`FONT-FAMILY:Arial,Helvetica,`
`Sans-Serif;`
		• `FONT-FAMILY:Arial,Helvetica,`
		`Geneva,Swiss,SunSans-Regular;`

`FONT-SIZE: 10pt;`		`FONT-SIZE: 10pt;`
`FONT-STYLE: normal;`		`FONT-STYLE: normal;`
`FONT-WEIGHT: bold`		`FONT-WEIGHT: bold`
`}`		`}`
`TD`		`TD`
`{`		`{`

`FONT-FAMILY:`
`Arial,Helvetica,Sans-Serif;`
	2	• `FONT-FAMILY:Arial,Helvetica,`
		`Geneva, Swiss, SunSans-Regular;`

`FONT-SIZE: 9pt;`		`FONT-SIZE: 9pt;`
`FONT-STYLE: normal`		`FONT-STYLE: normal`
`}`		`}`
`TABLE`		`TABLE`
`{`		`{`
`BACKGROUND: #CCCCCC;`	3	• `BACKGROUND: #FFFFCC;`
`PADDING-BOTTOM: 0px;`		`PADDING-BOTTOM: 0px;`
`PADDING-LEFT: 0px;`		`PADDING-LEFT: 0px;`
`PADDING-RIGHT: 0px;`		`PADDING-RIGHT: 0px;`
`PADDING-TOP: 0px`		`PADDING-TOP: 0px`
`}`		`}`

Table 1-2 Style Sheets used for Figure 2-1 and Figure 2-2

Style Sheet for Figure 1-3	Style Sheet for Figure 1-4
`.pagetitle`	`.pagetitle`
`{`	4 `{`
`BACKGROUND: #6666cc;`	• `BACKGROUND: #336699;`
`COLOR: #000000;`	`COLOR: #000000;`
`FONT-FAMILY: Arial,Helvetica,Sans-Serif;`	• `FONT-FAMILY:Arial,Helvetica, Geneva,Swiss,SunSans-Regular;`
`FONT-SIZE: 11pt;`	`FONT-SIZE: 11pt;`
`FONT-STYLE: normal;`	`FONT-STYLE: normal;`
`FONT-WEIGHT: bold;`	`FONT-WEIGHT: bold;`
`}`	`}`
`.subtitle`	`subtitle`
`{`	5 `{`
`BACKGROUND: #999999;`	• `BACKGROUND: #99CCCC;`
`COLOR: #ffffff;`	• `COLOR: #333366;`
`FONT-FAMILY: Arial,Helvetica,Sans-Serif;`	• `FONT-FAMILY: Arial,Helvetica,Geneva,Swiss, SunSans-Regular;`
`FONT-SIZE: 14pt;`	`FONT-SIZE: 14pt;`
`FONT-STYLE: normal;`	`ONT-STYLE: normal;`
`VERTICAL-ALIGN: middle;`	`VERTICAL-ALIGN: middle;`
`FONT-WEIGHT: bold;`	`FONT-WEIGHT: bold;`
`TEXT-ALIGN: left;`	`TEXT-ALIGN: left;`
`}`	`}`

Table 1-2 Style Sheets used for Figure 2-1 and Figure 2-2

Style Sheet for Figure 1-3		Style Sheet for Figure 1-4
`.nav`		`.nav`
`{`	6	`{`
`BACKGROUND: #CCCCCC;`		• `BACKGROUND: #FFFFCC;`

`FONT-FAMILY:`
`Arial,Helvetica,Sans-Serif;`
		• `FONT-FAMILY:`
		`Arial,Helvetica,Geneva,Swiss,`
		`SunSans-Regular;`

`FONT-SIZE: 8pt;`		**`FONT-SIZE: 9pt;`**
`FONT-STYLE: normal;`		`FONT-STYLE: normal;`
`TEXT-INDENT: 2pt;`		`TEXT-INDENT: 2pt;`
`TEXT-ALIGN: center;`		`TEXT-ALIGN: center;`
`}`		`}`
`.simpletable`		`.simpletable`
`{`	7	`{`
`BACKGROUND: #E6E6E6;`		• `BACKGROUND: #FFFFFF;`
`}`		`}`
`.buttonText`		`.buttonText`
`{`	8	`{`

`FONT-FAMILY:`
`Arial,Helvetica,Sans-Serif;`
		`FONT-FAMILY:`
		`Arial,Helvetica,Sans-Serif;`

`FONT-SIZE: 11pt;`		• `FONT-SIZE: 10pt;`
`FONT-WEIGHT: bold;`		`FONT-WEIGHT: bold;`
`COLOR: #FFFFFF;`		`COLOR: #FFFFFF;`
`TEXT-DECORATION: none;`		`TEXT-DECORATION: none;`
`}`		`}`

Table 1-2 Style Sheets used for Figure 2-1 and Figure 2-2

Style Sheet for Figure 1-3		Style Sheet for Figure 1-4
.mainTitle		.mainTitle
{	9	{
FONT-FAMILY: Arial,Helvetica,Sans-Serif;		FONT-FAMILY: Arial,Helvetica,Sans-Serif;
FONT-SIZE: 16pt;		FONT-SIZE: 16pt;
FONT-WEIGHT: bold;		FONT-WEIGHT: bold;
COLOR: #330066;		• COLOR: #3333366;
BACKGROUND: #FFFFFF;		BACKGROUND: #FFFFFF;
}		}
.tabThere		.tabThere
{	10	{
BACKGROUND: #6666cc;		• BACKGROUND: #336699;
FONT-FAMILY: Arial,Helvetica,Sans-Serif;		FONT-FAMILY: Arial,Helvetica,Sans-Serif;
COLOR: #FFFFFF;		COLOR: #FFFFFF;
FONT-SIZE: 12pt;		• FONT-SIZE: 11pt;
TEXT-DECORATION: none;		TEXT-DECORATION: none;
}		}
.tabGrey		.tabGrey
{	11	{
BACKGROUND: #CCCCCC;		• BACKGROUND: #FFFFCC;
FONT-FAMILY: Arial,Helvetica,Sans-Serif;		FONT-FAMILY: Arial,Helvetica,Sans-Serif;
FONT-SIZE: 12pt;		• FONT-SIZE: 11pt;
COLOR: #000000;		COLOR: #000000;
TEXT-DECORATION: none;		TEXT-DECORATION: none;
}		}

Customizing Buttons and Tabs

iPlanet Market Maker comes with a default set of `.gif` image files that provide the curve effects for buttons and tabs. These files define the appearance of buttons and tabs. To customize the appearance of buttons and tabs, you create your own graphics in `.gif` files and rename these files to the corresponding defaults in the `images` directory and subdirectories. The purpose of these graphics is to overlay the left and right sides of tabs and buttons to create the effects you want.

Figure 1-3 shows an example of the graphics that determine the appearance of the tabs and buttons. For a description of each of the graphics, see Table 1-3.

Figure 1-3 Tab and Button Graphics

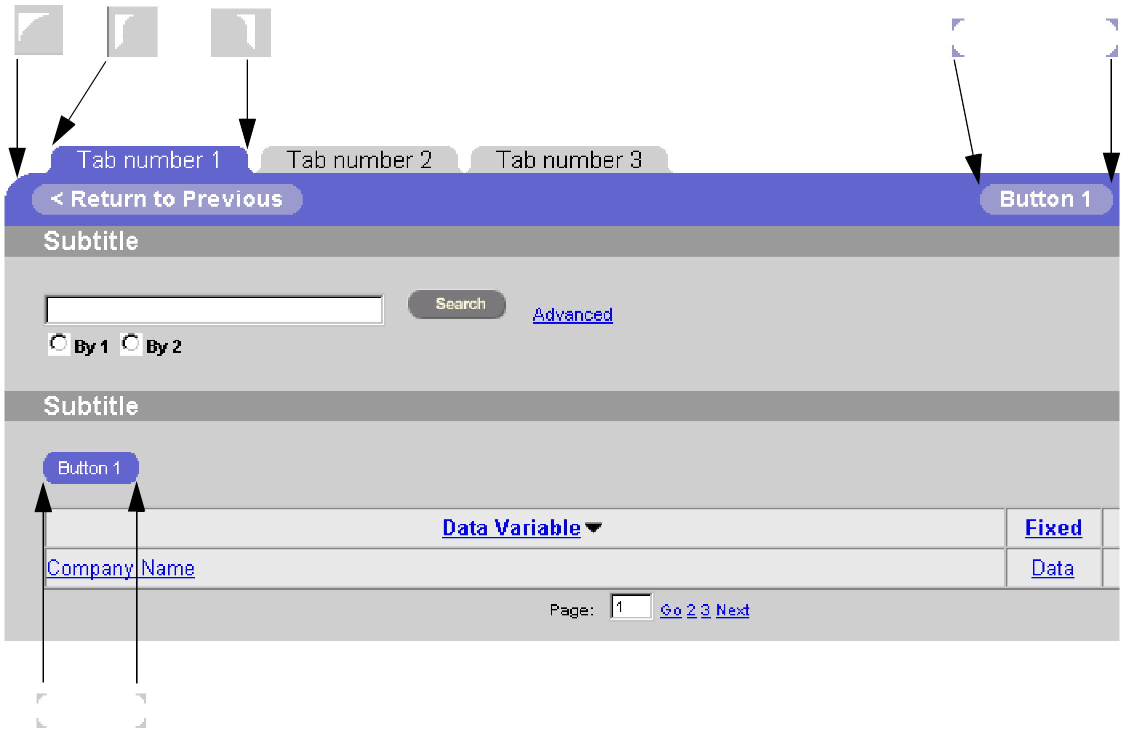


Table 1-3 Graphics Used in Figure 2-3

Image	File name	Used for
	`tab_left.gif`	Tab image to the left of the “Tab number 1” text.
	`tab_right.gif`	Tab image to the right of the “Tab number 1” text.
	`curve_left.gif`	Curve in the upper-left corner
	`s_left.gif`	Left curve for the “< Return to Previous” and “Button 1” buttons.
	`s_right.gif`	Right curve for the “< Return to Previous” and “Button 1” buttons.
	`left_sm.gif`	Left curve for the “Search” and “button 1” buttons.
	`right_sm.gif`	Right curve for the “Search” and “button 1” buttons.

To create a new look and feel for the tabs and buttons, use different graphics with the same file names as the default graphics. See Figure 1-4 and Table 1-4.

Figure 1-4 Alternative Tab and Button Graphics

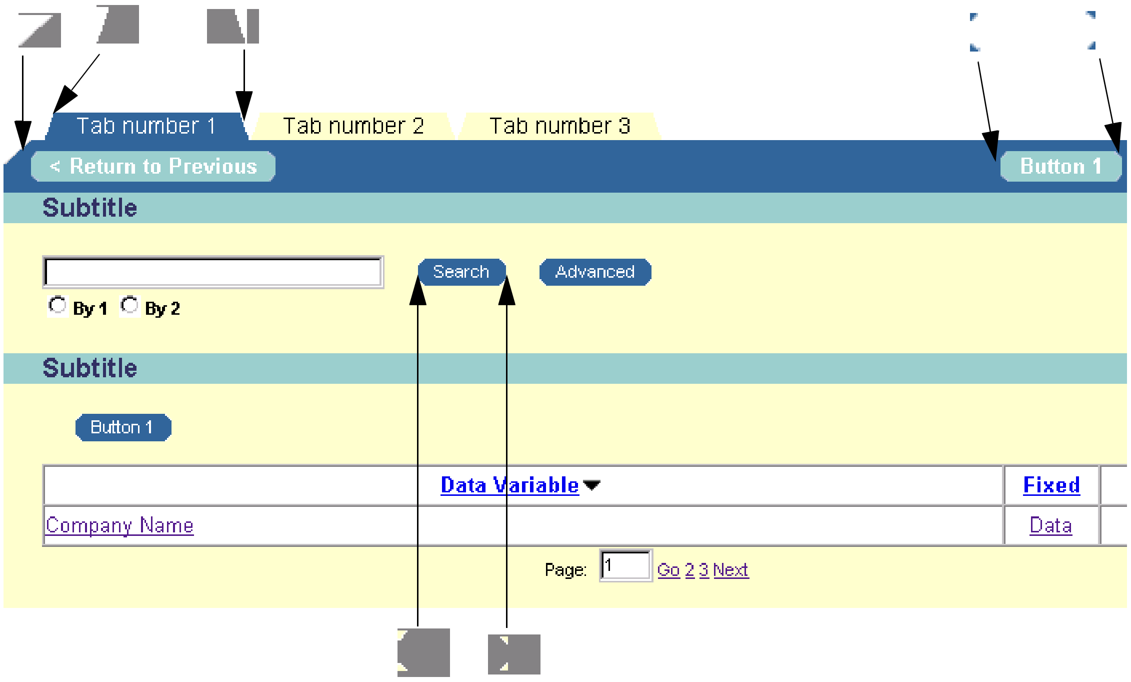


Table 1-4 Alternative Graphics Used in Figure 2-4

Image	File name	Used for
	`tab_left.gif`	Tab image to the left of the “Tab number 1” text.

Table 1-4 Alternative Graphics Used in Figure 2-4 *(Continued)*

Image	File name	Used for
	`tab_right.gif`	Tab image to the right of the "Tab number 1" text.
	`curve_left.gif`	Curve in the upper-left corner
	`s_left.gif`	Left curve for the "< Return to Previous" and "Button 1" buttons.
	`s_right.gif`	Right curve for the "< Return to Previous" and "Button 1" buttons.
	`left_sm.gif`	Left curve for the "Search" and and smaller "Button 1" button.
	`right_sm.gif`	Right curve for the "Search" and smaller "Button 1" button.

The files in Table 1-3 show some of the `.gif` images in iPlanet Market Maker. The complete list of default `.gif` files is shown in Table 1-5, Table 1-6, and Table 1-7.

Table 1-5 Image Files in the `@IMM_DOCROOT_FULL@/imm40/iMM/images` Directory

Image	File name	Used for
	`curve_left.gif`	Top border.
	`curve_right.gif`	Top border.
	`curve_bottom_left.gif`	Bottom border
	`curve_bottom_right.gif`	Bottom border.
	`tab_left.gif`	Tabs at the top of a page.
	`tab_right.gif`	Tabs at the top of a page.

Table 1-5 Image Files in the `@IMM_DOCROOT_FULL@/imm40/iMM/images` Directory *(Continued)*

Image	File name	Used for
◤	`m_arc.gif`	Upper-left corner curve in a module page after you login as a marketmaker.
◤	`b_arc.gif`	Upper-left corner curve in a module page after you login as a buyer.
◤	`s_arc.gif`	Upper-left corner curve in a module page after you login as a seller.
▫	`pixel.gif`	Adding spacing throughout (set width/height).
▲	`ascending.gif`	Indicating table column sort order.
▼	`descending.gif`	Indicating table column sort order.
▲	`arrow-up-blue.gif`	Blue arrows.

Table 1-5 Image Files in the `@IMM_DOCROOT_FULL@/imm40/iMM/images` Directory *(Continued)*

Image	File name	Used for
▼	`arrow-down-blue.gif`	Blue arrows.
▶	`arrow-right-blue.gif`	Blue arrows.
▼	`sortarrowdown.gif`	Descending sorting of search results.
▲	`sortarrowup.gif`	Ascending sorting of search results.
✓	`check.gif`	RFx award symbol.
	`aim.gif`	Accessing AOL Instant Message.
	`people.gif`	Order management system (OMS) Edit Approver Screen.
	`blank_connector.gif`	OMS Edit Approver Screen.

Table 1-5 Image Files in the `@IMM_DOCROOT_FULL@/imm40/iMM/images` Directory *(Continued)*

Image	File name	Used for
└	`corner_connector.gif`	OMS Edit Approver Screen.
├	`tee_connector.gif`	OMS Edit Approver Screen.
│	`vertical_connector.gif`	OMS Edit Approver Screen.
⊟	`collapsebutton.gif`	Collapsing a channel.
☒	`removebutton.gif`	Removing a channel.
☑	`checkBox.gif`	Check boxed displayed when you select a catalog item.
iPlanet Market Maker	`IMM_smBanner.gif`	Upper-right banner text inside a module.

Table 1-5 Image Files in the `@IMM_DOCROOT_FULL@/imm40/iMM/images` Directory *(Continued)*

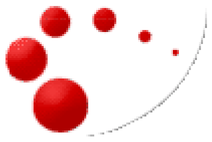
Image	File name	Used for
	`MM_iPlanet_logo.gif`	Large iPlanet logo for login page.
	`MM_smiPlanet_logo.gif`	Small iPlanet logo for portal pages.
	`MM_3rdP_logo.gif`	Large template for creating third party logos for login page.
	`MM_sm3rdP_logo.gif`	Small template for creating third party logos for login page.
iPlanet™ Market Maker	`IMM_Banner.gif`	Large Market Maker banner for login page.

Table 1-5 Image Files in the `@IMM_DOCROOT_FULL@/imm40/iMM/images` Directory *(Continued)*

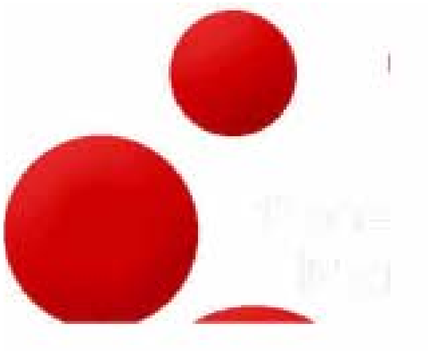
Image	File name	Used for
iPlanet™ Market Maker	`MM_Banner.gif`	Small Market Maker banner for portal pages.
	`Error.gif`	Indicating an error page.
	`login_BL.gif`	Bottom-left portion of graphic in the login page.
	`login_TL.gif`	Top-left portion of graphic in the login page.

Table 1-5 **Image Files in the `@IMM_DOCROOT_FULL@/imm40/iMM/images` Directory *(Continued)***

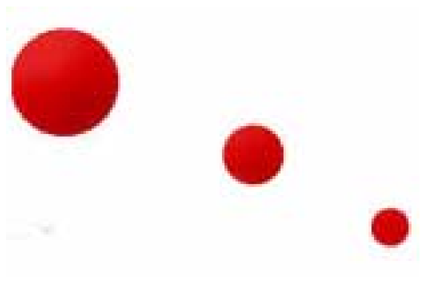
Image	File name	Used for
	`login_TR.gif`	Top-right portion of graphic in the login page.

Table 1-6 **Image Files in the `@IMM_DOCROOT_FULL@/imm40/iMM/images/buttons` Directory**

Image	File name	Used for
	`s_left.gif`	Dark buttons in the seller top border.
	`s_right.gif`	Dark buttons in the seller top border.
	`b_left.gif`	Dark buttons in the buyer top border.
	`b_right.gif`	Dark buttons in the buyer top border.

Table 1-6 **Image Files in the `@IMM_DOCROOT_FULL@/imm40/iMM/images/buttons` Directory *(Continued)***

Image	File name	Used for
	`m_left.gif`	Dark buttons in the market maker top border.
	`m_right.gif`	Dark buttons in the market maker top border.
	`x_left.gif`	Dark buttons before role is known (registration).
	`x_right.gif`	Dark buttons before role is known (registration).
	`left_sm.gif`	All buttons in the content area.
	`right_sm.gif`	All buttons in the content area.
	`page_button_left.gif`	Left curve for buttons inside the module content area.
	`page_button_right.gif`	Right curve for buttons inside the module content area.

Table 1-6 **Image Files in the `@IMM_DOCROOT_FULL@/imm40/iMM/images/buttons` Directory *(Continued)***

Image	File name	Used for
	`button_left.gif`	Left curve for buttons outside the module content area.
	`button_righ.gif`	Right curve for buttons outside the module content area.
	`search.gif`	Search in the content area.
	`go.gif`	Filtering the search in the content area.
	`s_help.gif`	Seller title bar Help button.
	`s_home.gif`	Seller title bar Home button.
	`s_logout.gif`	Seller title bar Logout button.

Table 1-6 **Image Files in the `@IMM_DOCROOT_FULL@/imm40/iMM/images/buttons` Directory *(Continued)***

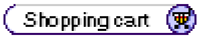
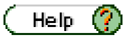
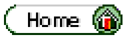
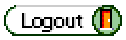
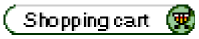
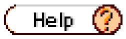
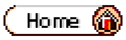
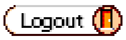
Image	File name	Used for
	`s_shoppingcart.gif`	Seller title bar shopping cart button.
	`b_help.gif`	Buyer title bar Help button.
	`b_home.gif`	Buyer title bar Home button
	`b_logout.gif`	Buyer title bar Logout button.
	`b_shoppingcart.gif`	Buyer title bar shopping cart button.
	`m_help.gif`	Market maker title bar Help button.
	`m_home.gif`	Market maker title bar Home button
	`m_logout.gif`	Market maker title bar Logout button.

Table 1-6 Image Files in the `@IMM_DOCROOT_FULL@/imm40/iMM/images/buttons` Directory *(Continued)*

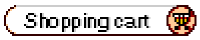
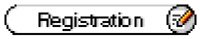
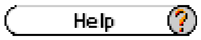
Image	File name	Used for
	`m_shoppingcart.gif`	Market maker title bar shopping cart button.
	`registration.gif`	Marketplace title bar Registration button.
	`help.gif`	Marketplace title bar Help button.
	`customize_up.gif`	Customizing portal page.
	`customize_down.gif`	Customizing portal page.
	`customize_left.gif`	Customizing portal page.
	`customize_right.gif`	Customizing portal page.

Table 1-6 **Image Files in the `@IMM_DOCROOT_FULL@/imm40/iMM/images/buttons` Directory *(Continued)***

Image	File name	Used for
	`pagination-go.gif`	Pagination.

Table 1-7 **Image Files in the `@IMM_DOCROOT_FULL@/imm40/iMM/images/nav` Directory**

Image	File name	Used for
	`b_left.gif`	Indicating the selected module.
	`b_right.gif`	Indicating the selected module.
	`b_spacer.gif`	Separating the modules.
	`m_left.gif`	Indicating the selected module.
	`m_right.gif`	Indicating the selected module.
	`m_spacer.gif`	Indicating the selected module.

Table 1-7 Image Files in the `@IMM_DOCROOT_FULL@/imm40/iMM/images/nav` Directory

Image	File name	Used for
	`m_spacer.gif`	Separating the modules.
	`s_left.gif`	Indicating the selected module.
	`s_right.gif`	Indicating the selected module.
	`s_spacer.gif`	Separating the module names.

Chapter 2

Customizing Display Profiles

This chapter contains the following sections:

- Accessing Display Profiles
- Using the Display Profile Editor
- XML Tags

Accessing Display Profiles

When you start iPlanet Market Maker, the main page displayed is called the portal page. This page is dynamically generated based on a description of the available content. This description of this content is called a display profile.

The display profile is defined in an XML document. iPlanet Market Maker features a display profile editor that lets you edit and customize the XML tags that define the display profile.

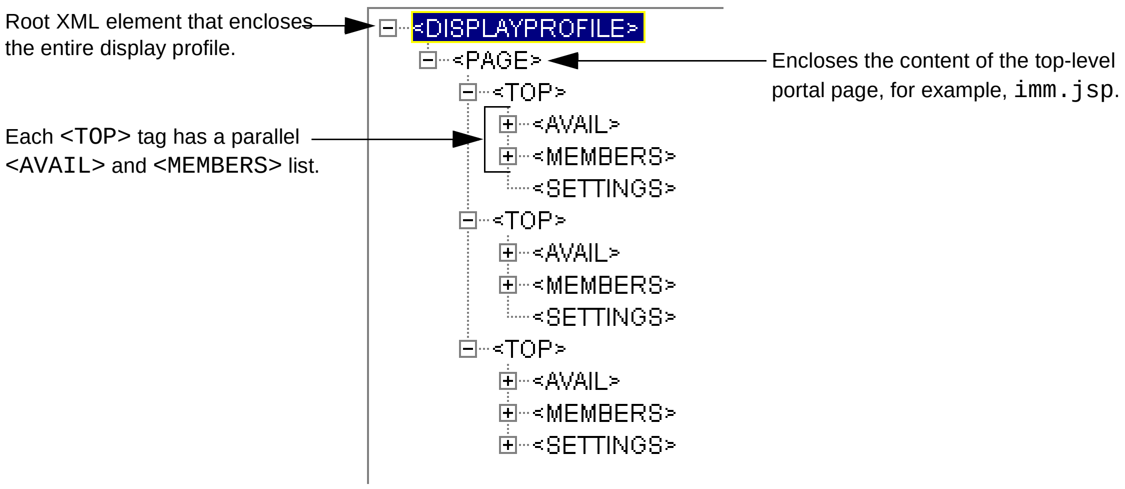
This chapter provides an example for interpreting and customizing a display profile. The type of display profile editing described is typically done by the market maker or company administrator.

Using the Display Profile Editor

The display profile editor is the tool you use to make changes in the display profile associated with a particular LDAP node. To start the display profile editor, type `DPEdit` on the command line. After you complete the login procedure, you can access and edit the XML tags that define the display profile.

Figure 2-1 shows the basic structure of the XML tags that define the display profile.

Figure 2-1 Display Profile Structure


Root XML element that encloses the entire display profile.
Each <TOP> tag has a parallel <AVAIL> and <MEMBERS> list.
Encloses the content of the top-level portal page, for example, imm.jsp.
<DISPLAYPROFILE>
<PAGE>
<TOP>
<AVAIL>
<MEMBERS>
<SETTINGS>
<TOP>
<AVAIL>
<MEMBERS>
<SETTINGS>
<TOP>
<AVAIL>
<MEMBERS>
<SETTINGS>

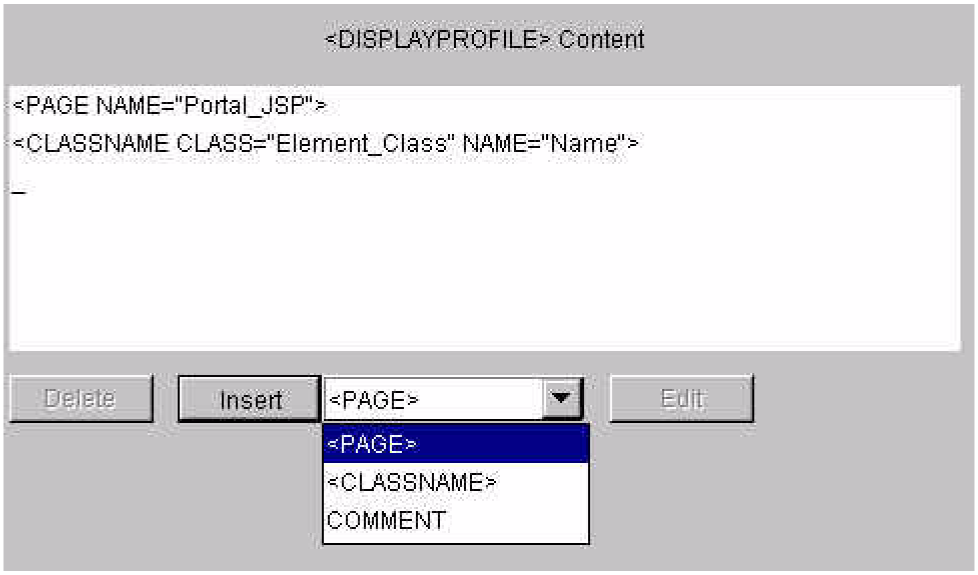
XML Tags

This section contains a list of the XML tags you can access and edit to customize the display of the iPlanet Market Maker portal.

<DISPLAYPROFILE>

The DISAPLAYPROFILE tag is the root element that encloses the entire document. It contains any number of the following two tags in any order.

- <PAGE>
- <CLASSNAME>

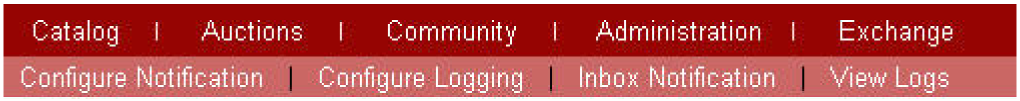


```
<PAGE NAME="Portal_JSP">
<CLASSNAME CLASS="Element_Class" NAME="Name">
```

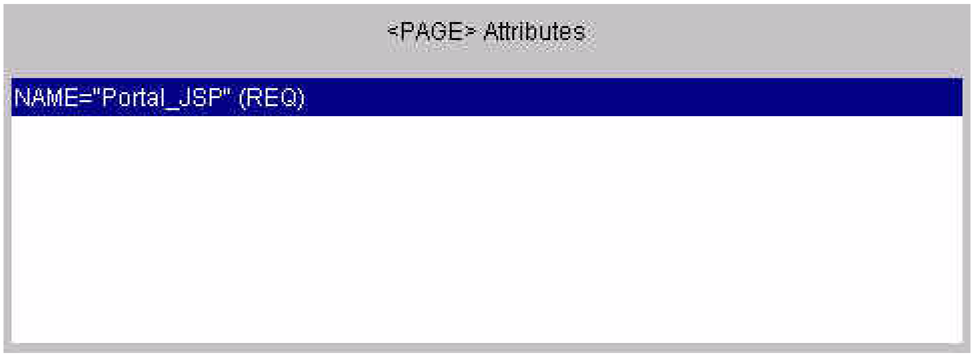
This tag has no attributes.

