

Catalog Import Guide

iPlanet Market Maker

Version 4.5

March 2002

Copyright © 2002 Sun Microsystems, Inc. All rights reserved.

Sun, Sun Microsystems, the Sun logo, iPlanet, Java and Solaris are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries.

Federal Acquisitions: Commercial Software—Government Users Subject to Standard License Terms and Conditions

The product described in this document is distributed under licenses restricting its use, copying, distribution, and decompilation. No part of the product or this document may be reproduced in any form by any means without prior written authorization of the Sun Microsystems, Inc. and its licensors, if any.

THIS DOCUMENTATION IS PROVIDED “AS IS” AND ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT SUCH DISCLAIMERS ARE HELD TO BE LEGALLY INVALID.

Copyright © 2002 Sun Microsystems, Inc. Tous droits réservés.

Sun, Sun Microsystems, et Sun logo, iPlanet, Java et Solaris sont des marques de fabrique ou des marques déposées de Sun Microsystems, Inc. aux États-Unis et d'autres pays.

Le produit décrit dans ce document est distribué selon des conditions de licence qui en restreignent l'utilisation, la copie, la distribution et la décompilation. Aucune partie de ce produit ni de ce document ne peut être reproduite sous quelque forme ou par quelque moyen que ce soit sans l'autorisation écrite préalable de Sun Microsystems, Inc. et, le cas échéant, de ses bailleurs de licence.

CETTE DOCUMENTATION EST FOURNIE “EN L'ÉTAT”, ET TOUTES CONDITIONS EXPRESSES OU IMPLICITES, TOUTES REPRÉSENTATIONS ET TOUTES GARANTIES, Y COMPRIS TOUTE GARANTIE IMPLICITE D'APTITUDE À LA VENTE, OU À UN BUT PARTICULIER OU DE NON CONTREFAÇON SONT EXCLUES, EXCEPTÉ DANS LA MESURE OÙ DE TELLES EXCLUSIONS SERAIENT CONTRAIRES À LA LOI.

Contents

Contents	3
List of Figures	5
List of Tables	7
About This Document	9
Audience	9
What's in This Document	9
Documentation Conventions	10
The Document Online	10
Related Documentation	11
Other Useful Documentation	11
Product Support	12
Chapter 1 Catalog Concepts	15
Public Master Catalog (PMC)	15
Defining the Catalog Structure	16
Defining Item Attributes	16
Identifying Items	17
Mapping Catalog Hierarchies	18
Attribute Normalization	20
Attribute Name Normalization	20
Renaming Attribute Units	21
Specifying Attribute Types	21

Chapter 2 Representing Catalog Information	23
Representing Catalogs	24
Creating the Seller Catalog and PMC in XML	24
Adding Column Structure to the CSF File	26
Adding, Deleting, and Updating Items	27
Using Column References	30
Creating the Seller Catalog	32
XML Representation Issues	33
Detecting Clashes	33
Resolving Name Clashes	33
Resolving Path Clashes	37
Defining the Ontology Mapping	41
Changing the Ontology Mapping	41
Using the Human-Readable Difference Report Generator (HRDRG)	46
Inputs to the HRDRG Flow	47
Output of the HRDRG Flow	48
Chapter 3 Running the Import Utility	51
Dependency Trees	51
Setting Up Your Environment	54
Preparing To Run the Import Utility	54
Specifying Output Targets	55
Specifying the Required Source File Names	56
Specifying the Optional File Names	57
Specifying the Ultimate Targets	58
Specifying the Ontology Mapping File	58
Specifying Optional Variables	58
Specifying Error Reports	59
Specifying Command Arguments	59
Specifying Internal Targets	60
Specifying Directory Locations	60
An Example of Running the Import Utility	64
Loading the Load File in the PMC	70
Exporting Catalogs	72
Running the Export Utility	73
Index	75

List of Figures

Figure 1-1	Catalog Hierarchy Structure Example	16
Figure 1-2	SKU Attribute Identifier	17
Figure 1-3	Mapping Items in the Seller Catalog to the PMC	19
Figure 1-4	Renaming Attribute Names	20
Figure 1-5	Renaming Attribute Units	21
Figure 2-1	Flow to Create the Seller Catalog and PMC in XML Format	25
Figure 2-2	CSF File Example	26
Figure 2-3	CSF File Annotated with Structural Information	26
Figure 2-4	CSS File Example	28
Figure 2-5	Seller Catalog Hierarchy Structure	30
Figure 2-6	CSF File Example with Different Currency Types	31
Figure 2-7	CSF File with Structural Information	31
Figure 2-8	Column Reference Section of a CSS File	31
Figure 2-9	Seller Catalog Represented in the <code>vendor.xml</code> File	32
Figure 2-10	Name Clash	34
Figure 2-11	Result of Name Clash	34
Figure 2-12	The <code>vendor.xml</code> File with a Name Clash	34
Figure 2-13	Edited Name Clash File	36
Figure 2-14	Name Clash Resolution	36
Figure 2-15	The <code>vendor.xml</code> File with a Resolved Name Clash	36
Figure 2-16	Path Clash	37
Figure 2-17	The <code>vendor.xml</code> File with a Path Clash	38
Figure 2-18	Edited Path Clash File	39
Figure 2-19	Path Clash Resolution	39