<PAGE>

The PAGE tag encloses the information relevant to a top-level or portal page. It can contain any number of <TOP> tags.


Catalog | Auctions | Community | Administration | Exchange
Configure Notification | Configure Logging | Inbox Notification | View Logs

This tag has a required NAME attribute.



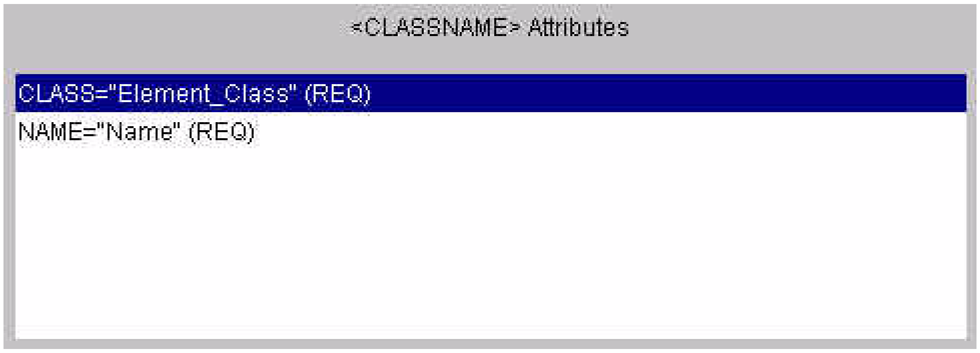
NAME="Portal_JSP" (REQ)

The NAME attribute specifies the name of page that applies to the <PAGE> tag, such as the file name of a portal JSP.

<CLASSNAME>

The CLASSNAME tag maps a symbolic name to a Java class so that you can change class names specified elsewhere in a document in one place. This tag is optional. You can also specify classes directly. If you use this tag, it must appear before, or at a higher inheritance level than, other tags that use it. This tag has no content.

This tag has the following two attributes.


<CLASSNAME> Attributes

CLASS="Element_Class" (REQ)
NAME="Name" (REQ)

ATTRIBUTE	DEFINITION
CLASS	Name of the Java class (sub-class of DisplayElement).
NAME	Name to refer to the class in symbolic references.

<TOP>

The TOP tag describes a top-level element, usually a container, that may be instanciated on the <PAGE> in which it is contained. The <TOP> element has subelements that fully describe the container: its settings, available list, and member list. With this information, the portal module can call the relevant classes and display the container and its content.

In the display profile you describe what can be displayed. In the JSP you use this description to tell the engine how to render it. The name provided in this tag is the name to use in the JSP when you create the container as a Java object. This name is the link between the portal JSP and the DisplayProfile.

For example:

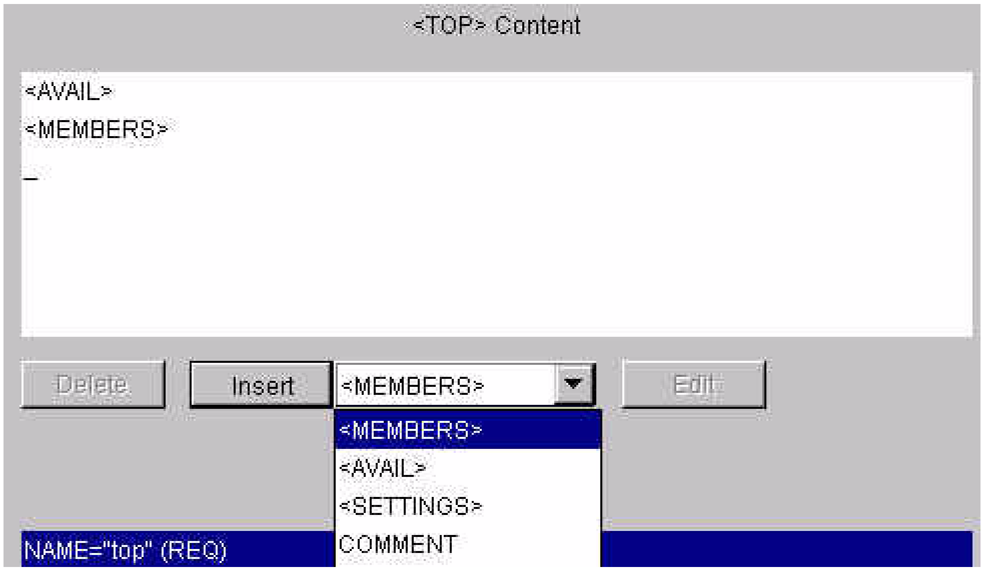
In the display profile there is a <PAGE> element called mypage.jsp that has a <TOP> element named myTabularLayoutContainer.

If you want to use this container, use a portal page named mypage.jsp that contains the following code:

```
<%
elt = new TabularDisplay("myTabularLayoutContainer");
elt.doDisplay();
%>
```

to specify that a TabularDisplay is to be shown in a particular part of the JSP, and that it corresponds to the <TOP> tag named `myTabularLayoutContainer` in the display profile.

This tag may optionally start with an <AVAIL> tag followed by a <MEMBERS> tag. Following these two tags can be a <SETTINGS> tag.

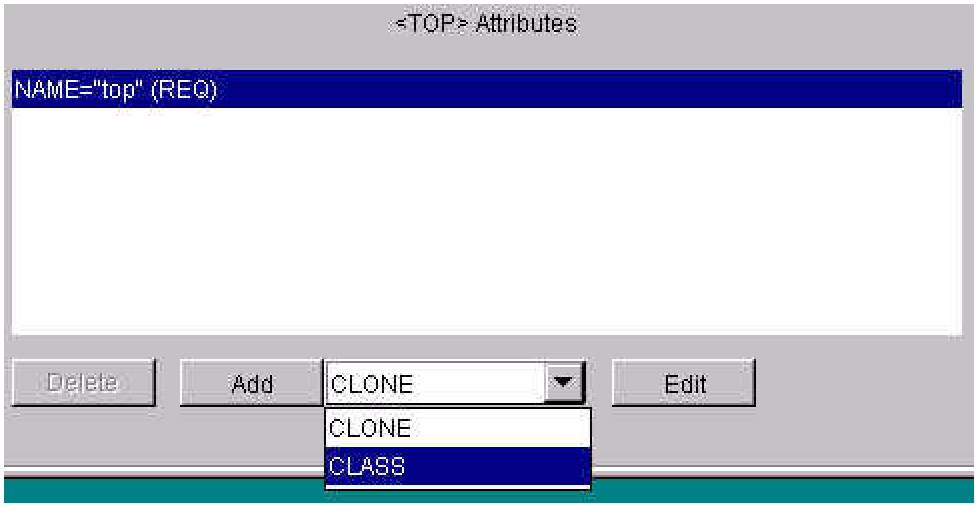


<AVAIL>
<MEMBERS>

NAME="top" (REQ)

The <AVAIL> and <MEMBERS> tags represent the content of a container, and are usually present. These tags should be missing only if this <TOP> tag is a single content item, or if it is cloned. If the CLONE attribute was specified, the <AVAIL> and <MEMBERS> tags can still be present, and are aggregated onto the cloned settings.

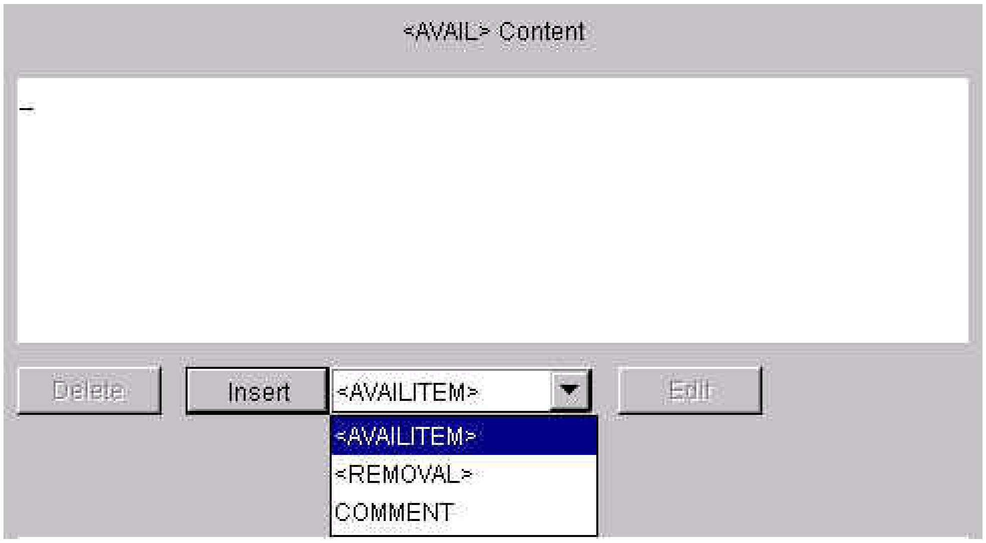
The <TOP> tag has the following attributes.



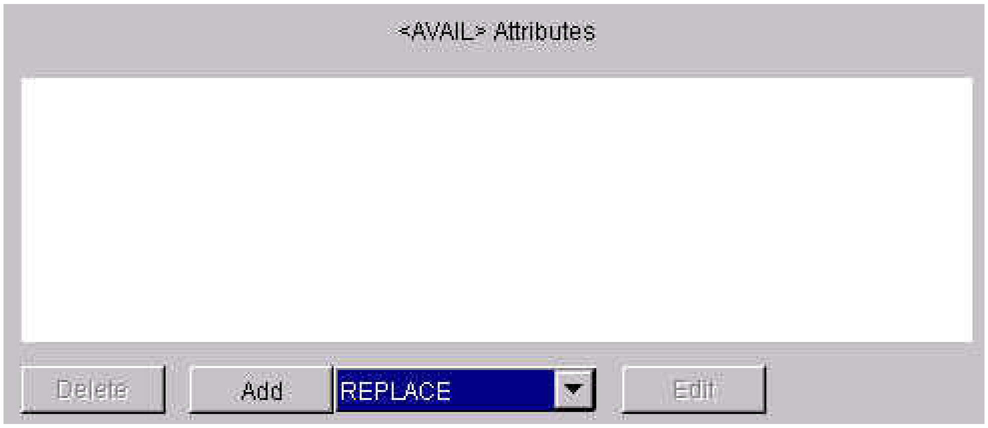
ATTRIBUTE	DEFINITION
NAME (required)	Links the item to a particular page element.
CLONE (optional)	Indicates that the content for this tag is to be copied from a previous place in the document. This attribute allows content to be moved to different places within the containment hierarchy without being duplicated. Currently, it is specified by surfacing an internal key mechanism. The syntax is [<page>]<elt>, where <page> is the JSP name, and <elt> is component names starting with the "top" name, connected by "_c_".
CLASS (optional)	Documents the class instanciated in the JSP. For informational purposes only.

<Avail>

The Avail tag defines the list of available content for a container, which can be content items or further containers. This tag can contain any number of <AVAILITEM> or <REMOVAL> tags.



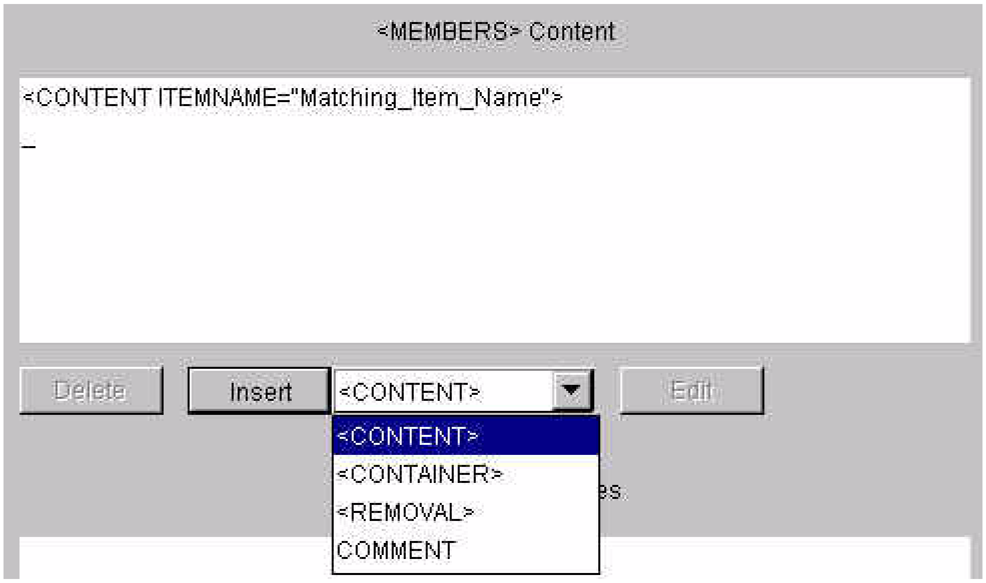
The <AVAIL> tag has an optional "REPLACE" attribute you can set to 1 or 0.



If 1, the list replaces any inherited list form an earlier document, rather than augmenting it. This attribute is not usually used. If it appears, it effectively defeats any "CLONE" mechanism specified on the tag containing it, because the list is cloned, then thrown away. The default is 0.

<MEMBERS>

Describes the current content for a container and always appears in with an <AVAIL> tag, whose list members it references by name. This tag can contain any number of <CONTENT>, <CONTAINER>, or <REMOVAL> tags. These tags reference items by name in the parallel <AVAIL> list.

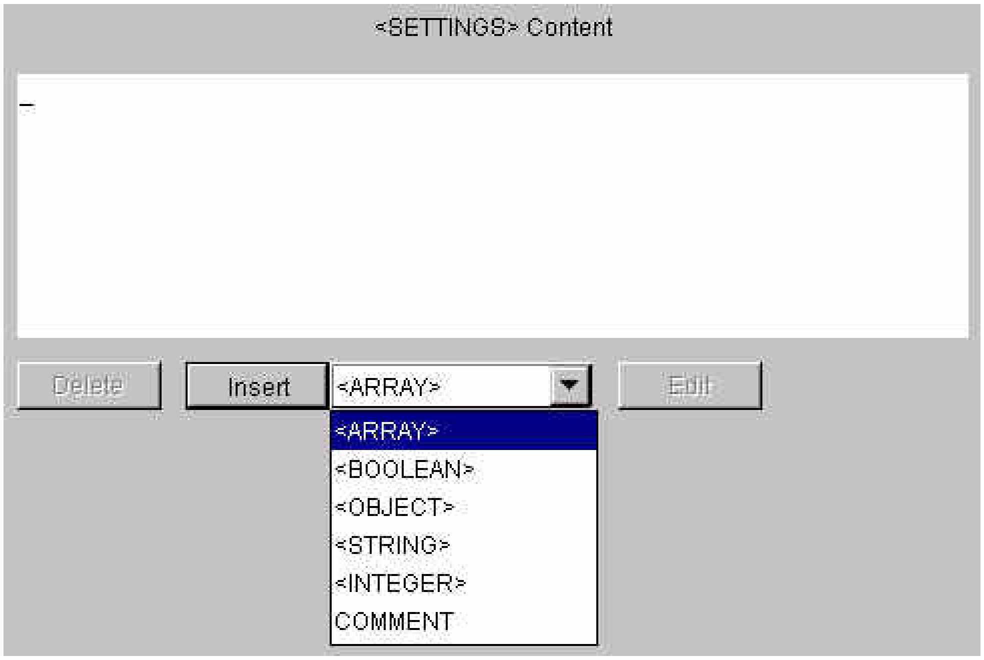


```
<CONTENT ITEMNAME="Matching_Item_Name">
_
```

The <MEMBERS> tag has an optional "REPLACE" attribute you can set to 1 or 0. If 1, the list replaces any inherited list form an earlier document, rather than augmenting it. This attribute is not usually used. If it appears, it effectively defeats any "CLONE" mechanism specified on the tag containing it, because the list is cloned, then thrown away.

<SETTINGS>

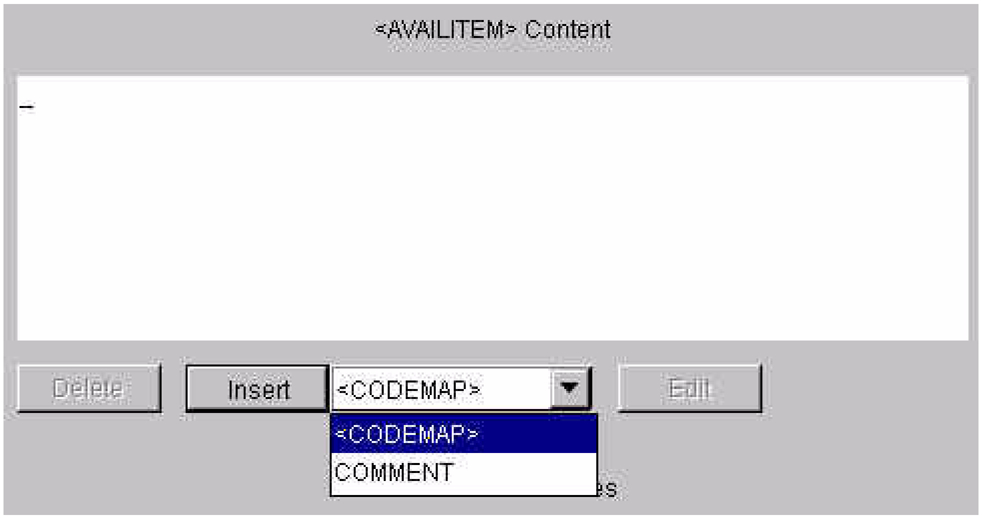
Contains arbitrary name-value pairs to be used by an enclosing <TOP>, <CONTENT> or <CONTAINER> tag. Represents item-specific attributes with particular types, either single elements or arrays. This tag can contain any number of <STRING>, <BOOLEAN>, <INTEGER>, <OBJECT> or <ARRAY> tags.



This tag has no attributes.

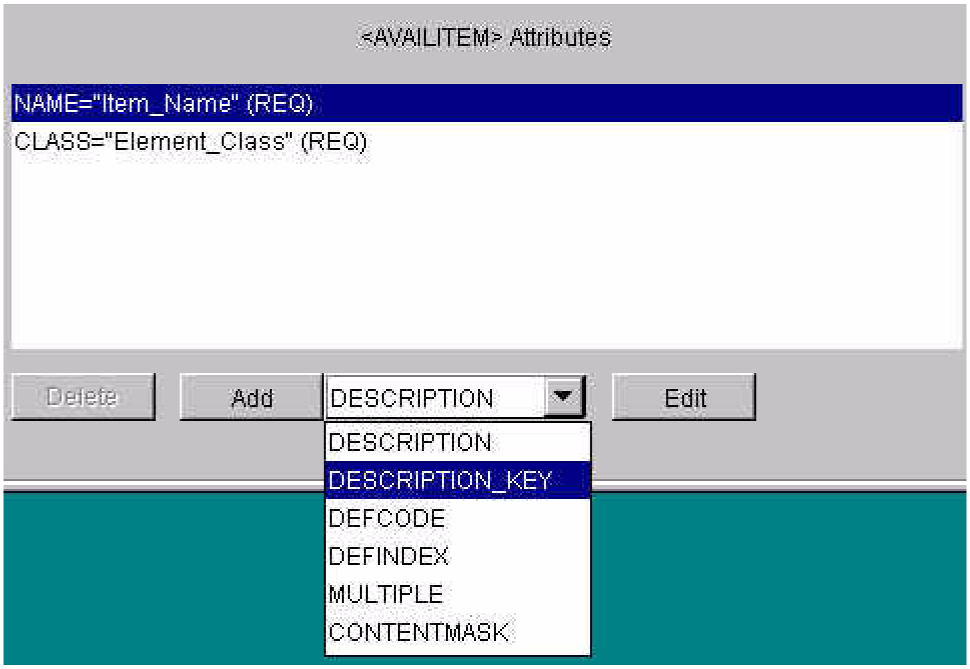
<AVAILITEM>

This tag defines an available content item within a container.



This tag appears inside <AVAIL> and represents a member of a list present for <TOP> and <CONTAINER>. It links a particular name to a class. <CODEMAP> is an alternate class to map an <AVAILITEM> based on an action tag. The <AVAILITEM> tag can also contain a <SETTINGS> tag that specifies the initial settings to be propagated to the parallel item in the <MEMBERS> list when the user adds a channel.

This tag has the following attributes.

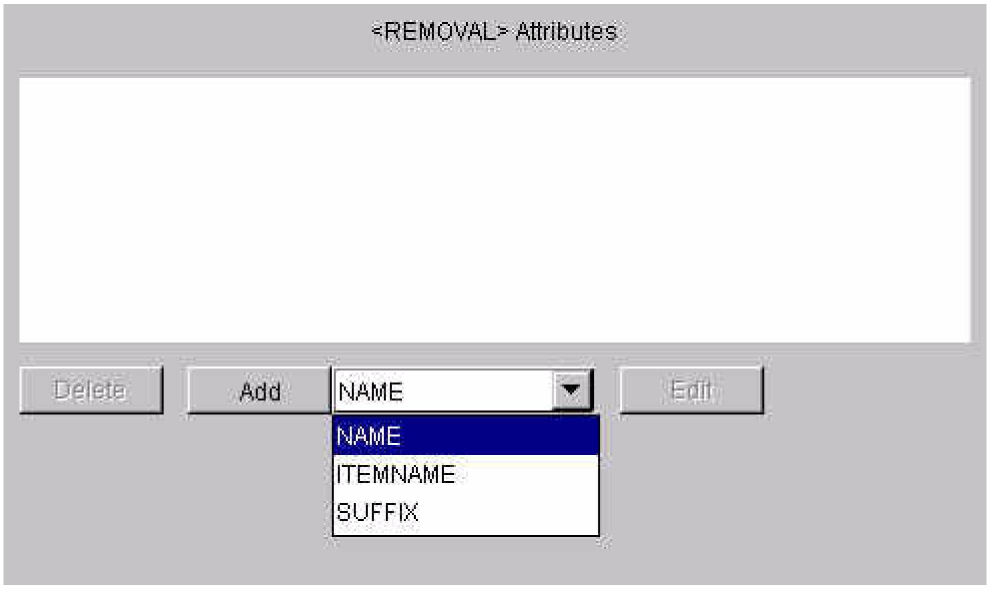


ATTRIBUTE	DEFINITION
NAME	Required attribute that specifies the name of the item as referred to in the <MEMBERS> lists parallel to the <AVAIL> list that contains this item.
CLASS	Required attribute that specifies the actual DisplayElement sub-class corresponding to this item or the symbolic name specified by an earlier <CLASSNAME> tag.
DESCRIPTION	A string that defines the user variable description for this item, such as dialogs.
DESCRIPTION_KEY	An integer used to look it up in the "DisplayProfile" properties file so the string may be internationalized (translated to a different language).
DEFCODE	Integer "action code" (operation) initially applied to the element.
DEFINDEX	Integer index associated with the operation.

ATTRIBUTE	DEFINITION
MULTIPLE	Can be 0 or 1. If 1, indicates that multiple instances of this item are allowed. The default is 0.
CONTENTMASK	A string specifying permissions required to see this content take the form of a comma separated list of "|" separated names. A user gets a list of permissions at login time, which is compared against this list. One of the "|" separated names from each comma separated component must be possessed to see the item. Could be generalized to an arbitrary boolean expression.

<REMOVAL>

Specifies the removal of an inherited named item from an <AVAIL> or <MEMBERS> list. This tag has the following attributes.

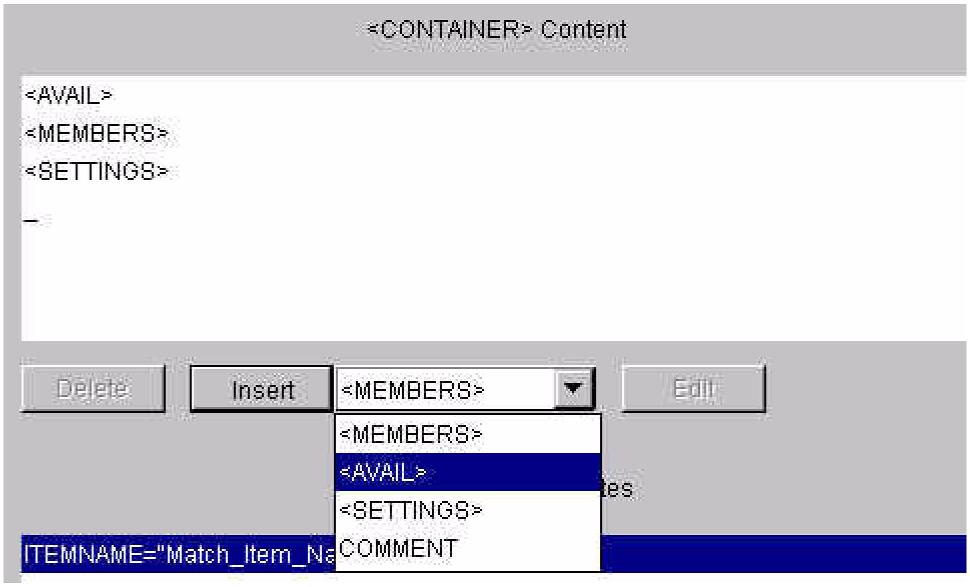


NAME	Specifies the item to remove in an `<AVAIL>` list.
ITEMNAME	Specifies the item to remove in a `<MEMBERS>` list.

SUFFIX (optional)	Distinguishes multiple members constructed from the same <AVAILITEM>.

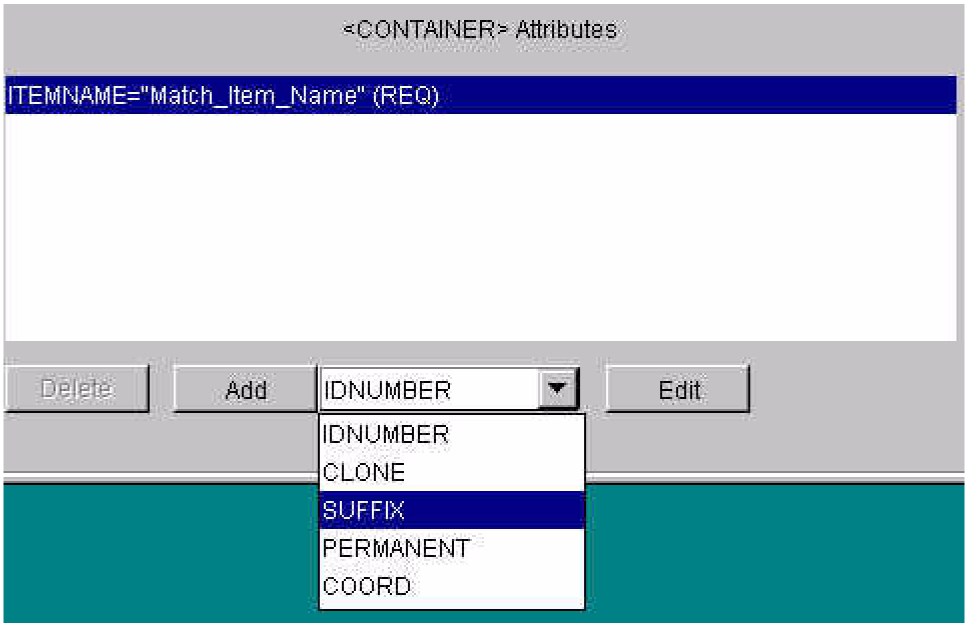
<CONTAINER>

A member of a <MEMBERS> tag that describes a content item that is a container aggregating more items. Its attributes link it to the appropriate <AVAILITEM> in the <AVAIL> list parallel to the <MEMBERS> list it resides in, as well as providing information about its manipulation within the container.



This tag must contain one <AVAIL> tag, followed by one <MEMBERS> tag. It can optionally contain a <SETTINGS> tag after the <MEMBERS> tag.

This tag has the following attributes.

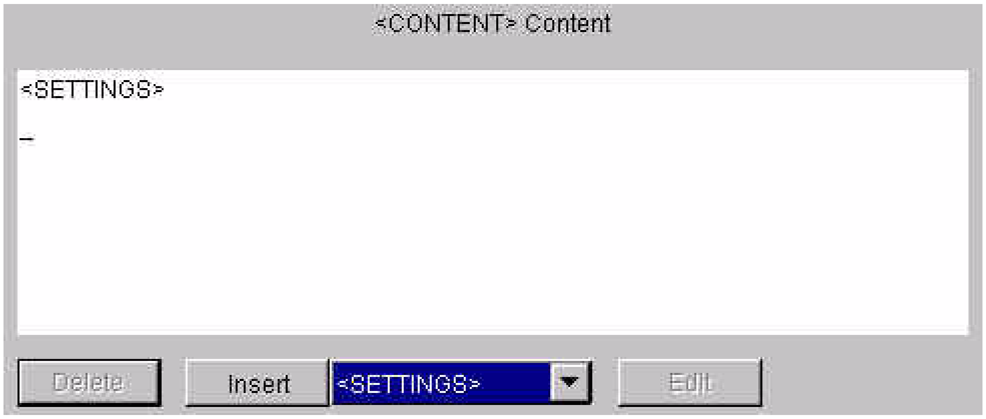


ATTRIBUTE	DEFINITION
ITEMNAME (required)	Links it by name to an `<AVAILITEM>`. If the `<AVAILITEM>` allows multiple copies, there should also be a "`SUFFIX`" attribute that distinguishes the multiple members constructed from the same `<AVAILITEM>`.
IDNUMBER (optional)	Integer attribute currently used only for tabbed displays. It allows JSP code constructing very dynamic tab bars to identify tabs programmatically. It is generally an application-defined integer identifier for the member.
CLONE	See the "CLONE" attribute description under `<TOP>`.
SUFFIX	Specifies the member to remove.
PERMANENT (optional)	Set to 0 or 1. If 1, indicates that the item is not intended to be removable by the user.

ATTRIBUTE	DEFINITION
COORD (optional, but usually	A comma-separated list of integers used by the container to arrange its members. The number of required coordinates and their interpretation depends on the container. For instance, in a tabular arrangement, this specifies the column / row.

<CONTENT>

This tag defines a member that is content, not a further container.

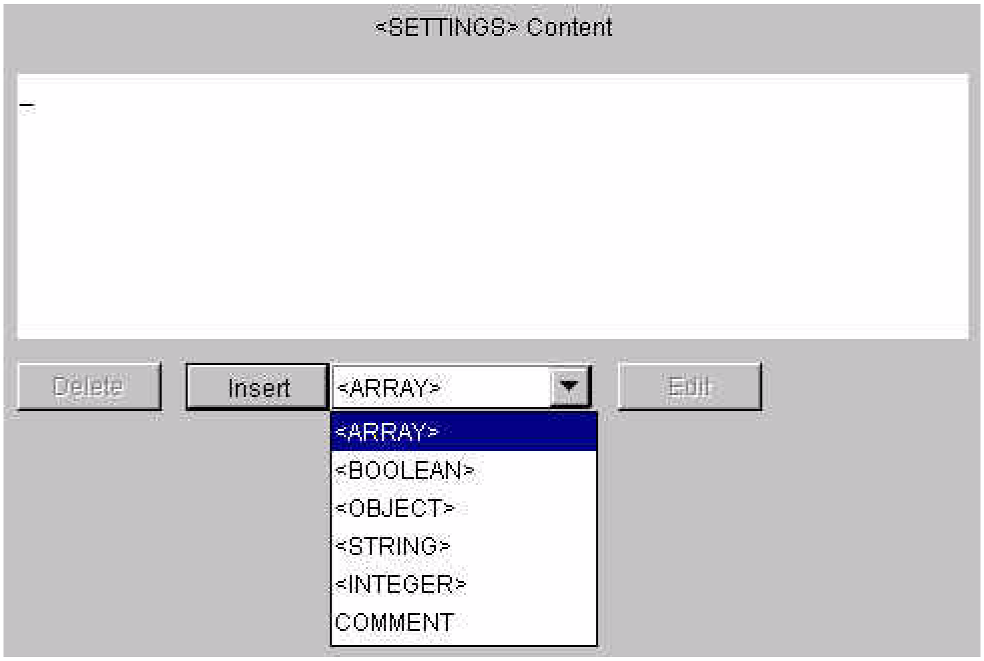


<SETTINGS>

This tag can contain a single <`SETTINGS`> tag. Its attributes are identical to the attributes for the <`CONTAINER`> tag.

<STRING>, <BOOLEAN>, <INTEGER>, <OBJECT>

Name data items contained in a <`SETTING`> tag.



The "NAME" attribute must be present, unless the tag is the member of an <ARRAY> tag. The "VALUE" attribute containing a string appropriate for the type must also be present, except that for <STRING>, it can be replaced with "VALUE_KEY" that provides an integer key allowing the value to be read from the DisplayProfile properties file, so that it can be internationalized (translated to a different language). Note that the appropriate type of <BOOLEAN> is "true" of "false".

For <OBJECT>, the value is understood to be a serialized object. You usually do not need to edit or enter <OBJECT> tags.

These <OBJECT> tags are usually generated at run-time by your program if it makes use of the display profile mechanism to store some settings for an individual user.

<ARRAY>

Allows for array-valued members of the <SETTINGS> tag. This tag can contains further <STRING>, <BOOLEAN>, <INTEGER>, <OBJECT> and <ARRAY> tags. The "NAME" attributes on these tags are ignored because they are accessed as members of the whole named array.

This tag has a "NAME" attribute that must be present unless this array is, in turn, a member of another <ARRAY>.

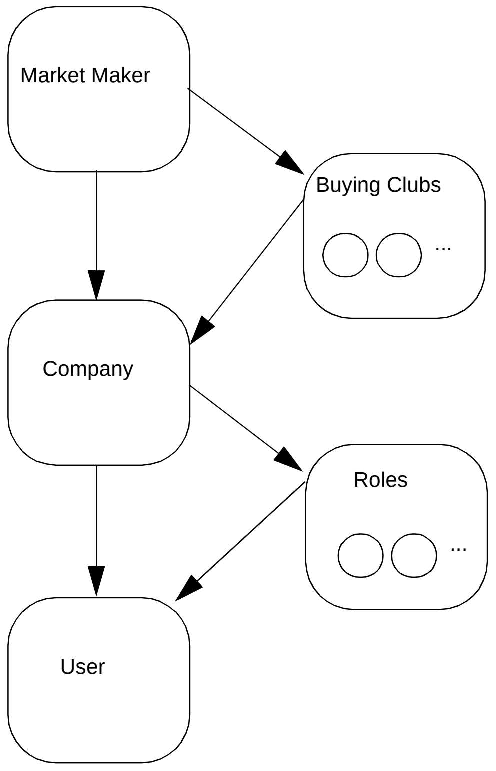
Inheriting and Merging of Display Profiles

The display profile that applies to a user is inherited depending on the LDAP node for that user. iPlanet Market Maker has the following types of nodes.

- Market Maker
- Buying club (also called Group)
- Company
- Role
- User

The market maker node is the root display profile. The buying club nodes inherit the display profile defined by the market maker node. The company node inherits the display profiles from the buying club node and the market maker node, and so on down the node hierarchy. See Figure 2-2.

Figure 2-2 LDAP Node Hierarchy



At run-time, the display profiles are merged depending on the user node. All the nodes show in Figure 2-2 are merged, in the specified order, to determine the user's run-time display profile.

Creating a New Display Profile

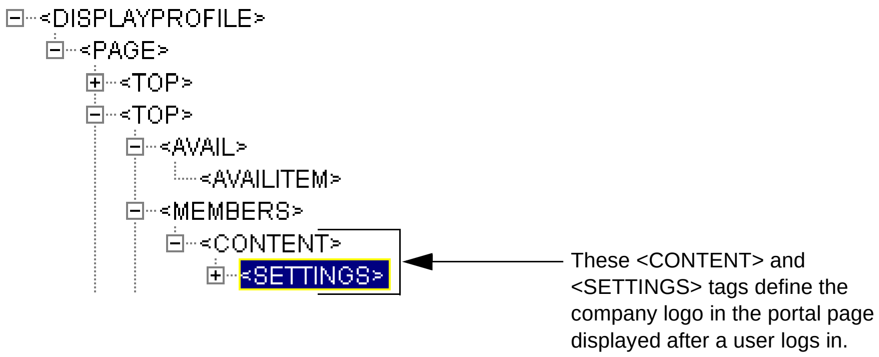
The `DPEdit` editor lets you access a particular LDAP node in isolation from the other nodes. Suppose, for example, that as a company administrator you wanted to change the company logo that appears in the portal page after a user logs in to iPlanet Market Maker. Do the following steps.

1. Log in to the display profile editor and open a new file.

2. Specify the Company node.
3. Create a new display profile that replaces the existing iPlanet Market Maker logo.

Because the Company display profile is inherited from the Market Maker display profile, the first step to change company logo is to understand how that logo is specified in the Market Maker display profile. The Market Maker must provide access to this display profile information. For example, Figure 2-3 shows where the company logo is defined in the default iPlanet Market Maker portal page displayed after a user logs in.

Figure 2-3 Location of Company Logo Definition



These <CONTENT> and <SETTINGS> tags define the company logo in the portal page displayed after a user logs in.

To specify a different company logo for your Company node, add the following code in the display profile editor to replace the market-maker defined tags shown in Figure 2-3.

```
<DISPLAYPROFILE>
    <PAGE NAME="imm.jsp">
    <TOP NAME="comlogosmall">
        <AVAIL />
        <MEMBERS REPLACE="true">
            <CONTENT ITEMNAME="ComLogoSmall">
            <SETTINGS>
                <STRING NAME="FileRoot" VALUE="imm" />
                <STRING NAME="ContentFile" VALUE="logo.html" />
                <BOOLEAN NAME="NoMenuBar" VALUE="true" />
            </SETTINGS>
            </CONTENT>
        </MEMBERS>
    <SETTINGS />
    </TOP>
    </PAGE>
</DISPLAYPROFILE>
```

In this example, the contents defined in the `logo.html` file replace the contents defined in the `immlogosmall.html` file in the default market maker display profile.

Interpreting Tabbed Displays

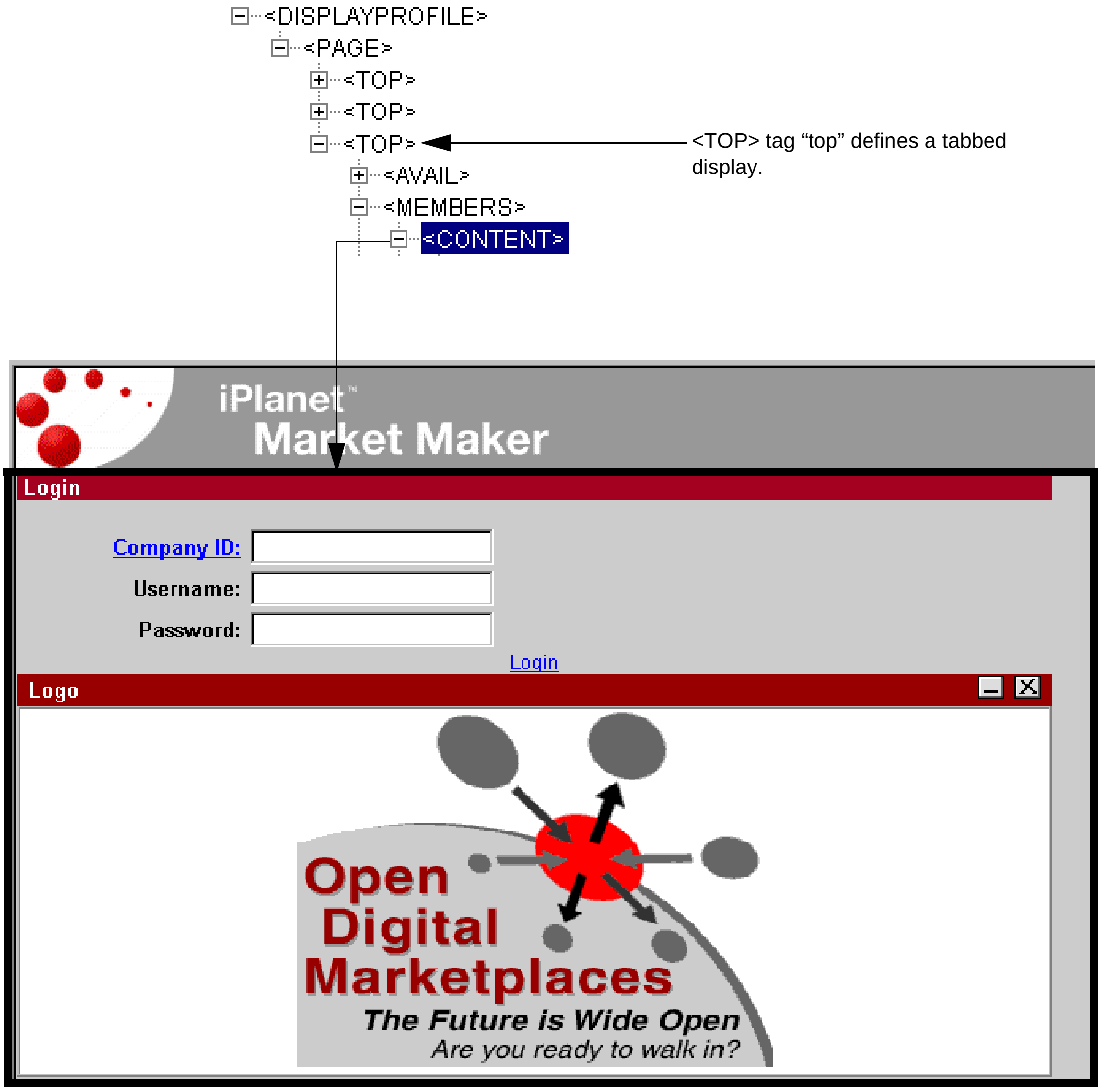
The `NAME` attribute of the third `<TOP>` tag (`"top"`) Figure 2-4 defines with a tabbed display (`TabbedDisplay`) in the `imm.jsp` file. In a tabbed display, the `COORD` attribute determines the order in which the tabs are displayed. The container for a tabbed display can provide a tab bar, or allow the JSP to construct a tab bar as is done in `imm.jsp`. If the JSP manages the display of the tab bar, the `COORD` attribute might be irrelevant depending on the logic in the JSP.

It is possible to define content filtered by user privileges, which is why the tabbed display shows different content for different users. This is also why the item corresponding to the login page is shown only when you are not yet logged in.

Under the third <TOP> tag shown in Figure 2-4, the first <CONTENT> tag defines a tabular layout container for the channels displayed in the “anonymous” portal page, which is the page displayed when you first start Market Maker and the role of the user is not known. This `<CONTENT>` tag is a typical container in which you add channels.

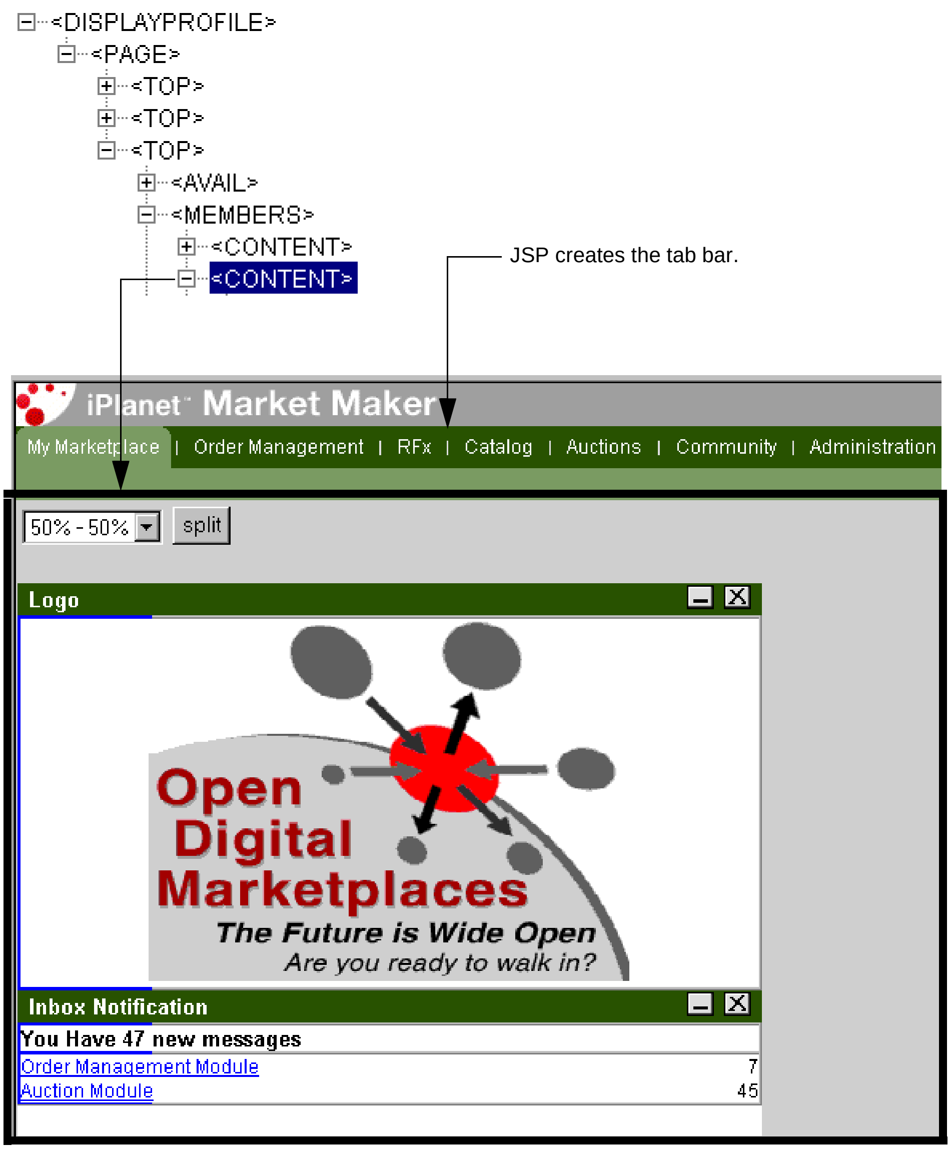
Because only one item is available to be displayed in this example, the JSP does not construct a tab bar.

Figure 2-4 **"Anonymous" Portal Page**



The second of these <CONTENT> tags in Figure 2-5 defines the channels displayed in the "MyMarketPlace" portal page, which is the page displayed when you log in to Market Maker. Regardless of your role, the "My Marketplace" tab is the default active tab. In this example, the JSP creates the tab bar.

Figure 2-5 "MyMarketplace" Portal Page



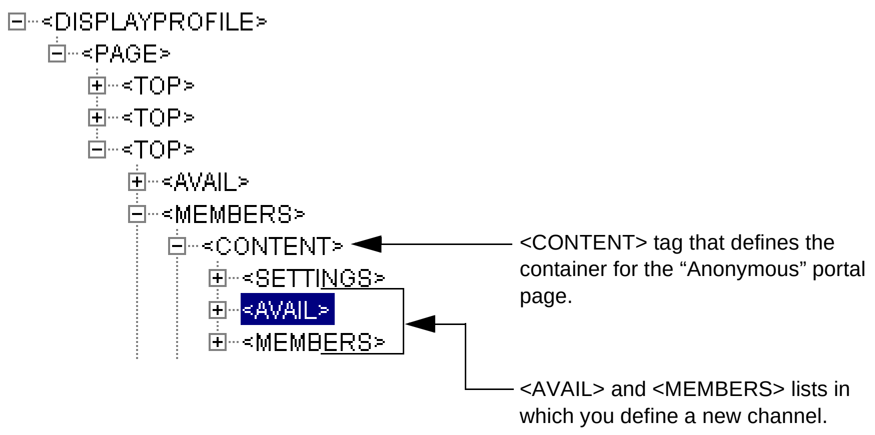
You Have 47 new messages

Order Management Module	7
Auction Module	45

Adding a Channel to a Portal Page

Suppose that as a market maker you wanted to add a channel to the "Anonymous" portal page. The default iPlanet Market Maker display profile for this page is shown in Figure 2-6.

Figure 2-6 Container for the "Anonymous" Portal Page


```
<DISPLAYPROFILE>
  <PAGE>
    <TOP>
    <TOP>
    <TOP>
      <AVAIL>
      <MEMBERS>
        <CONTENT>
          <SETTINGS>
          <AVAIL>
          <MEMBERS>
```
<CONTENT> tag that defines the container for the "Anonymous" portal page.
<AVAIL> and <MEMBERS> lists in which you define a new channel.

Under the `<CONTENT>` tag for the “Anonymous” container, you use the display profile editor to add a new channel as shown in the following example.

```
<CONTENT ITEMNAME="Anonymous">
       <AVAIL>
         <AVAILITEM NAME="Developer"
CLASS="com.iplanet.ecommerce.vortex.display.FileContent" DESCRIPTION="My
Documents" DEFCODE="-1" DEFINDEX="0" />
       </AVAIL>
       <MEMBERS>
           <CONTENT ITEMNAME="MyDocuments" COORD="0,1,0">
              <SETTINGS>
                  <STRING NAME="FileRoot" VALUE="MyDir" />
                  <STRING NAME="ContentFile" VALUE="doc/customization.html" />
              </SETTINGS>
       </CONTENT>
```

This code adds a channel called “My Documents” in the “Anonymous” page. The contents of this channel are defined by the `customization.html` file.

About the Display Profile Loader

Display profiles are XML documents that define the contents of the iPlanet Market Maker portal. Marketmaker administrators can use the display profile editor to access and edit display profiles. The display profile loader is a tool for marketmaker administrator to load the modified community display profile information in the LDAP server and Oracle database. You use the display profile loader at installation time and whenever you want to change the portal display.

The community module defines a hierarchy of entities for the marketmaker, companies, and users. You can modify display profiles at the level of each of these entities to customize the iPlanet Market Maker portal display. You use the display profile loader to load these XML documents.

Using the Display Profile Loader

To run the display profile loader, use the following command.

```
java com.iplanet.ecommerce.vortex.tools.dpimport.LoadDisplayProfile
-load | -unload config_file.xml
```

The inputs to this command line are described in the following table.

DISPLAY PROFILE LOADER INPUT	DESCRIPTION
-load	Uses the entries to load the display profiles.
-unload	Uses the entries GUID to delete any display profile associated with them.
-confiv_file.xml	Name of the XML configuration file that the display profile editor uses to control the load process. This file format is described below.

The load configuration file consists of a set of <`LOADITEM`> elements. A <`LOADITEM`> element specifies the community <`NODE`>, identified by its GUID, for which the `config_file.xml` file is loaded and a <`FILE`> to load.

The display profile loader does the following operations.

Goes to LDAP and sets the `hasdisplayprofile` attribute to true for this GUID.

Goes to Oracle and puts the file in a `CLOB` field in the `CMN_CONFIG_PROFILE` table, with the `GUID` as a key.

After the tool has run, the display profiles specified in the XML files are available for use in the iPlanet Market Maker software.

```
<?xml version="1.0"?>

<DPLOADCONFIG>

   <WORKDIR PATH="">

   </WORKDIR>

   <LOADITEM>

   <NODE NAME="vortex" GUID="5b6abf:de3f060a01:-7fff">

   </NODE>

   <FILE NAME="demodp.xml">

   </FILE>

   </LOADITEM>

</DPLOADCONFIG>
```

CONFIGURATION FILE XML TAG NAME	DESCRIPTION
WORKDIR	Working directory path.
LOADITEM	Has two mandatory elements: NODE and FILE. NODE describes a community node, with an optional NAME used in the output and a mandatory GUID.
FILE	Specifies a file to upload. The attribute NAME is the name of the file.

Chapter 3

Customizing Modules

This chapter contains the following sections:

- What You Can Customize
- Configuring Module Properties
- Extending Event Handling

What You Can Customize

This section describes the two general methods you can use to customize iPlanet Market Maker modules.

- "Configuring Module Properties"
- "Extending Event Handling"

Module properties let you customize the messages and configurations of modules. Each module has a set of associated property files that provide access to the properties.

Event handlers control the communication between any user activity in a Java server web page (JSP) and the resulting action taken by the module. You can extend the module event handlers to create new functions and capabilities.

Configuring Module Properties

To edit messages and configure the modules, you edit the proper files in the `classes` directory. You edit property files in a text editor. Each module has four property files, as shown in Table 3-1. The module name is the prefix (not shown in the table) for each of the property files. The property files are in the `<ias_server_root>/ias/APPS/imm40/imm40/WEB-INF/classes` directory.

The property files have the following characteristics.

- Changes to property files apply only after you restart iPlanet Market Maker.
- The "#" character specifies a comment. You use this character at the beginning of each line in the property file that you want to be ignored.
- Each property key and value must be on one line.

Table 3-1 Property Files

Property File Name	Description
`<Module Name>Messages.properties`	Contains parameterized messages, usually error and notification messages. This file must be localized.
`<Module Name>Strings.properties`	Contains non-parameterized messages. This file must be localized.
`<Module Name>Display.properties`	Contains parameterized messages that are only displayed in JSPs. Not every module has this type of property file. This file must be localized.
`<Module Name>Configuration.properties`	Contains configuration parameters.

Property Keys and Values

In property files, each property has a property key and property value specified as a name-value pair, separated by an equal sign ("="). Property keys are usually integer values. Each module is assigned a block of 10,000 property key numbers. Property keys cannot contain the "+", "=", or "*" characters. Key numbers are unique for each module. Do not change existing property keys. To do so might disable some functionality.

There are three types of name-value pairs:

- Simple name-value pairs.

  This most common type of name-value pair has a property key and a property value, such as `7=false`.

- Arrays

  An array name-value pair can contain multiple values. For example:

  ```
  status[0]=closed
  status[1]=open
  ```

  Defines a property named `status` with values of `closed` and `open`.

- Hash table structures

  A hash table can contain multiple values with unique names. For example:

  ```
  field[size]=10
  field[description]=Enter the address of the sender
  ```

  Defines a property named `field` with sub-values of `size`, with a value of `10`, and `description`, with a value of `Enter the address of the sender`.

Parameters In Messages

Some properties in the property files have parameterized messages. For example,

```
  123=Exception {0} on database {1}
```

defines a parameterized message with 2 parameters. Usually the parameters passed to the messages are localized strings. Do not change or remove these values.

See `java.text.MessageFormat` documentation in JDK for details about parameterized messages.

Localizing Property Files

As mentioned in Table 3-1, the Messages, Strings, and Display property files must be localized. For each language supported by iPlanet Market Maker, there is a property file. Property files are Java Resource Bundles, which are discussed in detail in `java.util.PropertyResourceBundle` and `java.util.ResourceBundle`. Consult the JDK documentation for more details.

Each iPlanet Market Maker locale has a language code and a country code. Property files are named based on the supported locale. For example, `AuctionMessages_en.properties` is the name of the English property files (`en` is the language code for English). `AuctionMessages_fr.properties` is the name of the French translation of the property file.

It is important to note that at this time, all property files (for all modules and their localizations) are in a single shared directory that must be part of the `CLASSPATH`.

Encrypting Property Files

Because many properties contain sensitive information, an encryption mechanism is necessary to protect their values. Currently only `VortexConfiguration.properties` contains encrypted values. But any of the module configuration properties can be encrypted by the technique described in this section. Keep in mind that only the Configuration property file (and not the Messages, Strings, or Display files) can be encrypted.

There are two special purpose markers used within property keys to denote encrypted values:

- The "+" character denotes an obfuscated property. Obfuscation is a process of byte swapping an XORing, which makes a property value unreadable.

  Obfuscation is not encryption. It is used to make properties that cannot be encrypted (because they drive the encryption process itself) hard to read.

- The "*" character denotes an encrypted property. All encryption in iPlanet Market Maker is done with JCE 1.2.1, using DES encryption with a key chosen at install time.

It is important to remember that encrypted and obfuscated properties can only be edited after being decrypted. The process of editing an encrypted property file (for example,`VortexConfiguration.properties`) is as follows:

1. Decrypt the property file using the `encrypt.sh` shell script. The syntax for this command is

   `encrypt.sh decrypt` <*filename*>

   You are prompted to enter the password protecting the encryption key. This password was chosen at install time.

2. Edit the property file using a text editor.

   Remember that if you remove the "*" character from the property key, the property is not encrypted.

3. Re-encrypt the property using the `encrypt.sh` shell script. The syntax for this command is

   `encrypt.sh encrypt` <*filename*>

4. Again, you are prompted for the encryption password. It is very important that you enter the same password chosen at install time.

Extending Event Handling

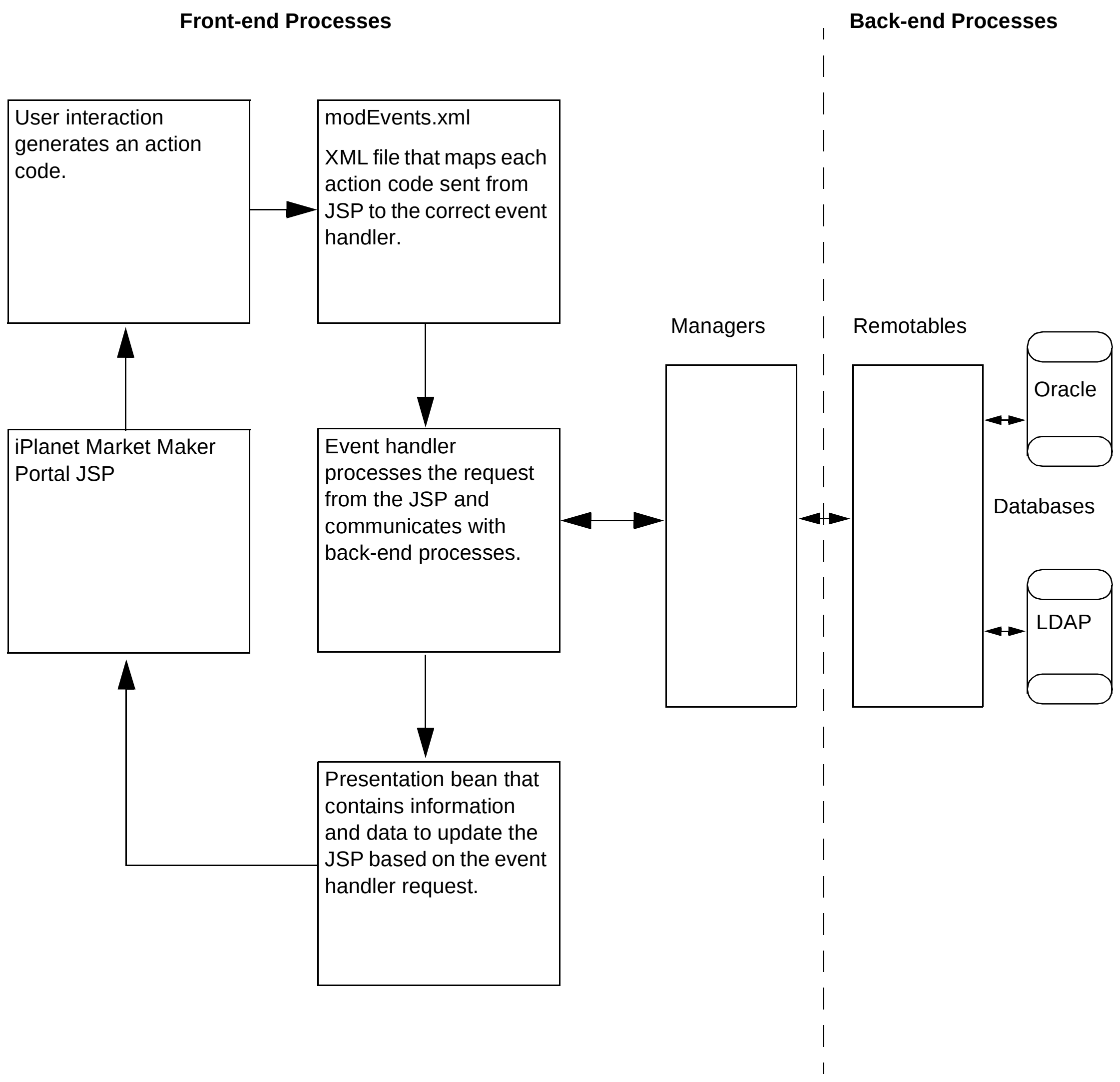
Event handling refers to the process of communicating from the user interface (JSPs) to the modules, then back to update the JSPs. A user interaction in a page, such as clicking on a button, needs to call the appropriate module functionality and return the results to the JSP. Event handling provides the mechanisms to complete these processes.

A user interaction in a JSP generates an action code. Each module has an XML file called `Events.xml` (prefixed with the module name) that maps each action code to a corresponding event handler. The event handler communicates with the back-end processes and creates an instance of a class called `PresentationBean` to update the JSP accordingly. This flow is shown in Figure 3-1.

In the iPlanet Market Maker back-end architecture, modules have components called “Managers” and “Remotables”. Managers are the point of access to the “fine-grained objects” within a module (a catalog, a catalog item, a community member, and so on). Remotables act on behalf of the fine-grained objects to provide efficient communication back and forth between the databases.

Note that event handlers are not managers or remotables, but communicate with them. Details about the iPlanet Market Maker architecture are not covered in this guide.

Figure 3-1 Overview of the Event Handling Flow


Front-end Processes
Back-end Processes
User interaction generates an action code.
modEvents.xml
XML file that maps each action code sent from JSP to the correct event handler.
Managers
Remotables
Oracle
iPlanet Market Maker Portal JSP
Event handler processes the request from the JSP and communicates with back-end processes.
Databases
LDAP
Presentation bean that contains information and data to update the JSP based on the event handler request.

Interpreting Event Handlers

Before you can extend the capabilities of event handlers, you need to understand how they work. This following list describes the events that take place when click on a link or button in the iPlanet Market Maker user interface.

- Each user action in a JSP generates an action code.
- Each action code invokes a corresponding event handler.
- The event handler implements the `IEventHandler` interface, which invokes the `execute()` method. For more information about the `execute()` method, see the "Using the execute() Call" section.
- The `execute()` method returns a `DisplayBean` Java object. This object specifies which JSP is to be displayed.
- The JSP contains HTML and Java code that defines how to display the elements in the `DisplayBean`.

To customize functionality, you need to identify the action codes, event handlers, `DisplayBeans`, and JSPs. To identify all these elements, you can turn on a tracing and debugging capability. To activate this capability, follow these steps.

1. In the `VortexConfiguration.properties` file, find the following lines.

```
#CFG_DEBUG_LEVEL (off=3, terse=7, verbose=15, vverbose=31 debug
window=40)

# - Debug level for the Market Maker system.

5=3
```

2. To turn on the debug window capability, change `5=3` to `5=40`.

3. In the `VortexConfiguration.properties` file, find the following lines.

```
#CFG_DEBUG_MODULES_TO_DEBUG. These are defined as follows:
MODULE_BASE=0,

#MODULE_RFX=1, MODULE_AUCTION=2, MODULE_CATALOG=3, MODULE_PRICING=4,

#MODULE_COMMUNITY=5, MODULE_DISPLAY=6, MODULE_OMS=7,
MODULE_CATALOG_IMPORT=8

# - Enable/Disable module level debugging.

#27[0]=0

#27[1]=1

#27[2]=2
```

```
#27[3]=3

#27[4]=4

#27[5]=5

#27[6]=6

#27[7]=7

#27[8]=8
```

4. Change `27[0] = 0` to `27[0] = 6`.

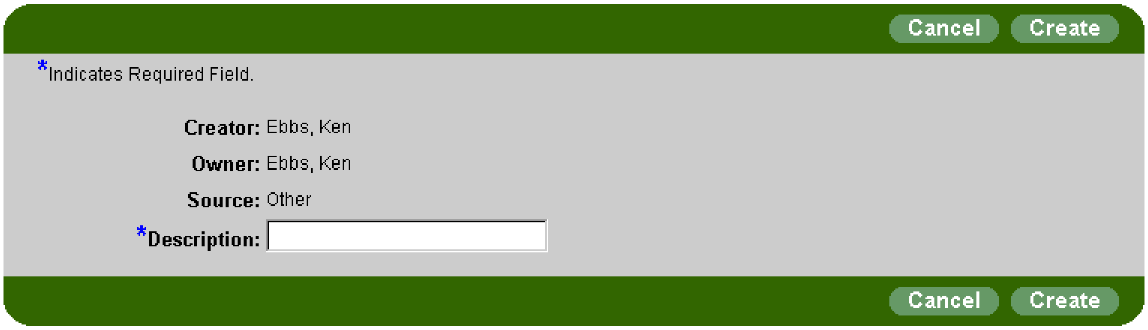
   Note that the value on the right side of the equal sign specifies the module to debug. The array index on the left side is just a placeholder to read in multiple values.

5. Restart the iPlanet Application Server (iAS) and the iPlanet Web Server (iWS).

   The debugging window is now enabled.

When you restart the iPlanet Market Maker software and click on a link or button, information about action codes, event handlers, `DisplayBeans`, and JSPs is displayed in the debug window.

Here is an example of a New Requisition form in the Order Management module.



*Indicates Required Field.

Creator: Ebbs, Ken

Owner: Ebbs, Ken

Source: Other

***Description:**

The following debug window output results when the Create Requisition JSP is displayed.

Note that the numbers in the left column are for documentation purposes and do not actually appear in the debug window.

```
-----------------------------------------------------------------

1. PARAMETER 'FilterByKeyword' = ''

PARAMETER 'FilterByEnum' = '-1'
PARAMETER ’FilterBySelector’ = ’0’

PARAMETER ’VDSP_PAGEID’ = ’63’

PARAMETER ’VDSP_UID’ = ’5b6abf:de3f060a01:-2222’

PARAMETER ’VDSP_ACTION’[0] = ’top,60012,4’

PARAMETER ’VDSP_ACTION’[1] = ’top,60012,4,top_c_Requisitions,70004,0’

2. Handler for code

com.iplanet.ecommerce.vortex.oms.display.OMSDisplayModule ACT REQ ADD REQ(70004)

com.iplanet.ecommerce.vortex.oms.display.ReqAddReqEventHandler

outgoing actionCode:

com.iplanet.ecommerce.vortex.oms.display.OMSDisplayModule ACT REQ ADD REQ(70004),
index = 0

3. JSP being displayed: oms/ReqAddReq.jsp

4. Display Bean:

       ReqAddLink      class com.iplanet.ecommerce.vortex.display.NVPLinkGen

       ReqCancelLink   class com.iplanet.ecommerce.vortex.display.NVPLinkGen

       Requisition     class com.iplanet.ecommerce.vortex.display.DisplayBean

            Creator

            Owner

            SourceType

-----------------------------------------------------------------------
```

1. The `PARAMETER` lines indicate the CGI name/value pairs that can be accessed by the `execute()` method of the event handler. They are not relevant in this example.

2. Action code 70004 for the New Requisition button triggered the generation of the Create Requisition page. The `execute()` method of the `ReqAddReqEventHandler` event handler was invoked to produce the `DisplayBean`.

3. `ReqAddReq.jsp` is the name of the JSP for the Create Requisition page. If you wanted to customize the Create Requisition page, `ReqAddReq.jsp` is the name of the file to edit.

4. The `DisplayBean` contains the following elements used by the JSP.

   - `ReqAddLink` of type `NVPLinkGen` corresponds to the Create URL.

- `ReqCancelLink` of type `NVPLinkGen` corresponds to the Cancel URL.
- `Requisition` is itself a `DisplayBean` with the following sub-elements. Note that indentation represents a nested `DisplayBean` object (a `DisplayBean` that contains as an element a display bean).

  `Creator` is the requisitions creator and corresponds to `Ebbs, Ken`.

  `Owner` is the requisitions owner and corresponds to `Ebbs, Ken`.

  `SourceType` is the source type of the requisition and corresponds to `Other`.

The `DisplayBean` with the contents shown in step 4 are passed to `oms/ReqAddReq.jsp`. This JSP contains the HTML and Java code that renders the contents of the `DisplayBean`.

Customizing JSPs

Before you customize the iPlanet Market Maker JSPs, it is important to understand how to edit and deploy them under iAS. The guidelines for customizing JSPs are as follows.

1. Do not edit JSPs directly in your production web server. Instead make and test changes in a development environment.
2. Make a backup copy of the JSP.
3. Edit the JSP with any HTML or text editor. Typically HTML editors do not work well for JSPs because of the amount of embedded Java code. In general when editing a JSP, be careful not to edit Java code.
4. After completing you edits, you can optionally compile the JSP. Because the JSP contains both HTML and Java code, this compilation ensures that you have not introduced syntax errors in the JSP. If you do not compile the JSP manually, iAS compiles it the first time it is accessed in a browser. The disadvantage of this approach is that you must view the iAS error log to see compilation errors.
5. Access the JSP in the browser. In most cases, you do not have to restart the application server, but if your changes do not appear, you try restarting iAS.

The following example shows the `ReqAddReq.jsp` JSP discussed in the "Interpreting Event Handlers" section:

```
<!-- Copyright © 2002 Sun Microsystems, Inc. All rights reserved.-->
<%@ include file="../include/OMSInclusionHeader.jsp"%>
<%PresentationBean bean =
(PresentationBean)request.getAttribute("DisplayBean");%>
<%DisplayBean rbean = bean.getDisplayBean(DF_REQUISITION_BEAN);%>
<%String requiredField = "<FONT CLASS=\"REQUIRED\">*</FONT>";%>
<FORM method=POST action="<%=bean.getPortalURL()%>"
name="<%=bean.getFormName()%>">
<TABLE BORDER="0" CELLSPACING="0" CELLPADDING="0" WIDTH="95%" ALIGN="center">
<TR>
<TD Class="mainTitle" ALIGN="left">OMS : Create Requisition<BR>&nbsp;</TD>
</TR>
<TR>
<TD>


   <TABLE BORDER="0" CELLSPACING="0" CELLPADDING="0"  HEIGHT="33"
WIDTH="100%">
   <TR>
   <TD Class="pagetitle" ALIGN="left" VALIGN="top" WIDTH="1%"><IMG HEIGHT="15"
SRC="/@IMM_DOCROOT@/images/curve_left.gif"></TD>
   <TD Class="pagetitle" ALIGN="left" VALIGN="middle">&nbsp;</TD>
   <TD Class="pagetitle" ALIGN="right" VALIGN="middle">

      <TABLE BORDER="0" CELLSPACING="0" CELLPADDING="0" HEIGHT="19">
      <TR>
       <TD NOWRAP Class="dark"><IMG
SRC="<%="/@IMM_DOCROOT@/images/buttons/"+FILE_PREFIX+"left.gif"%>"
BORDER="0"></TD>
   <TD NOWRAP Class="dark"><A Class="buttonText" href="JavaScript:document.<%=
bean.getFormName() %>.action =
   <%=bean.getNavigation(REQ_CANCEL_LINK) %>';
formSubmit(document.<%= bean.getFormName() %>,
'REQ_CANCEL_LINK');">Cancel</A></TD>
    <TD NOWRAP Class="dark"><IMG
SRC="<%="/@IMM_DOCROOT@/images/buttons/"+FILE_PREFIX+"right.gif"%>"
BORDER="0"></TD>
    <TD NOWRAP WIDTH="8" Class="medium">&nbsp;</TD>
    <TD NOWRAP Class="dark"><IMG
SRC="<%="/@IMM_DOCROOT@/images/buttons/"+FILE_PREFIX+"left.gif"%>"
BORDER="0"></TD>
    <TD NOWRAP Class="dark"><A Class="buttonText"
href="JavaScript:document.<%= bean.getFormName() %>.action =
'<%=bean.getNavigation(REQ_ADD_LINK) %>';
formSubmit(document.<%= bean.getFormName() %>,
'REQ_ADD_LINK');">Create</A></TD>
    <TD NOWRAP Class="dark"><IMG
SRC="<%="/@IMM_DOCROOT@/images/buttons/"+FILE_PREFIX+"right.gif"%>"
BORDER="0"></TD>
    </TR>
   </TABLE>
```

```
</TD>
   <TD Class="pagetitle" ALIGN="right" VALIGN="top" WIDTH="1%"><IMG
HEIGHT="15" WIDTH="17" SRC="/@IMM_DOCROOT@/images/curve_right.gif"></TD>
   </TR>
   </TABLE>
</TD>
</TR>
<TR>
<TD>
<TABLE  BORDER="0" CELLSPACING="0" CELLPADDING="0" WIDTH="100%"
ALIGN="center">
   <TR><TD width="3%">&nbsp;</TD><TD COLSPAN="2"><FONT
Class="required">*</FONT>Indicates Required Field.</FONT></TD><TD
width="3%">&nbsp;</TD></TR>
<TR><TD width="3%">&nbsp;</TD>
<TD ALIGN="center" COLSPAN="2">
<%@ include file="OMSError.jsp" %>
</TD><TD width="3%">&nbsp;</TD></TR>
  <TR>
  <TD width="3%">&nbsp;</TD>
  <TD Class="fieldtag" WIDTH="20%" HEIGHT="24" NOWRAP>Creator:&nbsp;
  </TD>
   <TD Class="fieldvalue" NOWRAP><%=rbean.getStringValue(DF_CREATOR)%>
   </TD>
   <TD width="3%">&nbsp;</TD>
     </TR>
        <TR><TD width="3%">&nbsp;</TD>
           <TD Class="fieldtag" WIDTH="20%" HEIGHT="24" NOWRAP>
             Owner:&nbsp;
            </TD>
           <TD Class="fieldvalue" NOWRAP>
             <%=rbean.getStringValue(DF_OWNER)%>
           </TD>
   <TD width="3%">&nbsp;</TD>
          </TR>
          <TR>
        <TD width="3%">&nbsp;</TD>
          <TD Class="fieldtag" WIDTH="20%" HEIGHT="24" NOWRAP>
                Source:&nbsp;
          </TD>
          <TD Class="fieldvalue" NOWRAP>
            <%=rbean.getStringValue(DF_SOURCE_TYPE)%>
          </TD>
        <TD width="3%">&nbsp;</TD>
          </TR>
          <TR>
        <TD width="3%">&nbsp;</TD>
              <TD Class="fieldtag" WIDTH="20%" HEIGHT="24" NOWRAP">
            <%=requiredField%>Description:&nbsp;<%=checkError(rbean,
DF_DESCRIPTION)%>
            </TD>
            <TD Class="fieldvalue" NOWRAP>
                    <INPUT TYPE=TEXT size="25"
NAME="<%=rbean.getRealKey(DF_DESCRIPTION)%>">
    </TD>
    <TD width="3%">&nbsp;</TD>
       </TR>
       </TABLE>
<BR>
   <TABLE BORDER="0" CELLSPACING="0" CELLPADDING="0"  HEIGHT="33" WIDTH="100%"
ALIGN="center">
       <TR>
        TD Class="pagetitle" ALIGN="left" VALIGN="bottom" WIDTH="1%"><IMG
HEIGHT="15"
SRC="/@IMM_DOCROOT@/images/curve_bottom_left.gif"></TD>
       <TD Class="pagetitle" ALIGN="left" VALIGN="middle">&nbsp;</TD>
       <TD Class="pagetitle" ALIGN="right" VALIGN="middle">
           <TABLE BORDER="0" CELLSPACING="0" CELLPADDING="0" HEIGHT="19">
       <TR>
        <TD NOWRAP Class="dark"><IMG
SRC="<%="/@IMM_DOCROOT@/images/buttons/"+FILE_PREFIX+"left.gif"%>"
BORDER="0"></TD>
       <TD NOWRAP Class="dark"><A Class="buttonText"
href="JavaScript:document.<%= bean.getFormName() %>.action =
'<%=bean.getNavigation(REQ_CANCEL_LINK) %>';
formSubmit(document.<%= bean.getFormName() %>,
'REQ_CANCEL_LINK');">Cancel</A></TD>
       <TD NOWRAP Class="dark"><IMG
SRC="<%="/@IMM_DOCROOT@/images/buttons/"+FILE_PREFIX+"right.gif"%>"
BORDER="0"></TD>
  <TD NOWRAP WIDTH="8" Class="medium">&nbsp;</TD>
        <TD NOWRAP Class="dark"><IMG
SRC="<%="/@IMM_DOCROOT@/images/buttons/"+FILE_PREFIX+"left.gif"%>"
BORDER="0"></TD>
  <TD NOWRAP Class="dark"><A Class="buttonText" href="JavaScript:document.<%=
bean.getFormName() %>.action =
'<%=bean.getNavigation(REQ_ADD_LINK) %>';
formSubmit(document.<%= bean.getFormName() %>,
'REQ_ADD_LINK');">Create</A></TD
        <TD NOWRAP Class="dark"><IMG
SRC="<%="/@IMM_DOCROOT@/images/buttons/"+FILE_PREFIX+"right.gif"%>"
BORDER="0"></TD>
     </TR>
      </TABLE>
   /TD>
       <TD Class="pagetitle" ALIGN="right" VALIGN="bottom" WIDTH="1%"><IMG
HEIGHT="15" WIDTH="17"
SRC="/@IMM_DOCROOT@/images/curve_bottom_right.gif"></TD>
     </TR>
      </TABLE>
</TD>
</TR>
<TR><TD NOWRAP WIDTH="760" BGCOLOR="#FFFFFF"><!--this row is to force the
minimum table width to 760-->&nbsp;</TD></TR>
</TABLE>
<%=bean.getButtonInfo()%>
</FORM>
```

Here is a list of conventions and comments about this JSP that is relevant to customization.

- Characters surrounded by `<!--  ...-->` characters are comment.
- Characters surrounded by `<%= ... %>` indicate Java code.
- .HTML cannot be embedded in Java code blocks (`<%= ... %>`).
- `<%PresentationBean bean = (PresentationBean)request.getAttribute("DisplayBean");%>` is Java code that sets up a variable called bean. This variable contains the `DisplayBean` created by the event handler invoked prior to rendering the JSP. In this case, looking at the debugging output shown in the "Interpreting Event Handlers" section, the event handler is `ReqAddReqEventHandler`. This pattern is common to all iPlanet Market Maker JPSs.
- `<%DisplayBean rbean = bean.getDisplayBean(DF_REQUISITION_BEAN);%>` creates a variable called `rbean,` which is a nested `DisplayBean` nested within `bean`. The `DF_REQUISITION_BEAN` field is a Java String constant.
- `<%=rbean.getStringValue(DF_CREATOR)%>` shows an example of fetching a String field from a `DisplayBean`.
- .It is possible to subclass the `ReqAddReqAddEventHandler` event handler, override the `execute()` method, add your own fields to the `DisplayBean` (using `DisplayBean.put("myField", "myValue")`). This data can then be rendered in the JSP using `bean.getStringValue("myField")`.
- Be very careful when editing Java code within the JSPs.

A JSP compilation tool is provided with iPlanet Market Maker. This tool requires iAS to do its compilation:

```
com.iplanet.ecommerce.vortex.tools.JspCompiler.IMMJspCompiler
```

The arguments are as follows.

`-d <jsp directory>` is the fully qualified path to the directory for the JSP.

`-u <jsp url>` is the URL to compile

`-t 3`

`-h <ias host>` is the host on which iAS is running (for example, `localhost`).

`-p <iws port>` is the iWS http port (for example, 80)

Community Custom Fields

Attributes in the iPlanet Market Maker Community Module

The Community module in iPlanet Market Maker 4.0 SP1 (and onward) has fifty custom attributes for the company and user object. The schema and backend API's are now supporting the `custom0..49` attributes. The DataBeanCompany & DataBeanUser also includes these custom attributes.

To use these custom attributes, the following steps must be performed:

To add a new field called *Company Notification Fax* for company, select one of these fifty custom attributes.

For example, if you choose `custom0`:

1. Edit the `CompanySub.jsp` and uncomment the first attribute `Custom0`.
2. Change the text `Custom0` to your `Company Notification Fax` to show this in the UI.
3. Deploy it again on iAS and restart the server.

When you login as *immhost* or *company admin* you see the new field `Company Notification Fax` in the Company Profile fax. If you update this new attribute, it is stored in LDAP, under the attribute `immCustom0`, under the company name. For example, if you updated this attribute for the company *sun* then you can see the value for *sun* in the attribute `immCustom0` in LDAP server.

Similarly, for the user you need to make changes in the you can make changes `userProfile.jsp`, `userDetail.jsp`, and `registerUser.jsp`.

The following JSP's are already modified and have `custom0..9` attributes. These `custom0..9` attributes are commented out and you need to uncomment them to use them. If you need more than ten custom attributes, add your custom attributes in these JSP's, similar to the way you have already added `custom0..9` attributes.

For the company, you need to modify only `CompanySub.jsp`, and for the user, you need to modify the remaining three JSP's.

a. `userProfile.jsp`

b. `userDetail.jsp`

c. `CompanySub.jsp`

d. `registerUser.jsp`

All of the other changes for these custom attributes are already done and you do not need to change anything other than JSP's.

Mapping Action Codes to Event Handlers

The mapping between the action codes and event handlers described in the "Interpreting Event Handlers" section is defined in the module `Events.xml` file. `ModuleJSP` uses the following method to return the XML file.

```
protected String getMapFile();
```

This method returns the name of the XML file, such as `AuctionEvent.xml`. Here is a mapping from the `AuctionEvents.xml` file.

```
<?xml version="1.0"?>
<EventMap class="com.iplanet.ecommerce.vortex.auction.display.AucJSP">
...
<EventHandler symcode="ACT_INITIATOR_VIEW_BROWSE"
class="com.iplanet.ecommerce.vortex.auction.display.EventHandlerInitiatorViewB
rowse" />
```

The AuctionEvent.xml file is enclosed in an <EventMap> tag. This tag can have a class attribute and an offset attribute (not shown in this example). The class attribute specifies the class used to look up all the symbolic names used inside that class. The OFFSET attribute is a numerical offset added to the integer values of action codes (not the symbolic names) referred to in the AuctionEvent.xml file. For an example of how OFFSET values are added to action code values, see "Defining a New Action Code and Event Handler" in Chapter 4.

You can use an <Include> tag to share common events between modules. This tag has a single attribute, file, that points to another XML file containing another set of action codes and event handlers. This file is read into the XML file during processing.

The Event.xml file is processed sequentially, so that subsequent <EventHandler> and <Include> tags that specify the same actions override earlier settings. This sequencing lets you include a set of actions and selectively override them.

The <EventHandler> ACT_INITIATOR_VIEW_BROWSE attribute is the symbolic name for the action code. EventHandlerInitiatorViewBrowse is the event handler. You can map different action codes to the same event handler, but you cannot map an action code to more than one event handler.

The event handler implements an interface called IEventHandler that constructs an instance of a class called a PresentationBean. This class is a look-up table that provides the data and link information needed to determine the next JSP to be displayed.

The IEventHandler interface implements a method called execute() that returns a PresentationBean that is passed an instance of an interface called IEventContext. The execute() method call uses IEventContext to obtain information such as parameters from the request, session information, profile data, interfaces to construct the appropriate portal-based URLs, and so on.

The ModuleJSP class maps an incoming action code to the right event handler, calls its exectue() method, and dispatches the appropriate JSP as content to be displayed. The PresentationBean returned by the event handler is placed into the request under a standard name, allowing the right channel in the JSP to obtain it. The channel information appears in the AVAILITEM of a display profile for the portal. Display profiles are XML documents that define the contents of the iPlanet Market Maker portal. For information about display profiles, see the iPlanet Market Maker online help.

Using Event Handlers

Each module has a single derivation of the `ModuleJSP` subclass. For each action you define in a portal JSP, you need a corresponding action code. It is important to make each of your action code numbers unique. You usually define these action codes in a single "constants" class. Consider the following entry in the `CatalogEvents.xml` file.

```
<EventHandler symcode="ACT_CATALOG"
class="com.iplanet.ecommerce.vortex.catalog.display.ViewsDataEventHandler" />
```

The action code integer definition for the `ACT_CATALOG` symbolic action code name is in the `IDisplayConstants.java` file for the Catalog module:

```
/**

 * Field

 * @audience

 */

public static final int ACT_CATALOG = OFFSET + 1;
```

You can edit the `Event.xml` files and "constants" classes to substitute your own event handlers for iPlanet Market Maker implementations or add new event handlers to implement new actions in a portal JSP. For an example of an implementation, see Chapter 4.

Using the execute() Call

An event handler uses an `execute()` call to pass information to a `DisplayBean`, which in turn passes information to a `PresentationBean` to represent that information in the JSP. The `execute()` call uses a public interface called `IEventContext` to pass this information. A typical code line for the `execute()` call is as follows.

```
public DisplayBean execute(IEventContext ctx) throws VortexException {
```

The code for the `IEventContext` public interface is as follows.

```
package com.iplanet.ecommerce.vortex.display;

import com.iplanet.ecommerce.vortex.arch.VortexException;

import com.iplanet.ecommerce.vortex.arch.Context;

import com.iplanet.ecommerce.vortex.display.Session;

import java.util.Enumeration;

/**
```

```
 * This class represents the interface which defines the methods an event
 * context object must contain. This class is used by the IEventHandler.execute().
 * Currently JSPEventContext implements this Interface.
 */
public interface IEventContext extends IParameterMap
{
     /**
     * This method retrieves the session.
     * @return  the session.
     * @throws VortexException
     * @audience user
     * @audience vortex-logic
     */
     public IVxDspSession getSession() throws VortexException;
     /**
     * Method getDisplayBeansFromSession
     * safely get an array of DisplayBeans from session
     * @param key
     * @return  the array of DisplayBeans from session or null
     * @throws VortexException
     * @audience user
     * @audience vortex-logic
     */
     public DisplayBean[] getDisplayBeansFromSession(String key)
         throws VortexException;
     /**
     * This method safely retrieves a DisplayBean from a session.
     * @param key
     * @return  the DisplayBean from session or null
     * @throws VortexException
     *
* @audience user
* @audience vortex-logic
*/
public DisplayBean getDisplayBeanFromSession(String key)
* @throws VortexException;
/**
* This method puts a DisplayBean array in session.
* @param key
* @param beans
* @throws VortexException
* @audience user
* @audience vortex-logic
*/
public void putDisplayBeansInSession(String key, DisplayBean[] beans)
  throws VortexException;
/**
* This method puts a DisplayBean in session.
* @param key
* @param bean
* @throws VortexException
* @audience user
* @audience vortex-logic
*/
public void putDisplayBeanInSession(String key, DisplayBean bean)
} throws VortexException;
/**
* This method retrieves the context.
* @return  the context.
* @audience user
* @audience vortex-logic
*/
```

```
public Context getContext();
/**
* This method retrieves the underlying portal display element. Essentially,
* the "escape valve" back to the portal framework from the more specific
* Market Maker implementation.  Expected common uses of this method are
* covered by some of the other methods which should be used instead.
*
* @return  the element.
* @audience user
* @audience vortex-logic
*/
public DisplayElement getSource();
/**
* This method retrieves the action code.
* @return  the action code.
* @throws VortexException
* @audience user
* @audience vortex-logic
*/
public int getActionCode() throws VortexException;
/**
* This method retrieves the action code index.
* @return  the action code index.
* @throws VortexException
* @audience user
* @audience vortex-logic
*/
public int getActionCodeIndex() throws VortexException;
/**
* This method retrieves the portal URL.
* @return  the portal url.
* @audience user
* @audience vortex-logic
*/
public String getPortalURL();
/**
* This method refreshes the information display profile.
* @return  true, if this is a refresh case.
* @throws VortexException
* @audience user
* @audience vortex-logic
*/
public boolean isRefreshable() throws VortexException;
/**
* This method retrieves an object from the display profile
* @param   key the lookup key.
* @return  object or null
* @throws VortexException
* @audience user
* @audience vortex-logic
*/
public Object getProfile(String key) throws VortexException;
/**
* This method removes an object from the display profile.
* @param   key the object to remove
* @throws VortexException
* @audience user
* @audience vortex-logic
*/
public void removeProfile(String key) throws VortexException;
/**
* This method places an object in the display profile.
* @param   key the lookup key.
* @param   value the object to place in the profile
* @throws VortexException
* @audience user
* @audience vortex-logic
*/
public void setProfile(String key, Object value) throws VortexException;
/**
* This method sets the action code.
* @param   code the code.
* @throws VortexException
* @audience user
* @audience vortex-logic
*/
public void setActionCode(int code) throws VortexException;
/**
* This method sets the action code index.
* @param   code the code index.
* @param index
* @throws VortexException
* @audience user
* @audience vortex-logic
*/
public void setActionCodeIndex(int index) throws VortexException;
/**
*  This method forwards the request to the EventHandler that is mapped to the
*  given code. Processing will continue as normal inside the
*  EventHandler that is targeted. When processing is returned, the
*  DisplayBean has been set for the forwarded EventHandler. The setting of
*  the DisplayBean a second time will override the setting made
*  by the forwarded EventHandler.
```

```
*
*  @param code     The action code to which you should forward the request.
*  @return  The DisplayBean created by the forwarded EventHandler
*
* @throws VortexException
* @audience user
* @audience vortex-logic
*/
public DisplayBean forward(int code) throws VortexException;
/**
* This method obtains an NVPLinkGen for use in constructing navigation.
* @return
* @audience user
* @audience vortex-logic
*/
public NVPLinkGen makeLinkGenerator();
/**
* This method retrieves the attribute access.
*
* @param   name the attribute key
* @return  the desired attribute, or null.
* @audience user
* @audience vortex-logic
*/
public Object getAttribute(String name);
/**
* This method sets the attribute setting.
* @param   name the attribute key
* @value   the attribute object
* @param value
* @audience user
```

```
    * @audience vortex-logic
    */
    public void setAttribute(String name, Object value);
    /**
    * This method retrieves attribute names.
    * @return  the attribute names
    * @audience user
    * @audience vortex-logic
    */
    public Enumeration getAttributeNames();
    /**
    * This method obtains the remote IP address
    * @return  remote address
    * @throws VortexException
    * @audience user
    * @audience vortex-logic
    */
    public String getRemoteAddr() throws VortexException;
}
```

Invoking and Redisplaying Event Handlers

As stated earlier, the `ModuleJSP` class invokes an event handler based on an action code setting. The action code might represent one of three things:

- An initial setting driven from the display profile the first time the channel was displayed. If defaulted, it is -1.
- An explicit code driven from the submit button or href placed in the JSP.
- Redisplay.

Redisplay requires some explanation. Certain portal actions, like tabbing onto a display for a channel, or handling an action for another channel displayed on the page along with your channel, might result in an `exectue()` call with no explicit action code for the channel in the URL. In this case, `execute()` is called with the same action code it had the last time it was invoked.

Three mechanisms aid the `execute()` method in handling the situation:

- The `IEventContext` passed to the `exectue()` mechanism includes a method `isRefreshable()`, which returns boolean true if the channel is being redisplayed. This allows the event handler to behave appropriately by restoring data cached in the session, not performing a requested DB update for a redisplay.
- The `IEventContext` contains a `setActionCode()` method that allows the event handler to control the recorded code that is invoked if a specific code was not specified through the URL. This, for instance, allows event handlers for transitional state to ensure that those transitional states are never reinvoked unless explicitly requested.
- For channels displayed in tabbed containers, there is a special property called `autohoming` that can be set for a channel in the display profile. If the channel is autohomed, whenever it is freshly tabbed onto, it is reset to its default action code.

Relating Event Handlers to JSPs

As stated earlier, the `PresentationBean` returned by the `execute()` method is made available to the JSP under a standard name registered as an attribute on the request. The module JSPs generally contain an initial scriptlet with a line like:

```
PresentationBean presBean = (PresentationBean) request.getAttribute(DISPLAY_BEAN);
```

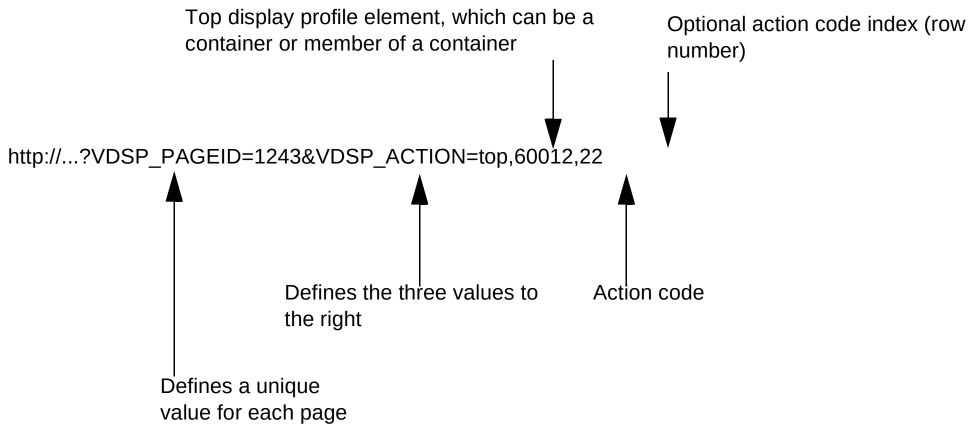
This line lets information from the `PresentationBean`, which is a descendant of the `DisplayBean`, be displayed in the JSP. For a JSP code example, see “Displaying the Added JSP” in Chapter 4.

Further beans, as well as many other types of data, can be registered in this object, allowing the JSP access to it.

Currently, there is not an explicit listing of the JSP displayed for a given action code. Part of the reason for this is that because the event handler determines the JSP to be displayed, it could be switched dynamically on the basis of the logic in the handler, so there is not always a one-to-one correspondence (there usually is, but not always).

You can determine the relationship between the event handlers and JSPs in two ways:

- Examine the JSP and the names used in them to generate the presentation. The action code appears in the URL after you perform an action in the JSP. For example, the following action code number appears after you log in as a buyer and select Catalog – Browse (the iPlanet Market Maker URL has been omitted and replaced by “...”).


Top display profile element, which can be a container or member of a container
Optional action code index (row number)
http://...?VDSP_PAGEID=1243&VDSP_ACTION=top,60012,22
Defines the three values to the right
Action code
Defines a unique value for each page

- Turn on terse debugging for the display module (debug level 7 for module 6). For details about how to turn on terse debugging, see the “Interpreting Event Handlers” section.

Other Resources for Customizing Event Handlers

The JavaDoc's for the following classes are also help good resources for understanding event handlers.

- `IEventHandler` - The interface implemented by an event handler.
- IEventContext and superclass `IParameterMap` - The context interface passed to the event handler's `execute()` method.
- `Context` -The context object for backend operations, setup by `ModuleJSP` and made available to the `execute()` method.
- `DisplayBean` and subclass `PresentationBean` -The data packager used to transfer data between the `execute()` method and the JSP.
- `LinkGen` (superclass of NVPLinkGen) -The helper class used to create URL's within the Portal Framework.
- `Session` - The session abstraction used inside iPlanet Market Maker.
- `Date, Price, LocalizedEnum` - Specialized datatypes used in conjunction with the DisplayBean.

- `DisplayField`, `FieldType` - Classes used in conjunction with field validation in `DisplayBean`.

Configuring the Debug Channel

When you go to iPlanet Market Maker in your browser, a popup window is launched. Every time you click a link in an iPlanet Market Maker window, a popup window gives you the following information:

- A list of all the http request parameters and values.
- The symbolic and numeric action code associated to this link.
- The name of the event handler that is invoked, with a link to the javadoc for that Event Handler. This works if the javadoc is installed on your iPlanet Market Maker instance in the `doc/userdocs` directory. The event handler javadoc is where all the information related to customization of this event handler is located, namely the produced DisplayBean and its content.
- Eventually, the name of the event handler to which the action is forwarded.
- The name of the JSP invoked to render the content created by the event handler.
- A serialization of the display bean, with all the fields, their types and values.

To enable the debug channel:

1. Set the DebugLevel to 40 (instead of the default of 3) in the following file:

   <*ias_install_dir*>`/ias/APPS/imm40/imm40/classes/VortexConfiguration.properties`

   For example:

   ```
   # - Debug level for the iPlanet Market Maker system
   5=40W
   ```

2. After you have edited this file, restart your web server.

   When you go to the iPlanet Market Maker site in your browser, a popup should be launched with all the information you need.

Chapter 4

Customizing the Auction Module

This chapter contains the following sections:

- Overview
- Extending Event Handling for the Auction Bidding Page

Overview

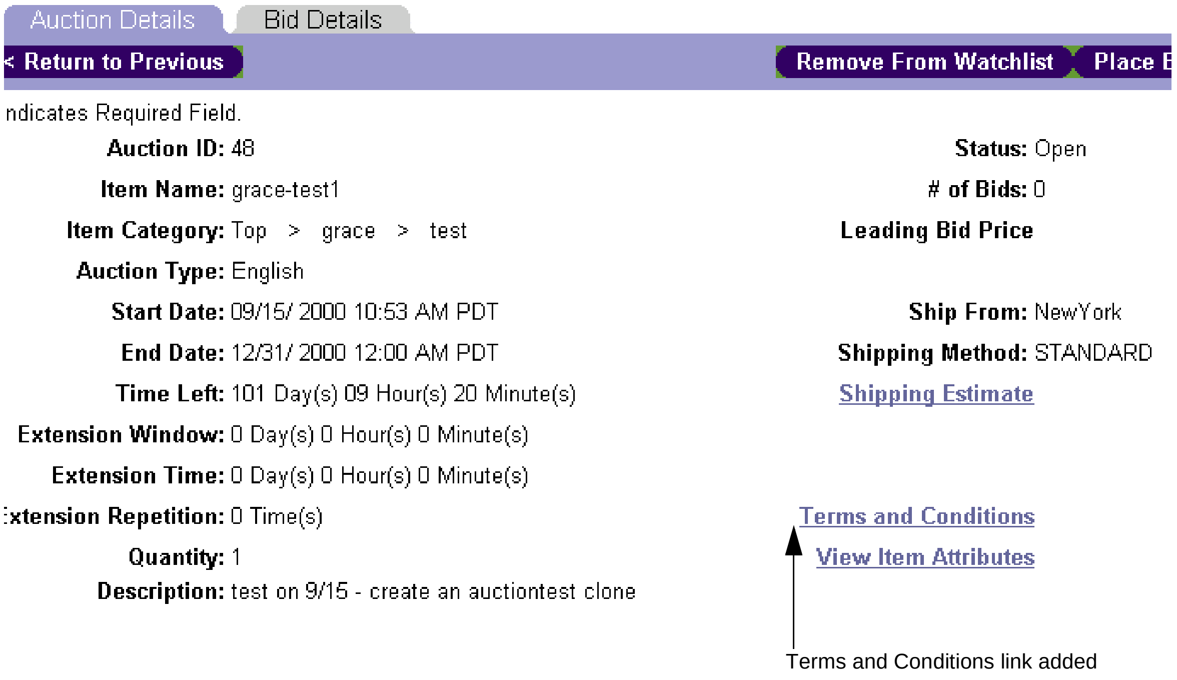
“Extending Event Handling” in Chapter 3 described the general process for customizing modules. This chapter describes a customization for the Auction module. It describes how to add a navigation link to an existing page through the extension of an existing event handler. It also describes how to add a new page flow by creating a new event handler and JSP.

For additional information about how the event handling mechanisms work, refer to the comments in the code examples in this chapter.

The example in this chapter adds a `Terms and Conditions` link to the Auction Bidding page that links to a Terms and Conditions page. The Terms and Conditions page provides a link to return to the Auction Bidding page. The content of the Terms and Conditions page is specified by the Auction Initiator when the auction is initiated and is not covered in this chapter.

Figure 4-1 shows part of the Auction Bidding page with a `Terms and Conditions` link.

Figure 4-1 Auction Bidding Page with Terms and Conditions Link



ndicates Required Field.

Auction ID:	48	**Status:**	Open
Item Name:	grace-test1	**# of Bids:**	0
Item Category:	Top > grace > test	**Leading Bid Price**	
Auction Type:	English		
Start Date:	09/15/ 2000 10:53 AM PDT	**Ship From:**	NewYork
End Date:	12/31/ 2000 12:00 AM PDT	**Shipping Method:**	STANDARD
Time Left:	101 Day(s) 09 Hour(s) 20 Minute(s)		
Extension Window:	0 Day(s) 0 Hour(s) 0 Minute(s)		
Extension Time:	0 Day(s) 0 Hour(s) 0 Minute(s)		
xtension Repetition:	0 Time(s)		
Quantity:	1		
Description:	test on 9/15 - create an auctiontest clone		

Extending Event Handling for the Auction Bidding Page

As described in “Interpreting Event Handlers”in Chapter 3, each user interaction is mapped to an action code and event handler. So the Terms and Conditions link in Figure 4-1 needs to have an action code and event handler to be able to link to the Terms and Conditions page. The following example reassigns an existing action code to a new event handler. It also creates a new action code and event handler.

The mapping of action codes to event handlers is defined in an XML file, which is shown in the following `SunBidAuctionEvents.xml` file.

```
<?xml version="1.0"?>

<EventMap class="com.sun.ecommerce.sunbid.auction.display.AucJSPSB">

<!-- Override the event handler for the Auction Bidding screen -->

<EventHandler symcode="ACT_BIDDER_BID"
class="com.sun.ecommerce.sunbid.auction.display.EventHandlerBidderBidSB" />

<!--Add a new event handler for the Terms and Conditions page -->

<!-- Java constant ACT_BIDDER_TANDC_DISPLAY is defined in AuctionConstants.java -->

<EventHandler symcode="ACT_BIDDER_TANDC_DISPLAY"
class="com.sun.ecommerce.sunbid.auction.display.EventHandlerCommonTandC" />

</EventMap>
```

The `SunBidAuctionEvents.xml` file first calls the `AucJSPSB` class to ensure that the constants defined in the `AuctionConstants` class can be referenced by the `SunBidAuctionEvents.xml` file. The code for the `AucJSPSB` class is as follows.

```
public class AucJSPSB extends AucJSP implements AuctionConstants {

    // intentionally left blank. This class is created to ensure the
// constants defined in AuctionConstants can be referenced in
// SunBidAuctionEvents.xml

}
```

The code for the AuctionConstants class is as follows.

```
public interface AuctionConstants {

// Define offsets with USER_OFFSET to ensure custom action codes do not conflict
// with base product action codes.  Action codes must be unique

public static final int USER_OFFSET =
com.iplanet.ecommerce.vortex.auction.AuctionModule.USER_OFFSET;

public static final int OFFSET = USER_OFFSET;

// Define custom action codes for the Terms and Conditions Page

public static final int ACT_BIDDER_TANDC_DISPLAY = OFFSET + 3;

}
```

Assigning an Existing Action Code to a New Event Handler

Next the SunBidAuctionEvents.xml file assigns the existing action code "ACT_BIDDER_BID" to a new EventHandlerBidderBidSB class to extend the existing functionality of the class originally assigned to"ACT_BIDDER_BID". The EventHandlerBidderBidSB class uses the existing EventHandlerBidderBid class and adds new functionality to it.

The code for the EventHandlerBidderBidSB class is as follows.

```
/**
 * This class is the event handler for ACT_BIDDER_BID
 */
public class EventHandlerBidderBidSB extends EventHandlerBidderBid
    implements IEventHandler, AuctionConstants {
    public DisplayBean execute(IEventContext ctx) throws VortexException {
    Guid auctionGuid;
    // Invoke the parent event handler
    PresentationBean bidScreenBean = (PresentationBean) super.execute(ctx);
    if (ctx.isRefreshable()) { // when REDISPLAYing
    auctionGuid = getSavedParamGuid(ctx, ATTR_ROWBEAN_GUID_STRING,
    INVALID_GUID);
    } else {
    auctionGuid = getParamGuid(ctx, ATTR_ROWBEAN_GUID_STRING,
    INVALID_GUID);
    }
        // Create Terms and Conditions link
    NVPLinkGen tandcDisplayLink = ctx.makeLinkGenerator();
    tandcDisplayLink.addAction(ACT_BIDDER_TANDC_DISPLAY);

tandcDisplayLink.addNVPair(ATTR_ROWBEAN_GUID_STRING,auctionGuid.toString()
);
```

```
    bidScreenBean.put(ATTR_BIDSCREENBEAN_TANDC_DISPLAY_LINK,
tandcDisplayLink);
    }
}
```

Defining a New Action Code and Event Handler

Finally the `SunBidAuctionEvents.xml` file defines a new action code `"ACT_BIDDER_TANDC_DISPLAY"` mapped to a new event handler `"EventHandlerCommonTandC"`. Note that in the `AuctionsConstants` class shown in the "Extending Event Handling for the Auction Bidding Page" section, a offset range predefined in `AuctionModule.USER_OFFSET` is added to the action code value of 3 that corresponds to the `"ACT_BIDDER_TANDC_DISPLAY"` symbolic action code name. This offset value ensures that each action code number is unique.

The code for the `"EventHandlerCommonTandC"` class is as follows.

```
public class EventHandlerCommonTandC extends EventHandlerAuction
implements IEventHandler {
    public DisplayBean execute(IEventContext ctx) throws
VortexException {
        Guid auctionGuid;
        PresentationBean presentationBean = new
PresentationBean(ctx);
        // Create data beans and extract data
        DataBeanTandC dataBeanTandC = new DataBeanTamdC(ctx);
    auctionGuid = getSavedParamGuid(ctx,
ATTR_ROWBEAN_GUID_STRING, INVALID_GUID);
    dataBeanTandC =
(DataBeanTandC)readFromSession("SESS_AUCTIONTANDCBEAN",
                AuctionConstants.ACT_COMMON_TANDC_DISPLAY, ctx);
    } else {
         auctionGuid = getParamGuid(ctx,
ATTR_ROWBEAN_GUID_STRING, INVALID_GUID);
// Grab the Terms and Condition content based the auctionGuid
from
// the Auction object
            // Customization of back-end objects not covered in
this example.
            String tandc = "Terms and Conditions goes  HERE!";
            dataBeanTandC.setTandC(tandc);
          }
            // create Return to Previous link
         NVPLinkGen prevLink = ctx.makeLinkGenerator();
         prevLink.addNVPair(ATTR_ROWBEAN_GUID_STRING,
auctionGuid.toString() );
```

```
         prevLink.addAction(ACT_BIDDER_BID);
         // populate the returning PresentationBean
         presentationBean.put(ATTR_BIDSCREENBEAN_PREV_LINK, 
prevLink);
         resentationBean.put("DataBeanTandC", dataBeanTandC);
         // Prepare for REDISPLAY - save paramMap in session
         String params[] = { ATTR_ROWBEAN_GUID_STRING };
         String values[] = { auctionGuid.toString() };
         saveParamMap(params, values, ctx);
         // Save AuctionTandCBean to session
         writeToSession("SESS_AUCTIONTANDCBEAN",
             AuctionConstants.ACT_COMMON_TANDC_DISPLAY, 
dataBeanTandC, ctx);
         presentationBean.setJSP("auction/CommonTandC.jsp");
         return presentationBean;
     }
}
```

Displaying the Added JSP

The event handler `EventHandlerBidderBidSB` calls the `CommonTandC.jsp` to display the Terms and Conditions page. The code for the `CommonTandC.jsp` is as follows.

```
<%@ include file="../include/AuctionInclusionHeader.jsp" %>
<%@ page 
import="com.iplanet.ecommerce.vortex.community.display.*" %>
<%@ page 
import="com.sun.ecommerce.sunbid.auction.display.DataBeanTandC" 
%>
<%@ page 
import="com.sun.ecommerce.sunbid.auction.display.AuctionConstant
s" %>
<%
     PresentationBean bean =
                 (PresentationBean) 
request.getAttribute("DisplayBean");
<%-- Get the Terms and Conditions DataBean --%>
                 DataBeanTandC dataBeanTandC =
                 (DataBeanTandC)bean.getDisplayBean("DataBeanTan
dC");
%>
<FORM ACTION="<%= bean.getPortalURL() %>" name="<%= 
bean.getFormName() %>" METHOD="post">
<TABLE BORDER="0" CELLSPACING="0" CELLPADDING="0" WIDTH="95%" 
ALIGN="center">
<TR>
<TD CLASS="mainTitle" ALIGN="left">Auctions : Terms and 
Conditions<BR>&nbsp;</TD>
```

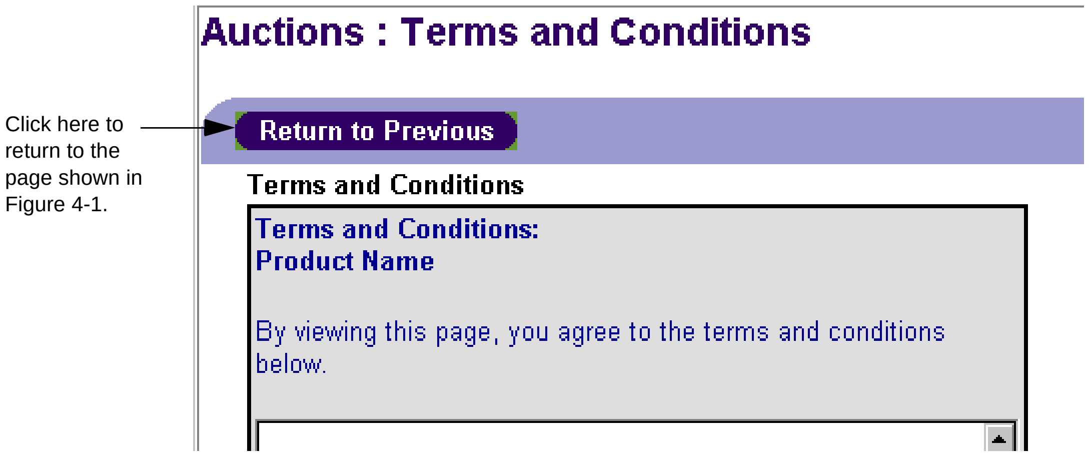
```
</TR>
<TR>
<TD>
        <TABLE BORDER="0" CELLSPACING="0" CELLPADDING="0"
HEIGHT="33" WIDTH="100%">
        <TR>
        <TD CLASS="pagetitle" ALIGN="left" VALIGN="top"
WIDTH="1%"><IMG HEIGHT="15" SRC="images/curve_left.gif"></TD>
        <TD CLASS="pagetitle" ALIGN="left" VALIGN="middle">
                <TABLE BORDER="0" CELLSPACING="0" CELLPADDING="0"
HEIGHT="19">
                <TR>

                     <TD NOWRAP CLASS="dark"><IMG
SRC="images/buttons/b_left.gif" BORDER="0"></TD>
                     <%-- Return To Previous link --%>
                      <TD NOWRAP CLASS="dark"><A
CLASS="buttonText" HREF="<%=
bean.getNavigation(ATTR_BIDSCREENBEAN_PREV_LINK) %>">< Return to
Previous</A></TD>
                     <TD NOWRAP CLASS="dark"><IMG
SRC="images/buttons/b_right.gif" BORDER="0"></TD>
                    </TR>
                    </TABLE>
         </TD>
         <TD CLASS="pagetitle" ALIGN="right" VALIGN="top"
WIDTH="1%"><IMG HEIGHT="15" WIDTH="17"
SRC="images/curve_right.gif"></TD>
         </TR>
         </TABLE>
</TD>
</TR>
<TR>
<TD><!-- begin the inner nested table-->
        <TABLE  BORDER="0" CELLSPACING="0" CELLPADDING="0"
WIDTH="100%">
        <TR>
        <TD WIDTH="3%">&nbsp;</td>
        <TD WIDTH="94%">
        <!--##############enter content
here###################-->
<TABLE BORDER="0" WIDTH="90%" CELLSPACING="0" CELLPADDING="0">
    <TR>
        <TH CLASS="menu" height=20 align="left">
            Terms and Conditions
        </TH>
   </TR>
</TABLE>
<TABLE BORDER="0" WIDTH="90%" CELLSPACING="0" CELLPADDING="0">
   <TR>
     <TD CLASS="fieldvalue" nowrap>
     <form>
     <table width="50%" border=0 cellspacing=0 cellpadding=2>
     <tr>
     <td width="50%" ALIGN="RIGHT" BGCOLOR="#000000">
```

```
        <table width="100%" border=0 cellspacing=0 cellpadding=2>
        <tr>
        <td BGCOLOR="#DFDFDF" ALIGN=left>
                      <FONT COLOR="darkblue">
                      <STRONG>
                      Terms and Conditions:<BR>
                      Product Name
                      </STRONG>
                      </FONT>
                      <FONT COLOR="darkblue" SIZE="-1">
                      <P>
                      By viewing this page,
                      you agree to the terms and conditions
below.
                      </P>
                      </FONT>
                      <font size="-1">
                      <textarea wrap = "virtual" readonly
rows="20" cols="60">
                      <%-- Get the text for Terms and Conditions
--%>
                      <%= dataBeanTandC.getTandC() %>
                      </textarea>
                      </font>
   </td>
   </tr>
   </table>
</td>
</tr>
</table>
</form>
         </TD>
         </TR>
</TABLE>
                 <BR>
         </TD>
         <TD width="3%">&nbsp;</td>
         </TR>
         </TABLE>
         <!-- end of the inner nested table-->
         <TABLE BORDER="0" CELLSPACING="0" CELLPADDING="0"
HEIGHT="33" WIDTH="100%">
         <TR>
         <TD CLASS="pagetitle" ALIGN="left" VALIGN="bottom"
WIDTH="1%"><IMG HEIGHT="15"
SRC="images/curve_bottom_left.gif"></TD>
         <TD CLASS="pagetitle" ALIGN="left" VALIGN="middle">
                 <TABLE BORDER="0" CELLSPACING="0"
CELLPADDING="0" HEIGHT="19">
                 <TR>
                 <TD NOWRAP CLASS="dark"><IMG
SRC="images/buttons/b_left.gif" BORDER="0"></TD>
                 <TD NOWRAP CLASS="dark"><A CLASS="buttonText"
HREF="<%= bean.getNavigation(ATTR_BIDSCREENBEAN_PREV_LINK) %>"><
Return to Previous</A></TD>
```

```
                 <TD NOWRAP CLASS="dark"><IMG
SRC="images/buttons/b_right.gif" BORDER="0"></TD>
                 </TR>
                 </TABLE>
         /TD>
         <TD CLASS="pagetitle" ALIGN="right" VALIGN="bottom"
WIDTH="1%"><IMG HEIGHT="15" WIDTH="17"
SRC="images/curve_bottom_right.gif"></TD>
         /TR>
   </TABLE>
</TD>
</TR>
<TR><TD NOWRAP WIDTH="760" BGCOLOR="#FFFFFF"><!--this row is to
force the minimum table width to 760-->&nbsp;</TD></TR>
</TABLE>
<%= bean.getButtonInfo() %>
</FORM>
```

Figure 4-2 shows part of the Terms and Conditions page.

Figure 4-2 Terms and Conditions Page


Click here to return to the page shown in Figure 4-1.
Auctions : Terms and Conditions
Return to Previous
Terms and Conditions

Terms and Conditions:
Product Name

By viewing this page, you agree to the terms and conditions below.

Chapter 5

Customizing the OMS Module

This chapter contains the following sections:

- Overview
- Configuring OMS
- Configuring the ECXpert System
- Configuring a New Business Event Checklist

Overview

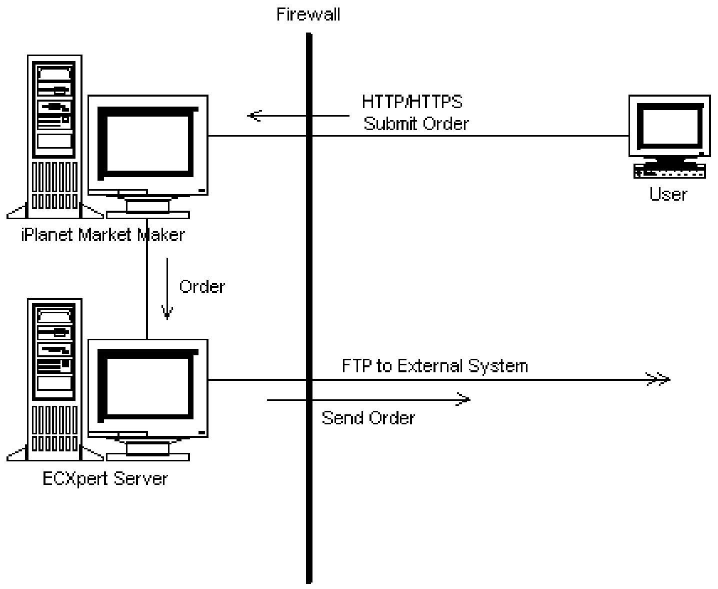
To process financial transactions in a marketplace, companies often need to transfer information about the transactions to their external ERP systems. To facilitate this type of transfer, a method is required for the marketplace to communicate with the ERP systems.

Within an iPlanet Market Maker marketplace, the Order Management System (OMS) module lets you transfer requisitions and orders in the form of text documents from OMS to external systems. To transfer documents to external systems, the iPlanet Marker Maker software communicates with the iPlanet ECXpert system, which controls the routing of documents to external systems. The iPlanet ECXpert system lets you automate and manage the communications processes between the marketplace and ERP systems. See Figure 5-1.

For more information about the iPlanet ECXpert system, see the following URL.

`http://www.iplanet.com/products/iplanet_ecxpert/home_2_1_1q.html`

Figure 5-1 Sending a Purchase Order to an External System via the ECXpert System



This chapter describes how to configure and customize OMS and the ECXpert system to enable the transfer of documents to external systems. It contains the following topics.

- Configuring OMS.
    - Editing the ESVConfiguration.properties file.
    - Adding a Document Encoding Type.
    - Adding a Business Event.
- Configuring the ECXpert System.
- Configuring a New Business Event Checklist.
    - Checking OMS and ECXpert system configuration requirements.

For more information about how to use OMS, see the iPlanet Market Maker online help.

Configuring OMS

Editing the ESVConfiguration.properties file

The OMS configuration settings shown in Table 5-1 are required in order to communicate with the ECXpert system. Specify these settings in the `ESVConfiguration.properties` file in the `<imm_root>/iMM/resources` directory. Note that after you specify these settings, you must restart the iPlanet Market Maker software for the settings to take effect.

Table 5-1 OMS Configuration Entries

Configuration Key	**Description**
`CFG_ECX_SENDER_MEMBER_NAME`	Member name of the sending member of ECXpert document submissions. This name becomes part of the sender email address and should match the email remote (ER) attribute of the corresponding membership in the ECXpert system.
`CFG_ECX_SENDER_MEMBER_PASSWORD`	ECXpert member password for the sending member for ECXpert document submissions. This password is not required if the submit mechanism for the ECXpert system is SMTP.
`CFG_ECX_ADAPTER_SMTP_SERVER_HOST`	SMTP server to be used by the ECXpert SMTP submit handler to submit documents to the ECXpert system. This server name is used when the ECXpert submit mechanism is set to SMTP. The email address of the sender becomes <*sender_ecx_member_name*>@<*this_server_name*> and the email address of the receiver becomes <*receiver_ecx_member_name*>@<*this_server_name*>

Adding a Document Encoding Type

To transfer documents to external systems, OMS encodes the documents. Document encoding has two parts: *envelope* encoding for identifying, representing, and routing the document and *payload* encoding for the business content. The iPlanet Market Maker external services support the following XML encoding standards.

- Envelope encoding - ebXML, BizTalk, SOAP, CXML, and native (iPlanet Market Maker)

- Payload encoding - xCBL, OBI-XML, CXML, and native (iPlanet Market Maker)

Note that the OBI-XML and CXML formats might require customization to work properly. Contact iPlanet Market Maker professional services if you need help using these encoding standards.

The example in this section shows you how to add a new envelope encoding type, XBRL (eXtensible Business Reporting Language), to transform all the purchase orders submitted to vendors into external documents and transport them to an external system that generates financial statements.

Defining Encoding Types

Define an encoding type in the `ESVCONFIG.XML` file in the `<imm_root>/iMM/omsB/etc/xsl` directory by giving it a unique internal persistent code. In Table 5-2, the new encoding type to add is highlighted.

Table 5-2 Defining Encoding Types in the `ESVCONFIG.XML` File

```
/*
*   DOCUMENT ENVELOPE ENCODINGS
*/
ENVELOPE_IMM = 0
ENVELOPE_EBXML = 1
ENVELOPE_OBIXML = 2
ENVELOPE_SOAP = 3
ENVELOPE_EDI = 4
ENVELOPE_XBRL = 5
```

Defining Document Encoding Types

A document encoding type is a combination of envelope encoding and payload encoding. To define the document encoding types in the `ESVCONFIG.XML` file, combine the new envelope encoding type with one or more payload encoding types as highlighted in Table 5-3.

Table 5-3 Document Encoding Type Mapping in the `ESVCONFIG.XML` File

```
/*
    *   ESV SUPPORTED DOCUMENT ENCODINGS
    * For these encoding combinations ESV will do the mapping
    */
    private static DocumentEncodingConfigurationEntry ENC_IMM_IMMPO =
        new DocumentEncodingConfigurationEntry(ENVELOPE_IMM, PAYLOAD_IMM_PO);
    private static DocumentEncodingConfigurationEntry ENC_IMM_XCBLPO =
```

Table 5-3 Document Encoding Type Mapping in the `ESVCONFIG.XML` File *(Continued)*

```
        new DocumentEncodingConfigurationEntry(ENVELOPE_IMM, PAYLOAD_XCBL_PO);
    private static DocumentEncodingConfigurationEntry ENC_EBXML_IMMPO =
        new DocumentEncodingConfigurationEntry(ENVELOPE_EBXML,
PAYLOAD_IMM_PO);
    private static DocumentEncodingConfigurationEntry ENC_EBXML_XCBLPO =
        new DocumentEncodingConfigurationEntry(ENVELOPE_EBXML,
PAYLOAD_XCBL_PO);
    private static DocumentEncodingConfigurationEntry ENC_IMM_IMMREQ =
        new DocumentEncodingConfigurationEntry(ENVELOPE_IMM, PAYLOAD_IMM_REQ);
    private static DocumentEncodingConfigurationEntry ENC_EBXML_IMMREQ =
        new DocumentEncodingConfigurationEntry(ENVELOPE_EBXML,
PAYLOAD_IMM_REQ);
    private static DocumentEncodingConfigurationEntry ENC_OBIXML_OBIXMLREQ =
        new DocumentEncodingConfigurationEntry(ENVELOPE_OBIXML,
PAYLOAD_OBIXML_REQ);
    private static DocumentEncodingConfigurationEntry ENC_XBRL_IMMPO =
        new DocumentEncodingConfigurationEntry(ENVELOPE_XBRL, PAYLOAD_IMM_PO);
```

Registering Encoding Types

.

Table 5-4 Registering Encoding Types

```
<ListOfEnvelope>..</ListOfEnvelope> from <mango
dir>/esvB/etc/xsl/ESVCONFIG.XML
```

Registering a New Document Encoding Type

In the `ListOfDocumentEncodingConfigurationEntry` method in the `ESVCONFIG.XML` file, add the new document encoding type as highlighted in Table 5-5.

Table 5-5 Registering a New Document Encoding Type in the `ESVCONFIG.XML` File

```
<ListOfDocumentEncodingConfigurationEntry>
        <DocumentEncodingConfigurationEntry envelopeEncoding="0"
payloadEncoding="0"/>
        <DocumentEncodingConfigurationEntry envelopeEncoding="0"
payloadEncoding="2"/>
```

Table 5-5 Registering a New Document Encoding Type in the ESVCONFIG.XML File

```
        <DocumentEncodingConfigurationEntry envelopeEncoding="1"
payloadEncoding="0"/>
        <DocumentEncodingConfigurationEntry envelopeEncoding="1"
payloadEncoding="2"/>
        <DocumentEncodingConfigurationEntry envelopeEncoding="5"
payloadEncoding="0"/>
```

Defining the Document Type Mapping

A SUBMIT_TO_VENDOR action already exists in OMS. A business event OMS_SUBMIT_TO_VENDOR is defined in the ESVCONFIG.XML file. Under the submit to vendor cases section of this class, map the combination of encoding types and business event OMS_SUBMIT_TO_VENDOR to a document type xbrl-imm-po as shown in Table 5-6.

Table 5-6 Document Type Mapping in the ESVCONFIG.XML File

```
private static DocumentExternalTypeConfigurationEntry
        XTYPE_OMS_SUBMIT_TO_VENDOR_XBRL_IMM =
            new DocumentExternalTypeConfigurationEntry(
                INT_SYS_OMS, INT_ACT_SUBMIT_TO_VENDOR,
                ENVELOPE_XBRL, PAYLOAD_IMM_PO,
                "xbrl-imm-po");
```

Registering a New Document Type

In the ListOfDocumentExternalTypeConfigurationEntry method in the ESVCONFIG.XML file, add the new document type as highlighted in Table 5-7.

Table 5-7 Registering a New Document Type in the ESVCONFIG.XML File

```
<ListOfDocumentExternalTypeConfigurationEntry>
        <DocumentExternalTypeConfigurationEntry system="0" action="1"
envelope="0" payload="1" externalType="imm-imm-req"/>
        <DocumentExternalTypeConfigurationEntry system="0" action="1"
envelope="1" payload="1" externalType="ebxml-imm-req"/>
        <DocumentExternalTypeConfigurationEntry system="0" action="1"
envelope="2" payload="3" externalType="obixml-obixml-req"/>
        <DocumentExternalTypeConfigurationEntry system="0" action="0"
envelope="0" payload="0" externalType="imm-imm-po"/>
        <DocumentExternalTypeConfigurationEntry system="0" action="0"
envelope="0" payload="2" externalType="imm-xcbl-po"/>
```

Table 5-7 Registering a New Document Type in the `ESVCONFIG.XML` File *(Continued)*

```
<ListOfDocumentExternalTypeConfigurationEntry>
        <DocumentExternalTypeConfigurationEntry system="0" action="0"
envelope="0" payload="4" externalType="imm-cxml-po"/>
        <DocumentExternalTypeConfigurationEntry system="0" action="0"
envelope="1" payload="0" externalType="ebxml-imm-po"/>
        <DocumentExternalTypeConfigurationEntry system="0" action="0"
envelope="1" payload="2" externalType="ebxml-xcbl-po"/>
        <DocumentExternalTypeConfigurationEntry system="0" action="0"
envelope="1" payload="4" externalType="ebxml-cxml-po"/>
        <DocumentExternalTypeConfigurationEntry system="0" action="2"
envelope="0" payload="0" externalType="imm-imm-po"/>
        <DocumentExternalTypeConfigurationEntry system="0" action="2"
envelope="0" payload="2" externalType="imm-xcbl-po"/>
        <DocumentExternalTypeConfigurationEntry system="0" action="2"
envelope="0" payload="4" externalType="imm-cxml-po"/>
        <DocumentExternalTypeConfigurationEntry system="0" action="2"
envelope="1" payload="0" externalType="ebxml-imm-po"/>
        <DocumentExternalTypeConfigurationEntry system="0" action="2"
envelope="1" payload="2" externalType="ebxml-xcbl-po"/>
        <DocumentExternalTypeConfigurationEntry system="0" action="2"
envelope="1" payload="4" externalType="ebxml-cxml-po"/>
        <DocumentExternalTypeConfigurationEntry system="0" action="0"
envelope="5" payload="0" externalType="xbrl-imm-po"/>
```

Editing XSL Style Sheets

Every encoding type has an XSL style sheet. The style sheet provides the encoding instructions for XSL processing, which does the encoding. There are some examples of style sheets in the `xsl` directory.

The XSL for payload encoding appears under specific modules like <*imm dir*>`/omsB/etc/xsl`, <*imm dir*>`/rfxB/etc/xsl`.

The XSL for envelop encoding is under <*imm dir*>`/esvB/etc/xsl`

Create an XSL style sheet for the your new type and define it in the `ESVCONFIG.XML` file (under `ENVELOPE OUTGOING XSL MAPPINGS`). Map the XSL style sheet with the new envelope encoding type as highlighted in Table 5-8.

Table 5-8 XSL Mappings in the `ESVCONFIG.XML` File

```
/*
    *   ENVELOPE OUTGOING XSL MAPPINGS
    *
    */
    private static DocumentMappingConfigurationEntry MAP_ENVELOPE_IMM =
        new DocumentMappingConfigurationEntry(ENVELOPE_IMM, "immenv.xsl");
    private static DocumentMappingConfigurationEntry MAP_ENVELOPE_EBXML =
        new DocumentMappingConfigurationEntry(ENVELOPE_EBXML, "ebxmlenv.xsl");
    private static DocumentMappingConfigurationEntry MAP_ENVELOPE_SOAP =
        new DocumentMappingConfigurationEntry(ENVELOPE_SOAP, "soapenv.xsl");
    private static DocumentMappingConfigurationEntry MAP_ENVELOPE_OBIXML =
        new DocumentMappingConfigurationEntry(ENVELOPE_OBIXML,
"obixmlenv.xsl");
    private static DocumentMappingConfigurationEntry MAP_ENVELOPE_XBRL =
        new DocumentMappingConfigurationEntry(ENVELOPE_XBRL, "xbrlenv.xsl");
```

Registering a New Envelope Encoding Type

In the `ListOfEnvelopeDocumentMappingConfigurationEntry` method in the `ESVCONFIG.XML` file, add the new envelope encoding type as highlighted in Table 5-9.

Table 5-9 Registering a New Envelope Encoding Type in the `ESVCONFIG.XML` File

```
<ListOfEnvelopeDocumentMappingConfigurationEntry>
        <DocumentMappingConfigurationEntry encodingType="0"
xslName="immenv.xsl"/>
        <DocumentMappingConfigurationEntry encodingType="1"
xslName="ebxmlenv.xsl"/>
        <DocumentMappingConfigurationEntry encodingType="2"
xslName="obixmlenv.xsl"/>
        <DocumentMappingConfigurationEntry encodingType="3"
xslName="soapenv.xsl"/>
        <DocumentMappingConfigurationEntry encodingType="5"
xslName="xbrlenv.xsl"/>
</ListOfEnvelopeDocumentMappingConfigurationEntry>
```

Defining the Encoding Property

In the `OMSstrings.properties` file in the `imm_root>/iMM/resources` directory, add the new envelope encoding type to the list as highlighted in Table 5-10.

Table 5-10 Defining the Encoding Property in the `OMSstrings.properties` File

```
#ESV_ENVELOPE_ENCODING_TYPE_NAMES
ESV_ENVELOPE_ENCODING_TYPE_NAMES[0]=IMM
ESV_ENVELOPE_ENCODING_TYPE_NAMES[1]=ebXML
ESV_ENVELOPE_ENCODING_TYPE_NAMES[2]=OBIXML
ESV_ENVELOPE_ENCODING_TYPE_NAMES[3]=SOAP
ESV_ENVELOPE_ENCODING_TYPE_NAMES[4]=EDI
ESV_ENVELOPE_ENCODING_TYPE_NAMES[5]=XRBL
```

The property files in <*ias_dir*>`/imm40/imm40/WEB-INF/classes` should also be changed.

The numeric index you use must match the `int` value in the static constant defined for this encoding type in the `ESVCONFIG.XML` file, in this case five.

Adding a Business Event

This section describes the current OMS external services implementation for submitting a business event from an iPlanet Market Maker marketplace to a remote vendor. A business event refers to an action taken on behalf of an iPlanet Market Maker marketplace. You can use a similar implementation to create your own business events to submit to external systems.

Defining a Module Constant

Each business event has a system and an action. OMS is currently the only iPlanet Market Maker module that supports external services. The system and the actions are defined as static constants with a `string` and an `int` in the `IESVModuleConstants` public interface class in the `external` directory. The string is used to specify an event type to call in the document handler (for example, see the middle column of Table 5-15). The `int` is the unique internal persistent code for the action.

The following code in `IESVModuleConstants` adds the action `SUBMIT_TO_VENDOR` to the list of supported actions.

```
public static final String STR_ACT_SUBMIT_TO_VENDOR =
"SUBMIT_TO_VENDOR";

    public static final int INT_ACT_SUBMIT_TO_VENDOR = 0;
//SUBMIT_TO_VENDOR
```

Initiating External Services for an Action

Every action defined as a business event in the external services must be initiated in the respective method in the remotable class of the OMS module. The `submitToVendor` method defines the action as an external business event. See Table 5-11. Note that "..." in the tables indicates that some syntax is omitted.

Table 5-11 `submitToVendor` Method

```
public Order submitToVendor(Order order, Context context)
...
handler = getOutgoingDocumentHandler();
            if(handler != null)
           {
                ExternalBusinessEvent event =
                    new ExternalBusinessEvent(INT_SYS_OMS,
INT_ACT_SUBMIT_TO_VENDOR);
                handler.processOutgoingEvent(order, event,
context);
           }
```

For information about implementing the code shown in Table 5-11, contact iPlanet Market Maker professional services.

Defining an Action in External Services

To define an action as a business event in the external services, you need to define and register the action in the `ESVCONFIG.XML` file and associate the action with the OMS system.

To define an action, map the `int` representation of the action to its `string` representation as commented in `ESVCONFIG.XML` as follows:

```
private static final ExternalServicesConfigurationEntry
ACT_SUBMIT_TO_VENDOR =
new ExternalServicesConfigurationEntry(INT_ACT_SUBMIT_TO_VENDOR,
STR_ACT_SUBMIT_TO_VENDOR);
```

Defining an OMS-Supported Event

In the `initializeAllSupportedActionsMap` method, add the business event as an OMS-supported event. See the example highlighted in Table 5-12.

Table 5-12 Defining an OMS-Supported Event

```
<ListOfAction>
    <ExternalServicesConfigurationEntry actionInt="0"
actionString="SUBMIT_TO_VENDOR"/>
    <ExternalServicesConfigurationEntry actionInt="1"
actionString="NEW_BUYERX_REQ"/>
    <ExternalServicesConfigurationEntry actionInt="2"
actionString="ORDER_CANCEL"/>
    <ExternalServicesConfigurationEntry actionInt="3"
actionString="ORDER_APPROVE"/>
</ListOfAction>
```

Registering a Business Event

Also in the `ESVCONFIG.XML` file, add the business event in the `loadSupportedActionsMap` method. See Table 5-13. Note that the code to add is highlighted in Table 5-13 and Table 5-12.

Table 5-13 Adding a Business Event

```
<ListOfExternalBusinessEvent>
    <ExternalBusinessEvent system="0" action="0"/>
    <ExternalBusinessEvent system="0" action="1" systemEvent="true"/>
```

Table 5-13 Adding a Business Event *(Continued)*

```
    <ExternalBusinessEvent system="0" action="2"/>
    <ExternalBusinessEvent system="0" action="3"/>
</ListOfExternalBusinessEvent>
```

Defining Document Encoding Types for the Event

Document encoding types are combinations of envelope and payload encoding types defined in the `ESVCONFIG.XML` file. For a new business event, define the document types by mapping an action and the document encoding types to document types. See Table 5-14.

Table 5-14 Document and Encoding Types

```
//order submit to vendor cases
    private static DocumentExternalTypeConfigurationEntry
        XTYPE_OMS_SUBMIT_TO_VENDOR_IMM_IMM =
            new DocumentExternalTypeConfigurationEntry(
                INT_SYS_OMS, INT_ACT_SUBMIT_TO_VENDOR,
                ENVELOPE_IMM, PAYLOAD_IMM_PO,
                "imm-imm-po");
    private static DocumentExternalTypeConfigurationEntry
        XTYPE_OMS_SUBMIT_TO_VENDOR_IMM_XCBL =
            new DocumentExternalTypeConfigurationEntry(
                INT_SYS_OMS, INT_ACT_SUBMIT_TO_VENDOR,
                ENVELOPE_IMM, PAYLOAD_XCBL_PO,
                "imm-xcbl-po");
    private static DocumentExternalTypeConfigurationEntry
        XTYPE_OMS_SUBMIT_TO_VENDOR_IMM_CXML =
            new DocumentExternalTypeConfigurationEntry(
                INT_SYS_OMS, INT_ACT_SUBMIT_TO_VENDOR,
                ENVELOPE_IMM, PAYLOAD_CXML_PO,
                "imm-cxml-po");
    private static DocumentExternalTypeConfigurationEntry
        XTYPE_OMS_SUBMIT_TO_VENDOR_EBXML_IMM =
            new DocumentExternalTypeConfigurationEntry(
                INT_SYS_OMS, INT_ACT_SUBMIT_TO_VENDOR,
                ENVELOPE_EBXML, PAYLOAD_IMM_PO,
                "ebxml-imm-po");
    private static DocumentExternalTypeConfigurationEntry
        XTYPE_OMS_SUBMIT_TO_VENDOR_EBXML_XCBL =
            new DocumentExternalTypeConfigurationEntry(
                INT_SYS_OMS, INT_ACT_SUBMIT_TO_VENDOR,
                ENVELOPE_EBXML, PAYLOAD_XCBL_PO,
                "ebxml-xcbl-po");
    private static DocumentExternalTypeConfigurationEntry
        XTYPE_OMS_SUBMIT_TO_VENDOR_EBXML_CXML =
            new DocumentExternalTypeConfigurationEntry(
```

Table 5-14 Document and Encoding Types *(Continued)*

`INT_SYS_OMS, INT_ACT_SUBMIT_TO_VENDOR,`
`ENVELOPE_EBXML, PAYLOAD_CXML_PO,`
`"ebxml-cxml-po");`

Registering Document Types for an Event

Register the document types defined for a new business event in the `ESVCONFIG.XML` file. See Table 5-14.

Defining the User Interface Property

In the `OMSstrings.properties` property file, add the definition for a new action. This following definition provides the localizable name for the action that appears in the user interface. The numeric index used here must match the `int` value in the static constant defined for this action in the `IESVModuleConstants` (in this case zero).

```
#ESV_ACTION_NAMES

ESV_ACTION_NAMES[0]=SUBMIT_TO_VENDOR

...
```

Configuring the ECXpert System

OMS uses an SMTP-based email mechanism to submit XML documents to the ECXpert system. To use the SMTP-based submission mechanism, you need to configure your ECXpert system as described in the following steps.

1. Configuring SMTP receive service in the ECXpert system.
2. Adding the iPlanet Market Maker marketplace membership in the ECXpert system.
3. Adding membership for the receiving company.
4. Setting up the partnership.
5. Setting up the service list.

Configuring SMTP Receive Service in the ECXpert System

To send XML documents from an iPlanet Market Maker marketplace to an ECXpert system, you must configure the ECXpert SMTP receive service. The ECXpert SMTP receive service needs to poll a specified inbox that receives all the emails from the marketplace.

The inbox that receives all email messages from the marketplace can either be a POP3 inbox on a mail server, or the SendMail inbox of a user on the local system. Based on the email solution you choose, you must provide the corresponding information when you install the ECXpert system. Figure 5-2 shows an example of the information you need to provide.

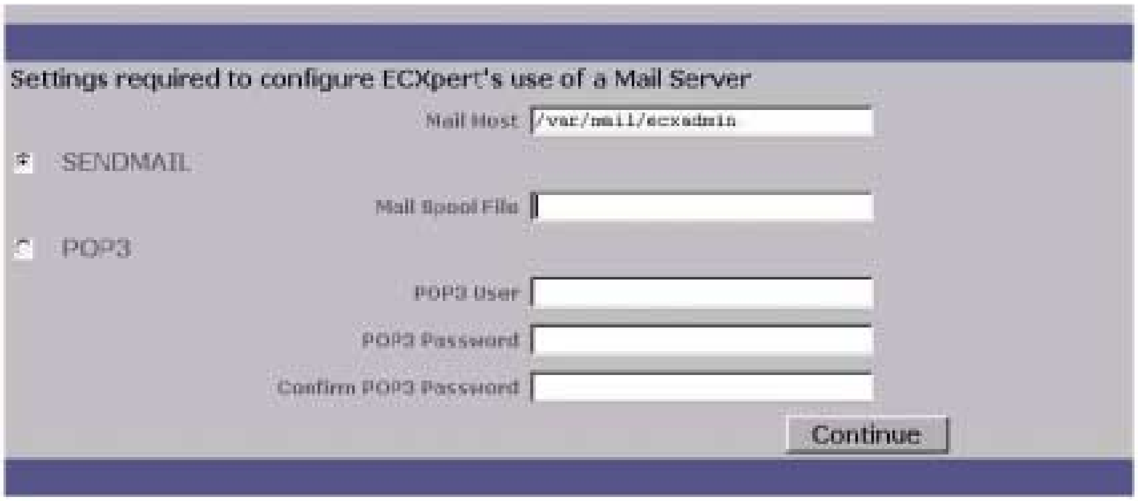


Figure 5-2 Configuring a Mail Server in the ECXpert System

Settings required to configure ECXpert's use of a Mail Server

[✓ SENDMAIL]

Mail Host	/var/mail/ecxadmin
Mail Spool File	

POP3

POP3 User	
POP3 Password	
Confirm POP3 Password	

It is important to note that by default the SMTP receive service in a newly installed ECXpert system is not started automatically. You must manually start this service for the ECXpert system to interact with the mail server. You can modify certain other settings for the SMTP receive service to tune it to handle the expected load on the system.

Adding an iPlanet Market Maker Marketplace Membership in the ECXpert System

After you install the ECXpert system, the next step is to add the iPlanet Market Maker membership to the ECXpert membership database. This membership helps the ECXpert system identify the origin of documents submitted from the marketplace. When a document is submitted to the ECXpert system through SMTP, the ECXpert system matches the sender’s email address with the various ER values of existing ECXpert memberships. When it finds a match, the ECXpert system identifies the sender of the document as the corresponding ECXpert member. This identification lets you create partnerships and service lists to process business events originating from the iPlanet Market Maker marketplace.

The sender's email address is derived from the following two configuration settings (see Table 5-1).

`CFG_ECX_SENDER_MEMBER_NAME`

`CFG_ECX_ADAPTER_SMTP_SERVER_HOST`

For example, if the sender member name is `immhost` and the ECX SMTP server host is `bluewater.red.iplanet.com`, then the sender's email address becomes `immhost@bluewater.red.iplanet.com`.

Adding Membership for a Receiving Company

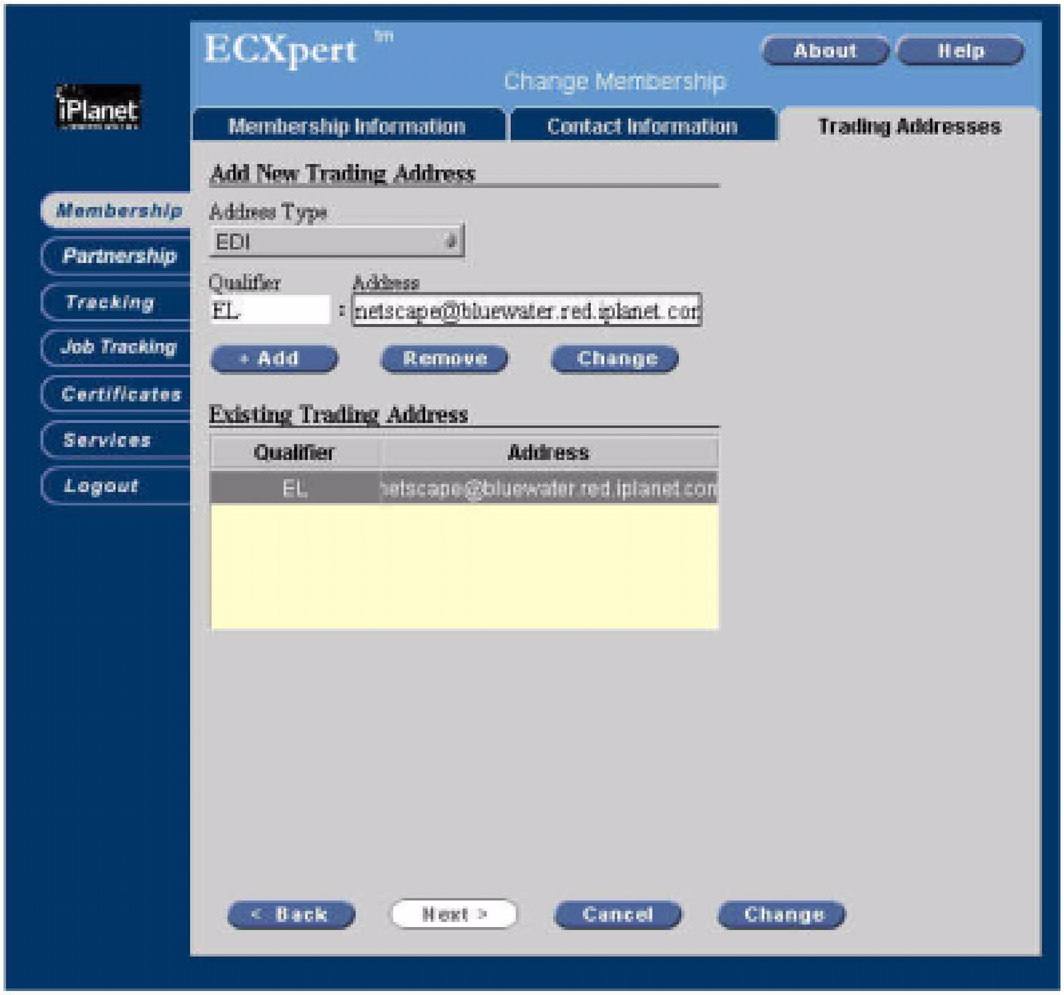
After you set up the mail server and the membership for the iPlanet Market Maker marketplace in the ECXpert system, the next step is to add the membership for a receiving company. This membership should be unique for each receiving company.

Because the iPlanet Market Maker software uses SMTP-based email for submitting documents to the ECXpert system, you must assign an inbox to the new receiving company on the mail server. For SendMail, you must create a local UNIX account for the receiving company. After an inbox is assigned to the new company, the iPlanet Market Maker software is able to send documents via email to this inbox. However, this inbox is not the one polled by the ECXpert system for incoming email submissions. You must set up a forwarding mechanism to the inbox polled by the ECXpert system for incoming emails sent to the new company inbox. For SendMail, a you can use a `.forward` file to forward the incoming email to the ECXpert inbox. For a POP3 server, refer to the server administration manual to set up a forwarding service to another inbox.

The reason for having a separate inbox for each company, and forwarding the incoming email to the ECXpert inbox, is to preserve the original SMTP headers and provide sufficient information to the ECXpert system to determine the intended receiving member. For example, email sent to `iplanet@bluewater.red.iplanet.com,` which is forwarded to `actraadm@bluewater.red.iplanet.com`, contains the necessary header information that allows the ECXpert system to determine that the mail was originally sent to `iplanet@bluewater.red.iplanet.com`.

Based on the original email address where the email was sent, the ECXpert system can look up its membership database and identify the corresponding receiver's membership entry. For every company that receives documents through this system, you must create a membership in the ECXpert system that has a Local Email (EL) attribute value set to the email address of its corresponding inbox. See Figure 5-3.

Figure 5-3 Adding a Local Email Address



Add New Trading Address

Address Type: EDI

Qualifier: EL

Address: netscape@bluewater.red.iplanet.com

Existing Trading Address

Qualifier	Address
EL	netscape@bluewater.red.iplanet.com

When a business event occurs that requires a document to be submitted to the ECXpert system, the receiving email address is derived from the following.

- The receiving company's ECXpert member name, which is determined by the details of the active registry entry associated with the business event.
- The SMTP server host, as defined in the configuration file.

For example, if the active registry entry contains the name `iplanet` as the ECXpert member name, and the SMTP server host is set to `bluewater.red.iplanet.com` based on the configuration settings, the document is emailed to the address `iplanet@bluewater.red.iplanet.com`. The email addresses should all exist on the same domain name, which is the domain name of the SMTP server. It is not possible for a receiving company to have an email address at a different domain other than the domain of the SMTP server used to submit documents.

With sufficient care in configuring the email accounts for companies participating in this system, you can resolve any possible collisions. The advantage of this convention is that it is internal and does not have dependencies with the external email addresses. It allows the administrator to choose any name for the company's internal email and ECXpert membership. Even though it is not a requirement for the internal email name for a company to be the same as the ECXpert member name, it helps to keep the configuration simple and easy to understand.

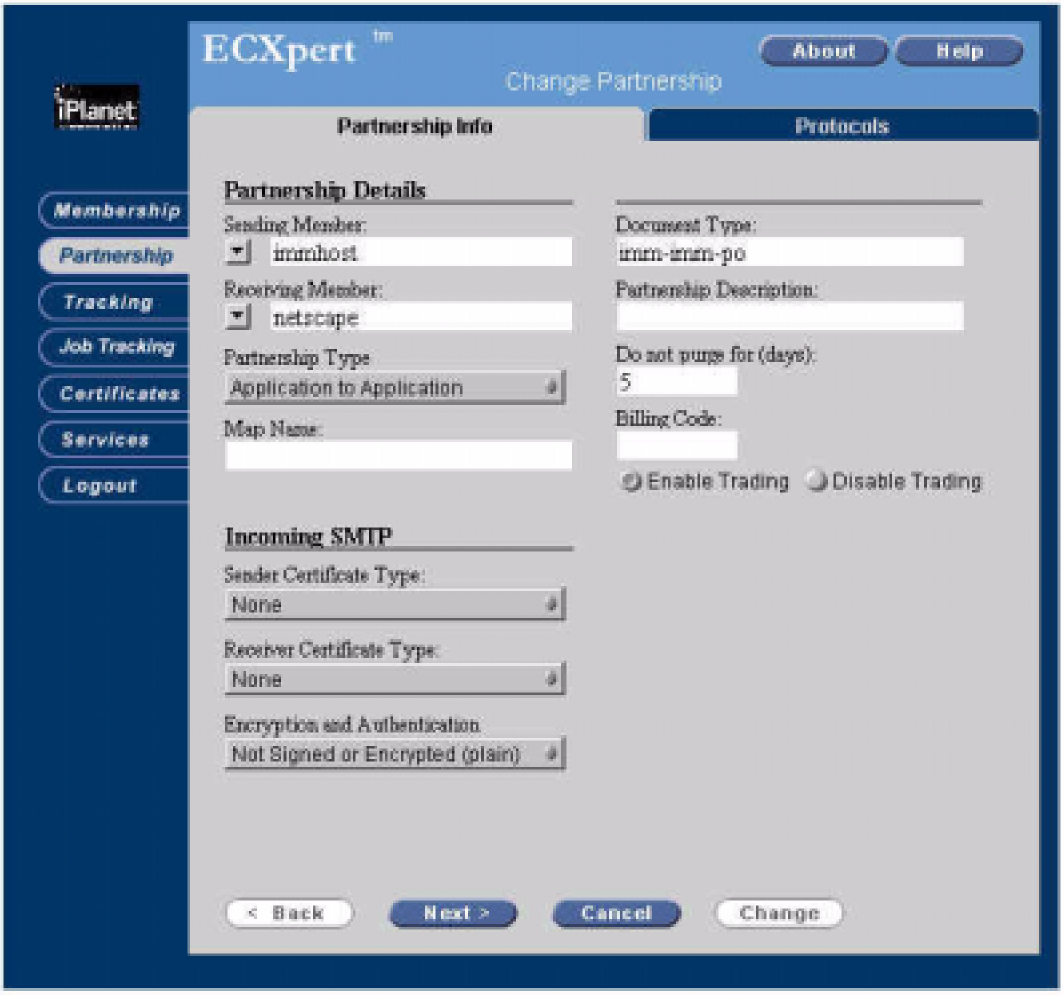
Setting up the Partnership

The ECXpert partnership determines the outbound delivery mechanism. The partnership also determines any ECXpert-based system translations for the document sent by the iPlanet Market Maker marketplace to the receiving company's inbox. The partnership is based on the following criteria:

- The sending member name.
- The receiving member name.
- The document data type.

The sending member name is described in the "Adding an iPlanet Market Maker Marketplace Membership in the ECXpert Systems" section. The receiving member name is described above in the "Adding Membership for a Receiving Company" section. The document type is determined at runtime based on the type of document encoding scheme and the event type specified by the corresponding business event. See Figure 5-4.

Figure 5-4 Setting Partnership Information



Partnership Details

Sending Member: immhost

Receiving Member: netscape

Partnership Type: Application to Application

Map Name:

Document Type: imm-imm-po

Partnership Description:

Do not purge for (days): 5

Billing Code:

Enable Trading Disable Trading

Incoming SMTP

Sender Certificate Type: None

Receiver Certificate Type: None

Encryption and Authentication: Not Signed or Encrypted (plain)

Table 5-15 lists the document data types for the encoding schemes you can use for setting partnerships.

Table 5-15 Document Data Types for OMS

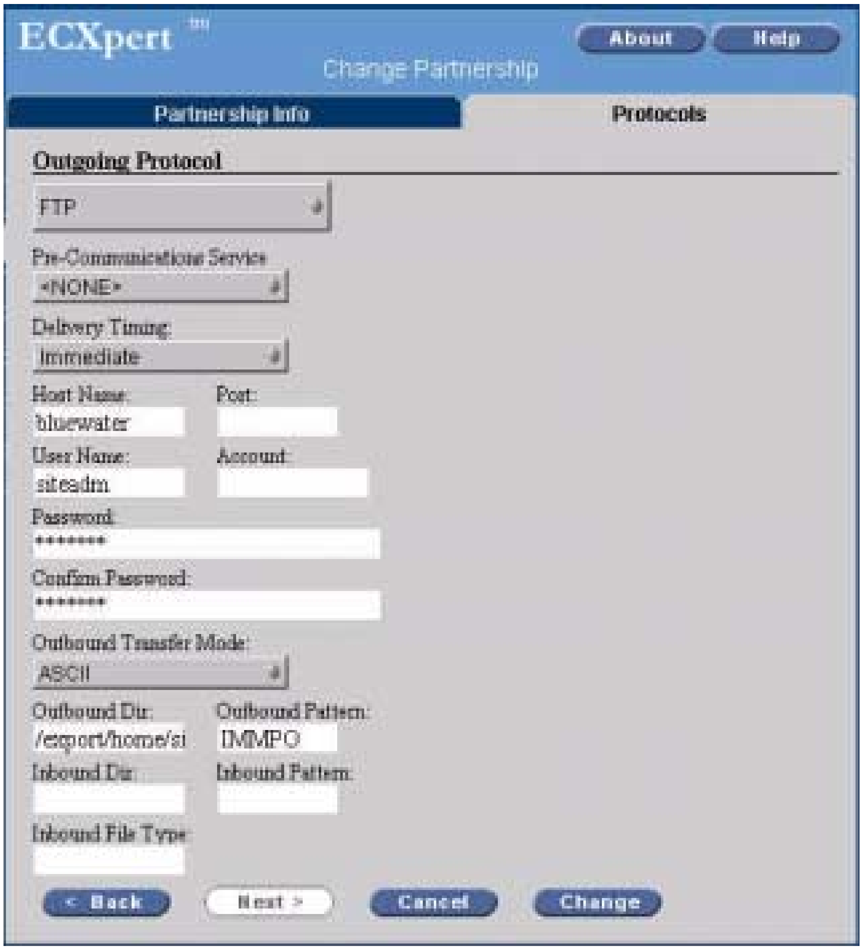
Envelope and Payload Encoding	Event Type	Document Data Type
iMM Envelope, iMM Payload	System: OMS Action: SUBMIT_TO_VENDOR	imm-imm-po

Table 5-15 Document Data Types for OMS *(Continued)*

iMM Envelope, iMM Payload	System: OMS Action: NEW_BUYERX_REQ	imm-imm-req
ebXML Envelope, iMM Payload	System: OMS Action: NEW_BUYERX_REQ	ebxml-imm-req
OBIXML Envelope, OBIXML Payload	System: OMS Action: NEW_BUYERX_REQ	obixml-obixml-req
iMM Envelope, xCBL Payload	System: OMS Action: SUBMIT_TO_VENDOR	imm-xcbl-po
iMM Envelope, cXML Payload	System: OMS Action: SUBMIT_TO_VENDOR	imm-cxml-po
ebXML Envelope, iMM Payload	System: OMS Action: SUBMIT_TO_VENDOR	ebxml-imm-po
ebXML Envelope, xCBL Payload	System: OMS Action: SUBMIT_TO_VENDOR	ebxml-xcbl-po
ebXML Envelope, cXML Payload	System: OMS Action: SUBMIT_TO_VENDOR	ebxml-cxml-po

After you define the partnership information, the next step is to define the outgoing protocol. Figure 5-5 shows an example that uses FTP protocol. Note `Outbound Dir` is the directory that contains the document received from the OMS module.

Figure 5-5 Setting Protocol Information



Outgoing Protocol

FTP

Pre-Communications Service: <NONE>

Delivery Timing: Immediate

Host Name: bluewater

Port:

User Name: siteadm

Account:

Password: *******

Confirm Password: *******

Outbound Transfer Mode: ASCII

Outbound Dir: /export/home/si

Outbound Pattern: IMMPO

Inbound Dir:

Inbound Pattern:

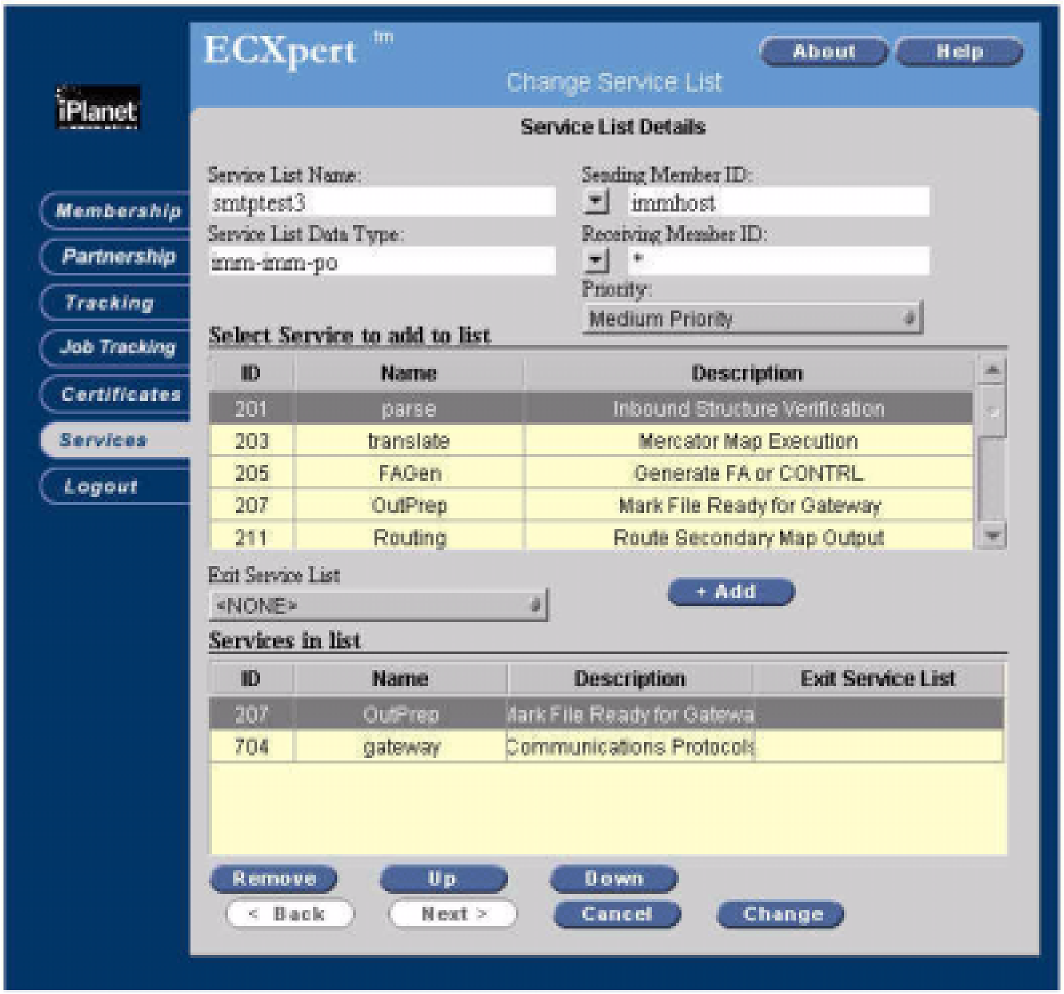
Inbound File Type:

Setting Up the Service List

A service list identifies the services that apply to the submitted document before it is delivered to the destination. The Gateway Service delivers the document, and without it the document is not delivered to the intended destination. Like the partnership, the service list is also dependent on the sending member name, the receiving member name, and the document data type.

It is also possible to have generic service lists that are applicable to more than one set of sending member, receiving member, and document data type. When the ECXpert system is being used only for document delivery, the service list should have the Outprep and Gateway services only. See Figure 5-6. For further information about ECXpert, see the ECXpert Administration Manual.

Figure 5-6 Setting Service List Information



Change Service List

Service List Details

Service List Name: smtptest3

Sending Member ID: immhost

Service List Data Type: imm-imm-po

Receiving Member ID: *

Priority: Medium Priority

Select Service to add to list

ID	Name	Description
201	parse	Inbound Structure Verification
203	translate	Mercator Map Execution
205	FAGen	Generate FA or CONTRL
207	OutPrep	Mark File Ready for Gateway
211	Routing	Route Secondary Map Output

Exit Service List: <NONE>

Services in list

ID	Name	Description	Exit Service List
207	OutPrep	lark File Ready for Gatewa	
704	gateway	Communications Protocols	

Configuring a New Business Event Checklist

This section provides a checklist for configuring a new business event. This list is divided into two parts. The first part is a one-time configuration for the entire ECXpert system and iPlanet Market Maker software. The second part is the configuration required for any new business event.

One-time System Setup

- Configure the Mail Server or SendMail, whichever is used with the ECXpert system.
- Install the ECXpert system and configure it to poll the ECXpert inbox on the Mail Server.
- Start the SMTP Receive Service on the ECXpert system through the administration screen.
- Modify the `ESVConfiguration.properties` file to set the name of SMTP Server to be used for submitting documents to the ECXpert system.
- Create the ECXpert membership for the iPlanet Market Maker marketplace. This membership should have a remote email (ER) attribute equivalent to `<sending member name for configuration file>@<SMTP server name from configuration file>`. For example `immhost@bluewater.red.iplanet.com`.

Configuration Required per Business Event

- Create the business event entry in the iPlanet Market Maker Registry using the ESV Admin screens in the iPlanet Market Maker software.
- Using the ECXpert member name specified for this event, ensure that an inbox exists on the SMTP Server. If this is a new company, create the inbox on the SMTP Server for the company. If an inbox of this name already exists because of previously configured business events, no action is necessary for this step. Note that you can use the same inbox for multiple business events for a given company.
- Ensure that the above-mentioned inbox is configured to forward all incoming emails to the ECXpert inbox. If this inbox is already configured because of previously configured business events, no action is necessary for this step.
- Create an ECXpert membership for the receiving company. This membership should have a local email (EL) attribute equivalent to `<ECXpert member name>@<SMTP server name from configuration file>`. For example `iplanet@bluewater.red.iplanet.com`. If this membership already exists because of previously configured business events, no action is necessary for this step.
- Create the ECXpert partnership. Using the ECXpert membership name of the receiving company, the ECXpert membership name of the iPlanet Market Maker marketplace as the sending company, and the document data type from Table 5-15, create the ECXpert partnership for this business event. The partnership should have the necessary outgoing protocol parameters set up for delivering documents according to the requirements of the receiving company. Optionally you can specify a map name if needed for any translations. Refer to the ECXpert Administration Manual for details about Partnership setup.
- Based on the same sender member name, receiver member name, and document data type, use the partnership information to create a service list that has the Outprep and Gateway services at the minimum. You can add other services to the service list if necessary. Refer to the ECXpert Administration Manual for further details about the available services you can use within a service list.

Chapter 6

Module Pluggability

This chapter contains the following sections:

Definition

Module *pluggability* is defined as the ability to enable or disable one or more optional modules with the following considerations:

- The installer installs all the modules.
- Although a module is typically enabled or disabled post installation, it can be enabled or disabled at any time.
- The modules *Display* should not be disabled.
- The runtime jar files, database tables, and configuration of a disabled module are still in place and can be invoked.
- Enabling or disabling a module results in changes to only the GUI portion of the product at this time, and when a module is disabled, it is removed completely from the GUI. All menu items, href links, buttons, and other objects accessing the module disappear.

Business Application

The out of the box iPlanet Market Maker 4.0 software consists of three core modules:

- Base
- Community
- Display

There are six optional business modules:

- Catalog
- RFx
- Auctions
- Exchange
- Order Management System
- Pricing

Almost every production installation of the iPlanet Market Maker 4.0 software is a subset of the out-of-the-box product.

Depending upon the business application, the customer might need to enable one module and disable another. For example, a marketmaker might host a marketplace using only the exchange module, disabling the RFx and Auctions modules. Another marketmaker might decide to host a vertical marketplace using only the core modules. Though it was possible in earlier versions of the iPlanet Market Maker software to disable one or more modules, it was done largely by modifying the code.

In version 4.0 SP1 of the iPlanet Market Maker software, pluggability is formally supported and is made very simple. A module can be disabled by removing it (commenting it out) from the module list in a property file.

This reduces the effort to create a custom version of the product considerably. Additionally, it results in the following:

- Reduced training.
- Reduced complexity.
- Reduced initial cost to the customer.

Notification is Pluggable

The default option for Notification is *Inbox*. Notification appears in a user's *Inbox* for iPlanet Market Maker events in which the user is interested.

Inbox is part of the main portal screen that appears after login, and it is also available by accessing the Administration tab.

By default, all of the notification modes are disabled. To set the default notification mode, change the following in the VortexConfiguration.properties file:

Set `CFG_DEFAULT_NOTIF` to true and set the `CFG_NOTIFICATION_DEFAULT_MODE` to the mode that you want to enable by default.

Additional modes of notification can be configured by uncommenting config parameters in the `VortexConfiguration.properties` file.

```
VortexConfiguration.properties:

# INBOX is the default.
# Developers can change the impl class and display ID, but we don't recommend
changing the key.
#CFG_NOTIFICATION_INBOX_ATTRS
85[0]=2
85[1]=com.iplanet.ecommerce.vortex.arch.InboxNotifierImpl
85[2]=21

#
# - SMTP
CFG_NOTIFICATION_MODE_1_ID=4
CFG_NOTIFICATION_MODE_1_IMPL=com.iplanet.ecommerce.vortex.arch.SMTPNotifierImpl
CFG_NOTIFICATION_MODE_1_DISPLAY_STR_ID=22

#
# - FAX
#CFG_NOTIFICATION_MODE_2_ID=8                              <== customize
#CFG_NOTIFICATION_MODE_2_IMPL=                           <== customize
#CFG_NOTIFICATION_MODE_2_DISPLAY_STR_ID=23  <== customize
#
# - MEMO Page
#CFG_NOTIFICATION_MODE_3_ID=16                             <== customize
#CFG_NOTIFICATION_MODE_3_IMPL=                            <== customize
#CFG_NOTIFICATION_MODE_3_DISPLAY_STR_ID=24   <== customize
#
# - SMS
#CFG_NOTIFICATION_MODE_4_ID=32                             <== customize
#CFG_NOTIFICATION_MODE_4_IMPL=                            <== customize
#CFG_NOTIFICATION_MODE_4_DISPLAY_STR_ID=25   <== customize
#
# - VOICE
#CFG_NOTIFICATION_MODE_4_ID=64                            <== customize
```

```
VortexConfiguration.properties:

# INBOX is the default.
#CFG_NOTIFICATION_MODE_4_IMPL=                            <== customize
#CFG_NOTIFICATION_MODE_4_DISPLAY_STR_ID=26  <== customize
```

The template shown above is present in the `VortexConfiguration.properties` file. These are all commented out (as indicated by # sign preceding the `CFG_*` parameters).

Follow the template exactly if you intend to add more modes than are available now. The above mentioned modes are potentially the most practical and useful.

`ID`:	Always a two times the previous value, starting with two for *Inbox* (default).
`IMPL`:	Class that implements the `com.iplanet.ecommerce.vortex.arch.IExternalNotifier` interface corresponding to the notification mode.
`STR_ID`:	The display string ID corresponding to the notification mode in `VortexStrings.properties` the String corresponding to the ID (in `VortexStrings.properties`) appears in the `NotificationConfiguration` Screen.

The impl class provided by default in iPlanet Market Maker is tuned and multi-threaded for efficiency. We recommend that you not change the *Inbox* and *SMTP* (email) configuration as provided.

Effects on Modules

Auction, RFx, Exchange

- When disabled, these modules are removed completely from the GUI.
- When disabled, anything accessing one of the modules (for example, all menu items, hrefs, and buttons) disappear.

Pricing

- When disabled, the pricing GUI (for example, approve and create pricing rules) disappears.
- When disabled, catalog does not show negotiated prices (previously this was configurable via a catalog configuration).
- When disabled, OMS does not apply pricing rules to orders. This ensures that existing pricing rules (those in place when pricing is disabled) are not applied.

OMS

- When disabled, this functionality provides OMS with a way to get a list of settled transactions (for example, from shopping cart, RFx, auction and exchange) out of the system in a consistent way.
- When disabled, the OMS GUI disappears. However, the ability to create requisitions (from the shopping cart, RFx, auction, and exchange) is still present. For example, the *Create Requisition* button is still visible. All settled transactions (requisitions) are routed via ESV (a functionality already available).

NOTE Currently, the createRequisition() API called by all modules can be routed via ESV by a global switch. This switch is set when OMS is disabled). ECXpert is not required to receive these requisitions (for example, they can be created on the file system by default).

Catalog

- When disabled, the catalog GUI (for example, maintenance, browse, shopping cart) disappears.
- When disabled, all references by other modules to the catalog (for example, add catalog items in Auction and RFx) disappear from the GUI as well.
- When disabled, the maintain master catalog menu item is moved under the Auction menu (instead of the Catalog menu), since the auction ontology is dependent on the catalog. This allows the auction ontology to be maintained in the absence of catalog.

Community

- When disabled, the community GUI disappears. However, community search screens used by other modules are still present.

Base

- When disabled, the Administration menu (accessed for logging and notification) disappears from the GUI.

Supported Test Cases

Because the number of module permutations is too great to test, only the following test cases are supported for Market Maker at this time:

- Test Case #1—Auction + Catalog + Pricing + Community + Base
- Test Case #2—Auctions Only
- Test Case #3—RFx + Catalog + Pricing + OMS + Community + Base
- Test Case #4—Exchange + Community + Base
- Test Case #5—Catalog + OMS + Community + Base

Test Case #1

Auction + Catalog + Pricing + Community + Base

This configuration contains everything except OMS, RFx, and Exchange.

Menu Bar

Order Management, RFx and Exchange main menu selections are not present.

Community

Advanced Search: RFx Module and Exchange Module do not appear as Market Place Services.

Company Profile and Create Company: RFx Module and Exchange Module do not appear as a Business Services.

Create Role, Role Detail: Exchange Module, Order Management Module and RFx Module privileges do not appear.

Administration

Configure Notification, Configure Logging and Log Viewer: Exchange Module, RFx Module and OMS Module are not in the module list.

Inbox: Messages never appear for Exchange, RFx or OMS Module.

Auctions

Settled auctions are automatically routed through ESV.

Test Case #2

Auctions Only

This configuration supports nothing but Auctions.

Menu Bar

Only the Auction menu is present. However, since the auction ontology is dependent on the catalog, for the immhost user, the Maintain Master Catalog sub-menu needs to appear under the Auctions main menu.

Auctions

Settled auctions are automatically routed through ESV.

Test Case #3

RFx + Catalog + Pricing + OMS + Community + Base

This configuration contains everything but Auctions and Exchange.

Menu Bar

Auctions and Exchange main menu selections are not present.

Catalog

Product Details: Initiate Reverse Auction button is not present.

RFx

Request Details: Auction It! link on Line Items is not present.

Community

Advanced Search: Auctions Module and Exchange Module do not appear as Market Place Services.

Company Profile and Create Company: Auctions Module and Exchange Module do not appear as a Business Services.

Create Role, Role Detail: Exchange Module and Auction Module privileges do not appear.

Administration

Configure Notification, Configure Logging and Log Viewer: Exchange Module and Auction Module are not in the module list.

Inbox: Messages never appear for Exchange or Auction Module.

Test Case #4

Exchange + Community + Base

This configuration contains only Exchange and supporting modules.

Menu Bar

Only Exchange, Community and Administration main menu items are present.

Community

Advanced Search: Auctions Module and RFx Module do not appear as Market Place Services.

Company Profile and Create Company: Auctions Module and RFx Module do not appear as a Business Services.

Create Role, Role Detail: Order Management Module, Pricing Module, Catalog Module, Auction Module and RFx Module privileges do not appear.

Administration

Configure Notification, Configure Logging and Log Viewer: Order Management Module, Pricing Module, Catalog Module, Auction Module and RFx Module are not in the module list.

Inbox: Messages never appear for Order Management Module, Pricing Module, Catalog Module, Auction Module and RFx Module.

Test Case #5

Catalog + OMS + Community + Base

This configuration has everything but Auctions, RFx, Pricing and Exchange.

Menu Bar

Auctions, RFx, and Exchange main menu selections are not present. Pricing Rules sub-menu is not present under the Catalog main menu.

Catalog

Negotiated Price field should not be present on the following pages: Browse/Search, Item Detail, Compare Items, Shopping Cart, Requisition Details and Order Details (both Buyer and Seller). Furthermore, verify pricing rules in place before disabling the pricing module are no longer being applied.

Product Details: Initiate Reverse Auction button is not present

Community

Advanced Search: Auctions Module, RFx Module and Exchange Module do not appear as Market Place Services.

Company Profile and Create Company: Auctions Module, RFx Module and Exchange Module do not appear as a Business Services.

Create Role, Role Detail: Exchange Module, Auction Module and RFx Module privileges do not appear.

Administration

Configure Notification, Configure Logging and Log Viewer: Exchange Module, Pricing Module, Auction Module and RFx Module are not in the module list.

Inbox: Messages never appear for Exchange Module, Pricing Module, Auction Module and RFx Module.

Appendix A

Configurable Property Names

This appendix contains the following sections:

- Overview
- Configure Properties for iPlanet Market Maker
- Configurable Properties for the Auction Module
- Configurable Properties for the Catalog Module
- Configurable Properties for the Community Module
- Configurable Properties for the iPlanet Market Maker Display
- Configurable Properties for the OMS Module
- Configurable Properties for the Pricing Module
- Configurable Properties for the RFx Module

Overview

The following tables list the configurable properties for the modules. Before you configure a property, make sure that you understand what the property does and how it affects the module. Failure to do so can cause the module to fail. Also, note that you must restart the web server for property changes to take effect.

This chapter describes the following types of properties.

- IPlanet Market Maker
- Auction Module
- Catalog Module

- Community Module
- Display
- Order Management Module
- Pricing Module
- RFx Module

The property files that contain the properties are in the following directories.

`<ias_dir>/ias/APPS/imm40/imm40/WEB-INF/classes`

`<imm_dir>/iMM/resources`

When you make changes to the `Vortex.Configuration.properties` file, make sure that your edits are in both of these directory locations.

Configure Properties for iPlanet Market Maker

The following properties are in the `Vortex.Configuration.properties` file.

Table A-1 Market Maker Configuration Properties

Property Name and Value	Description
`CFG_DATABASE_DRIVER`	Specifies the Oracle JDBC driver class.
`CFG_DATABASE_URL`	Specifies the Oracle database connectivity URL. This field is typically encrypted.
`CFG_DATABASE_USER`	Specifies the database user. This property should be encrypted.
`CFG_DATABASE_PASSWORD`	Specifies the database password. This property should be encrypted
`CFG_DEBUG_LEVEL`	Specifies the debug level for the iPlanet Market Maker system. All debug output is written to a file specified by the `CFG_DEBUG_DESTINATION` property. Acceptable values are off=3, terse=7, verbose=15, vverbose=31.
`CFG_DEBUG_DESTINATION`	Specifies the directory path for the debug log file.
`CFG_DEBUG_STDOUT`	Specifies whether to write debug output to the standard output stream.

Table A-1 Market Maker Configuration Properties *(Continued)*

Property Name and Value	Description
`CFG_LDAP_HOST`	Specifies the LDAP server host. Note that this property is typically encrypted and must also be set in the `userxconfig.properties` file
`CFG_LDAP_PORT`	Specifies the LDAP server port. Note that this property is typically encrypted and must also be set in the `userxconfig.properties` file.
`CFG_LDAP_READ_USER`	Specifies the name of an LDAP user with read privilege. This property should be encrypted.
`CFG_LDAP_READ_PASSWORD`	Specifies the LDAP password corresponding to `CFG_LDAP_READ_USER`. This property should be encrypted.
`CFG_LDAP_WRITE_USER`	Specifies the name of an LDAP user with write privilege (e.g. admin or cn="Directory Manager"). This property should be encrypted.
`CFG_LDAP_WRITE_PASSWORD`	Specifies the LDAP password corresponding to `CFG_LDAP_WRITE_USER`. This property should be encrypted.
`CFG_DEFAULT_LANGUAGE_CODE`	Specifies the default Language Code for the market maker.
`CFG_DEFAULT_COUNTRY_CODE`	Specifies the default Country Code for the market maker.
`CFG_DEFAULT_DATE_FORMAT`	Specifies the default date format used by the toString() method of the iPlanet Market Maker Date class.
`CFG_NOTIFIER_REMOTABLE_FACTORY`	Specifies the factory used to create the notification Remotable interface. By subclassing, you can augment processing as notifications are posted.
`CFG_NOTIFIER_SMTP_HOST`	Specifies the SMTP host for the Notifier.
`CFG_LOG_REMOTABLE_FACTORY`	Specifies the factory used to create the log Remotable interface. By subclassing, you can augment processing as messages are logged.
`CFG_DEFAULT_TIME_ZONE`	Specifies the default TimeZone value for the market maker.
`CFG_DEFAULT_CURRENCY`	Specifies the default Currency value for the market maker.

Table A-1 Market Maker Configuration Properties *(Continued)*

Property Name and Value	Description
CFG_LOG_THREAD_SLEEP_TIME	Specifies the polling interval of the background configuration thread in milliseconds.
CFG_ATTACHMENT_DEFAULT_MAX_SIZE	Specifies the maximum size of the file attachment.This defaults to 2M bytes.
CFG_ATTACHMENT_DEFAULT_SAVE_DIRECTORY	Specifies the directory in which file attachments are saved.
CFG_MODULES_TO_INITIALIZE	Specifies the list of modules that are initialized (plugged into the iPlanet Market Maker bus). This can be used to enable or disable the Auction and RFx modules, but should not be modified.
CFG_DEBUG_MODULES_TO_DEBUG	Defines the list of modules for which debugging output will be displayed. If none are specified, then debugging for all modules is enabled. These are defined as follows: MODULE_BASE=0,MODULE_RFX=1, MODULE_AUCTION=2, MODULE_CATALOG=3, MODULE_PRICING=4,MODULE_COMMUNITY=5, MODULE_DISPLAY=6, MODULE_OMS=7, MODULE_CATALOG_IMPORT=8
CFG_DEV_LOG	Specifies the location of the log file used by the FileDebug utility.
CFG_SMTP_PORT	Specifies the SMTP port used by Notification facility.
CFG_MAX_THREADS_IN_NOTIFIER_POOL	Defines the initial number of threads created to post SMTP messages.
CFG_VALID_COUNTRIES	Specifies the list of valid countries (ISO standard country codes) to display (for example, when defining the user profile). Note if this property is empty, all are countries considered valid.
CFG_VALID_LOCALE_LANGUAGES	Specifies the list of valid languages to display (for example. when defining the user profile). Note if this property is empty, all languages are considered valid.

Table A-1 Market Maker Configuration Properties *(Continued)*

Property Name and Value	Description
`CFG_ACTIVATE_DP`	Specifies an internal setting, set to true to enable the display profile code. This property should be set to true.
`CFG_JS_REMOTABLE_FACTORY`	Specifies the remotable factory for the JobScheduler. This allows the job scheduler to be subclassed.
`CFG_JS_EXE`	Specifies the JobScheduler process. This allows flags (for example, -Djava.compiler=none) to be passed to the job scheduler.
`CFG_JS_ONCEONLY_NUM_WORKERS`	Specifies the total number of worker threads in the thread pool for the OnceOnly Timer in the Job Scheduler.
`CFG_JS_REPEAT_NUM_WORKERS`	Specifies the total number of worker threads in the thread pool for the Repeat Timer in the Job Scheduler.
`CFG_JS_ONCEONLY_SLEEP_INTERVAL`	Specifies the sleep interval (in milliseconds) for the OnceOnly timer database thread maintained by the Job Scheduler.
`CFG_JS_REPEAT_SLEEP_INTERVAL`	Specifies the sleep interval (in milliseconds) for the Repeat timer database thread maintained by the Job Scheduler.
`CFG_NOTIFIER_POOL_WAIT_INTERVAL_FOR_CLEANUP`	Used for internal testing.
`CFG_DEFAULT_JSP`	Used for internal testing. Personalizes the default selected JSP at login time.
`CFG_IN_DEPLOYMENT_MODE`	Used for internal testing. When false this enables the nocache header. This property should be set to true.
`CFG_CRYPTO_SALT`	Specifies the salt as needed by the PKCS #5 standard.
`CFG_CRYPTO_COUNT`	Specifies the iteration count as needed by the PKCS #5 standard.
`CFG_CRYPTO_PROVIDER`	Specifies the Java Cryptography Extension 1.2.1 provider name.
`CFG_CRYPTO_CIPHER_TYPE`	Specifies the type of JCE cipher used for encryption.

Table A-1 Market Maker Configuration Properties *(Continued)*

Property Name and Value	Description
`CFG_CRYPTO_KEY_FACTORY_TYPE`	Specifies the type of JCE key factory to used.
`CFG_MAX_NOTIFIER_THREADS_ALLOWED`	Specifies the maximum number of threads in the pool. Do not change this value.
`CFG_CPOOL_INITIAL`	Specifies the initial number of initial database connections created in the connection pool.
`CFG_CPOOL_MAXIMUM`	Specifies the maximum number of database connections created in the connection pool.
`CFG_CPOOL_IDLE_TIMEOUT`	Specifies the timeout for stale database connections in milliseconds.
`CFG_CPOOL_WAIT_TIMEOUT`	Specifies the maximum time in milliseconds to wait for a connection to be released when all database connections are in use.
`CFG_CRYPTO_KEYFILE`	Specifies the file that holds the password to the JCE crypto key.
`CFG_JS_WATCHDOG_EXE`	Specifies the JobScheduler WatchDog process that ensures the Job Scheduler process is running.
`CFG_LNC_REMOTABLE_FACTORY`	Specifies the class name of the Remotable used to write Logging and Notification configuration information to the RDBMS.
`CFG_TRANSACTIONAL_MAIL_SCHEDULER_INTERVAL`	Specifies the polling interval for the scheduled job that posts unsent messages in the Notification subsystem.
`CFG_DEFAULT_NOTIFICATION_INITIATOR`	Specifies a default email address when the sender of the notification has not specified an email address. This is possible because a user's email address is not a required field.
`CFG_DATA_INTEGRITY_OBJECT`	Specifies the license key.
`CFG_DB_DATE_SYNC_INTERVAL`	Specifies in milliseconds how often to poll the database machine to synchronize the date. The intent is to synchronize the database clock with the server's clock if they are on different machines. The synchronization keeps the clocks within a few seconds. The default is 30 minutes
`CFG_JS_HOST`	Specifies he host name in which the JobScheduler is running.

Table A-1 Market Maker Configuration Properties *(Continued)*

Property Name and Value	Description
CFG_JS_PORT	Specifies the port number in which the JobScheduler listens. Use an unused TCP port on your machine. The JS_HOST and PORT are used to ensure only a single instance of the Job Scheduler is running.
CFG_CURRENCY_EXCHANGE_REFRESH_INTERVAL	Specifies the amount of time between refreshes for currency exchange rates. The unit is in minutes.

Configurable Properties for the Auction Module

The following properties are in the Auction.Configuration.properties file.

Table A-2 Auction Configuration Properties

Property Name and Value	Description
CFG_DEFAULT_NUM_ROWS_PER_PAGE Property # = 3; Current Setting : (3=10)	Sets the number of rows displayed per page.
CFG_SHOW_N_COL Property # = 4; Current Setting : (4=3)	Sets the number of columns displayed per page.
CFG_SHOW_N_SUBCATEGORIES Property # = 5; Current Setting : (5=2)	Sets the number of auction subcategories displayed below the second level of categories when browsing by category.
#CFG_ARCHIVE_CHECKING_RATE , the unit # is millisecond. # - the frequency at which we check # for auctions eligible for # archiving, the unit is in # milliseconds. # - Currently set to 1 week, which is 7 * 24 * 60 * 60 * 1000 = 604800000 millisecond. 6=604800000	Represents the frequency (in milliseconds) at which auctions are checked to be eligible for archiving. If you want to change the frequency for auction archive checking, change the current setting (1 week) from 6=604800000 to 6=(the number of milliseconds; result of calculation <number of weeks> * 24 * 60 * 60 * 1000).
CFG_ENABLE_ARCHIVE_SCHEDULER Property # =7; Current Setting: 7=false	Enables or disables the archive scheduler.

Table A-2 Auction Configuration Properties *(Continued)*

Property Name and Value	Description

`CFG_ATTACHMENT_TMP_DIR`
`Property # =8; Current Setting:`
 	
`8=@IMM_DOCROOT_FULL@/imm40/iMM/auction/images/uploaded_images/tmp`
	Specifies the temporary directory for auction images.

`CFG_ATTACHMENT_DIR`
`Property # =9; Current Setting:`
`9=@IMM_DOCROOT_FULL@/imm40/iMM/auction/images/uploaded_images`
	Specifies the permanent directory for auction images.

`CFG_IMAGE_ATTACHMENT_TMP_URI`
`Property # = 10; Current Setting:`
`10=/imm40/iMM/auction/images/uploaded_images/tmp`
	Specifies the temporary relative uniform resource identifier (URI) to store the image files.

`CFG_IMAGE_ATTACHMENT_URI`
`Property # = 11; Current Setting:`
 	
`11=/imm40/iMM/auction/images/uploaded_images`
	Specifies the permanent relative URI to store the image files.

`CFG_CATALOG_IMAGE_DIR`
`Property # = 12; Current Setting:`
`12=@IMM_DOCROOT_FULL@`
	Directory in which to store catalog images.

`CFG_ARCHIVE_CLOSING_DELAY`
`Property # = 13;`
`Current Setting : (13=14400)`
`( 14400 minutes = 10days * 24 * 60)`
	Represents the maximum delay (duration in minutes) allowed between the closing date (time in minutes) and the check point at which the time is checked (time in minutes). If you want to change the delay, change the current settings from 13=14400 to 13=(the number of minutes).

`CFG_NOTIFY_ALL_INITIATORS`
`Property #=27;`
`Current Setting : 27 = false`
	This parameter indicates whether to send notification to everyone in the company with initiating privilege or just to the initiator who created the auction.

`CFG_NOTIFY_ALL_BIDDERS`
`Property #=28;`
`Current Setting : 28 = false`
	This parameter indicates whether to send notification to everyone in the company with bidding privilege or just to the bidder who bid on the auction.

The following properties are in the `Auction.Messages.properties` file.

Table A-3 Auction Message Properties

Message Name and Value	Description
Error codes 20000 to 20028	Auction error messages. If you change an error message, change the text against the error code.
Error code 22024 to 22068	Auction general messages for Auction status, bid, invitation, win, and so on.

Configurable Properties for the Catalog Module

The following properties are in the `Catalog.Configuration.properties` file.

Table A-4 Catalog Configuration Properties

Property Name and Value	Description

`#CFG_FORMAT_SEARCH_STRING`
 	
`7=true`
	True means that the search strings you type in the browser will be preprocessed for the Oracle interMedia search engine. For example, special characters are removed or translated and "AND" is inserted between consecutive words. True is the recommended value.
	
	False means the string you type in string is immediately passed to the search engine without being preprocessed.

`#CFG_NO_ADJUSTED_PRICES`
 	
`8=false`
	Default number of items to show on one page on the catalog search and browse pages. You can change this value on the browse page as well. Possible values are 10, 25, 50, 100.

`#CFG_NESTED_CATEGORIES_SIZE`
 	
`10=16`
	The catalog browse page can show nested categories when browsing. If the number of categories being displayed during browsing is less than this configuration parameter, then up to three subcategories are also displayed below the parent.
	
	Set this to zero if no subcategories are to be displayed, or set to a very high number if subcategories should always be shown. No nested categories are shown when displaying search results. The recommended default value is 16.

Table A-4 Catalog Configuration Properties

Property Name and Value	Description
`#CFG_DOUBLE_COLUMN_SIZE` `11=32`	Categories can be displayed in one or two columns. If categories names are long, then a one column display is better. If category names are generally short, then two column of categories are a better use of screen space. Also, if nested categories are being displayed, it is less likely that double columns will be the best presentation, so this value should be set to a lower number. The recommended default is 32 characters.

The following properties are in the `Catalog.Messages.properties` file.

Table A-5 Catalog Message Properties

Message Name and Value	Description
Error codes 30000 to 30011	Catalog error messages. If you change an error message, change the text against the error code.
Error code 80000 to 80010	Catalog error messages. If you change an error message, change the text against the error code.

Configurable Properties for the Community Module

The following properties are in the `Community.Configuration.properties` file.

Table A-6 Community Configuration Properties

Property Name and Value	Description
`CFG_SYSTEM_USER_ID`	System user ID.The default is `system.`
`CFG_SYSTEM_PASSWORD`	System user password. The default is `system`.
`CFG_SITE_ID`	Site ID for the LDAP directory structure. The default is `imm`.
`CFG_HOST_COMPANY_ID`	Host ID under which the market administrator operates. The default is `immhost`.
`CFG_ROOT_USER_ID`	User ID which the Market Maker Administrator uses to log in. The default is `root`.

Table A-6 Community Configuration Properties *(Continued)*

Property Name and Value	Description
`CFG_MMADMIN_ROLE - ID`	Market maker administrator role ID. The default is `mmadmin`.
`CFG_COMPANY_ADMIN_ROLE`	Company administrator roleID. The default is `companyadmin`.
`CFG_EXTENDED_INFO_MANAGER`	Path for the `ExtendedInfoManager` class you can use to extend the Community module. The default is `com.iplanet.ecommerce.vortex.community.ExtendedInfoManager`.
`CFG_ANONYMOUS_USER_ID`	User ID for anonymous log in. The default is `anonymous`.
`CFG_DEFAULT_ORG_USER_LIMIT`	Maximum number of users that can be registered into any company. The default is 500.
`CFG_VORTEX_BASE_DN`	Base ID for the LDAP directory from which distinguished names (DN's) may be formulated. The default is `imm`.
`CFG_SEARCH_MAX_RESULTS`	Maximum number of search results that is returned for a search. The default is 50.
`CFG_DEFAULT_NUM_ROWS_PER_PAGE`	Maximum number of results displayed on a page. The default is 10.
`CFG_OPTIONAL_MODULES`	List of optional modules you can plug in.The defaults are `auction` and `catalog`.

The following properties are in the `Community.Messages.properties` file.

Table A-7 Community Message Properties

Message Name and Value	Description
Error code 50000 to 50024 Error code 51000 to 51018	Contains warning and error messages generated by the back-end of the Community module.

Configurable Properties for the iPlanet Market Maker Display

The following properties are in the `Display.Configuration.properties` file.

Table A-8 Display Configuration Properties

Property Name and Value	Description

`#CFG_GENERIC_CONFIG`
`0=Generic Config`
	Define a generic iPlanet Market Maker configuration.

`#LOGIN PROPERTIES`
`# Pages used for login`
`4=community/login/Login.jsp`
`8=community/login/AskRole.jsp`
	Specifies the log-in JSPs used when you log in to iPlanet Market Maker.

`#CFG_APPNAME: name of the application`
`14=iMM`
	Specifies the name of the application in LDAP. The default is iPlanet Market Maker.

`# IMM Doc root: name of the directory under which iMM will appear on the server url`
`16=/imm40/iMM/`
	Specifies the iPlanet Market Maker root directory name. The default is @IMM_DOCROOT@.

`#CFG_SEND_NOCACHE_HEADER: send a nocache http header with each page served by the portal.`
`19=false`
	Sends a nocache header with each page served by the portal.

The following properties are in the `Display.Messages.properties` file.

Table A-9 Display Message Properties

Message Name and Value	Description
Error code 60000 to 60029	Error Messages related to URL, login, display profile, and event handlers.

Configurable Properties for the OMS Module

The following properties are in the `OMS.Configuration.properties` file.

Table A-10 OMS Configuration Properties

Property Name and Value	Description

`#CFG_GENERIC_CONFIG`
`0=GenericConfig`
	Specifies a generic configuration.

`#CFG_OMS_MANAGER`
`1=com.iplanet.ecommerce.vortex.oms.OMSManagerFactory`
	Name of the factory class for the OMS manager class.

`#CFG_OMS_REMOTABLE`
`2=com.iplanet.ecommerce.vortex.oms.OMSRemotableFactory`
	Name of the factory class for the OMS remotable class.

`#CFG_ROWS_TO_DISPLAY`
`3=10`
	Number of rows to display in OMS screens containing lists.

The following properties are in the OMSStrings.properties file.

Table A-11 OMS Message Properties

Property Name and Value	Description
#OMS_PAYMENT_METHOD_NAMES	Payment methods for orders. The default is AMEX (American Express Card).
OMS_PAYMENT_METHOD_NAMES[0]=VISA	
OMS_PAYMENT_METHOD_NAMES[1]= MasterCard	
OMS_PAYMENT_METHOD_NAMES[2]=COD	
OMS_PAYMENT_METHOD_NAMES[3]=PCARD	
OMS_PAYMENT_METHOD_NAMES[4]=AMEX	
5=AMEX	
#OMS_SHIPPING_METHOD_NAMES	Specifies the shipping method for orders.
OMS_SHIPPING_METHOD_NAMES[0]=DHL	
OMS_SHIPPING_METHOD_NAMES[1]=FedEx	
OMS_SHIPPING_METHOD_NAMES[2]=USPS	
OMS_SHIPPING_METHOD_NAMES[3]=UPS	
#OMS_PAYMENT_TERM_NAMES	Specifies the payment terms for orders.
OMS_PAYMENT_TERM_NAMES[0]=NET15DAYS	
OMS_PAYMENT_TERM_NAMES[1]=NET30DAYS	
OMS_PAYMENT_TERM_NAMES[2]=NET45DAYS	
OMS_PAYMENT_TERM_NAMES[3]=NET60DAYS	

Configurable Properties for the Pricing Module

The following properties are in the Pricing.Configuration.properties file.

Table A-12 Pricing Configuration Properties

Property Name and Value	Description
#CONSTANTS FOR PRICING DISPLAY MODULE #STR_PRICING_DISPLAY_START_YEAR 100=2000	Sets the start year to display in the listbox that shows the range of years. If you want to change pricing display start year to 2001, for example, change the current setting from100=2000 to 100=2001. Note that must change the #STR_YEAR_STR fields accordingly in the Pricing.Strings.properties file if you change the start year with this property.
#STR_PRICING_DISPLAY_END_YEAR 101=2003	Sets the end year to display in the listbox that shows the range of years. If you want to change pricing display end year to 2004, for example, change the current setting from 100=2003 to 100=2004. Note that must change the #STR_YEAR_STR fields accordingly in the Pricing.Strings.properties file if you change the end year with this property
#STR_PRICING_DISPLAY_ROW_INCREMENT 102=4	Sets the row increment in the pricing display, which is the number of empty rows for dates to be filled in. If you want the row increment to be 5 in pricing display, for example, change the current setting from 102=4 to 102=5
#STR_PRICING_DISPLAY_NO_RECORDS_PER_ PAGE 103=10	Sets the number of records per page in the pricing rules display in a page. If you want the number of records per page to be 4 in pricing display, for example, change the current setting from 103=10 to 103=11.
#STR_PRICING_DISPLAY_NO_ITEMS_PER_ LISTBOX 105=4	Sets the number of catalog items per listbox.If you want the number of items per listbox to be 5, for example, change the current setting from 105=4 to 105=5.

The following properties are in the `Pricing.Messages.properties` file.

Table A-13 Pricing Message Properties

Message Name and Value	Description
Error code 40000 to 40022	General and error messages related to pricing rules administration.
Error code 40100 to 40112	Error messages related to pricing rules display.

Configurable Properties for the RFx Module

The following properties are in the `RFx.Configuration.properties` file.

Table A-14 RFx Configuration Properties

Property Name and Value	Description

`#CFG_ENABLE_ARCHIVE_SCHEDULER`
`# - true: turn on. false: no action`
`3=false`
	Turns the archive scheduler on or off.

`#CFG_ARCHIVE_CHECKING_RATE`
`# - the frequency at which we check for archiving RFx data.`
`# - the unit is in millisenconds.`
`# - currently set to 100`
`4=8640000000days`
	Frequency to check for archiving data.

`#CFG_ARCHIVE_CLOSING_DELAY`
`# - the maximum delay allowed between the closing date and the check point.`
`# - the unit is in msecs.`
`# - currently set to 100 days`
`5=8640000000`
	The maximum delay allowed for archiving between the closing time and the check point at which the time is checked.

Table A-14 RFx Configuration Properties *(Continued)*

Property Name and Value	Description

`#CFG_RFX_JOB_SCHEDULER_RUN_INTERVAL`
 	
`# - the frequency at which we check`
`# for running the job scheduler.`
 	
`# - the unit is in millisenconds.`
 	
`# - currently set to 1 day`
 	
`7=86400000`
	Frequency to check for running the job scheduler.

`#CFG_PADDING_MINUTES_FOR_MAX_EXPIRATION_DATE`
 	
`# this is the number of minutes to`
`# pad to expiration date so that`
 	
`# validation of (if`
`# getExpirationDate() >`
`# maxExpirationDate) is`
 	
`#more lenient.`
 	
`# Note: this does not need to be`
`# dynamic.`
 	
`9=60`
	Number of minutes to add to (pad) the expiration date to make the date deadline more lenient.

`#CFG_SPAWN_THREAD_FOR_NOTIFICATION`
 	
`10=false`
	Runs notification as a separate thread. Leave this set as `false`.

`#CFG_DISPLAY_WARNINGS_AS_MESSAGES`
 	
`11=false`
	Turns warning messages in the user interface on or off.

Index