Figure 2-20	The <code>vendor.xml</code> file with Resolved Path Clashes	40
Figure 2-21	A <code>vendor.xml</code> Hierarchy Example	41
Figure 2-22	Resulting <code>omd.xml</code> Ontology Mapping File	42
Figure 2-23	Different <code>vendor.xml</code> and <code>mm.xml</code> Hierarchies	43
Figure 2-24	PMC <code>omd.xml</code> Ontology Mapping File	44
Figure 2-25	Sample <code>mm.xml</code> File	44
Figure 2-26	HRDRG Flow	46
Figure 2-27	HRDRG Input	47
Figure 2-28	A <code>full_diff</code> Utility <code>.csf</code> Example	47
Figure 2-29	A <code>vendor.xml</code> File with Changes Noted	48
Figure 3-1	Dependency Tree for Creating a Seller Ontology with No Existing PMC Ontology	52
Figure 3-2	Dependency Tree for Updating an Existing PMC Ontology	53
Figure 3-3	Sample <code>makefile</code> File	61
Figure 3-4	An <code>input.csf</code> Example	64
Figure 3-5	A <code>css.xml</code> Example	65
Figure 3-6	An <code>attr-renaming.xml</code> Example	66
Figure 3-7	A <code>units.xml</code> Example	67
Figure 3-8	Edited <code>makefile</code> file	67
Figure 3-9	The <code>mm.xml</code> File Created by the Import Utility	68
Figure 3-10	Export Utility Flow	73

List of Tables

Table 0-1	Chapter Summaries	10
Table 2-1	Seller Catalog Files	24
Table 2-2	Table Format Representation of the CSF File	27
Table 2-3	Action Results	28
Table 2-4	The <code>full_diff</code> Utility Keywords	48
Table 3-1	Output Targets	55
Table 3-2	Required Source Input File Names	56
Table 3-3	Optional Input File Names	57
Table 3-4	Ultimate Target File Names	58
Table 3-5	Optional Variable Names	58
Table 3-6	Error Report File Names	59
Table 3-7	Command Arguments	59
Table 3-8	Internal Targets	60
Table 3-9	Directory Locations	60
Table 3-10	Inputs to the <code>Loader</code> Command Line	70
Table 3-11	Inputs to the <code>Export</code> Utility Command Line	74

About This Document

The iPlanet Market Maker Catalog Import Guide provides information to the user who needs to import seller catalogs into an iPlanet Market Maker marketplace.

The following sections are contained in this preface:

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

Audience

The audience for this guide is the user who needs to import seller catalogs into an iPlanet Market Maker marketplace.

To import catalogs successfully, you must be familiar with the information in ", ", and the extensible markup language (XML). This information is especially important to know before importing a catalog for the first time.

What's in This Document

The following table summarizes what each chapter covers.

Table 0-1 Chapter Summaries

If you want to know about this	See this chapter
Concepts you need to be familiar with to prepare to import catalogs	Chapter 1, "Catalog Concepts"
Conventions that the Import utility uses to represent catalog data	Chapter 2, "Representing Catalog Information"
How to set up and run the Import utility	Chapter 3, "Running the Import Utility"

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 page.
- 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 format at the following web site:

<http://docs.iplanet.com/docs/manuals>

Related Documentation

NOTE Documentation for all iPlanet products can be found at the following web site:

<http://docs.iplanet.com/docs/manuals/>

The iPlanet Market Maker documentation set includes:

- *Release Notes*—Contains important information on the current release of iPlanet Market Maker. Read this document before working with the new iPlanet Market Maker release.
- *Installation Guide*—Provides instructions for installing the iPlanet Market Maker product and its enabling software.
- User interface Help (*Online Help*)—Provides guidelines and instructions for performing the iPlanet Market Maker tasks.
- *Catalog Import Guide*—Provides guidelines and instructions for setting up and running the Import utility.
- *Customization Guide*—Provides guidelines for customizing the existing iPlanet Market Maker user interface and module functionality.
- *Exchange Module Customization Guide*—Provides guidelines for customizing services to create exchanges.
- *Deployment Guide*—Provides guidelines for planning the deployment, installing and configuring the software, importing and configuring data, and tuning and monitoring performance.
- *Upgrade Guide*—Provides guidelines for planning, preparing and performing upgrades to the iPlanet Market Maker software.

Other Useful Documentation

- *iPlanet Application Server Installation Guide*—Provides guidelines for installing the iPlanet Application Server.
- *iPlanet Application Server Administrator's Guide*—Provides guidelines for tasks carried out by the administrator of one or more iPlanet Application Server machines.

- *iPlanet Web Server Administrator's Guide*—Provides guidelines for configuring, administering, and maintaining the iPlanet Web Server.
- *iPlanet Directory Server Documentation*—Provides guidelines for platform requirements and the iPlanet Directory Server installation.
- *ECXpert Administrator's Guide*—Explains concepts, structure, functions, and operation of iPlanet ECXpert, and provides guidelines for administering the system.
- *ECXpert Operations Reference Guide*—Explains ECXpert operational issues, and provides a reference on error messages that can be generated by ECXpert or passed through from third-party software components that ECXpert uses.
- *Oracle 8i and 9i Installation Guide for Sun SPARC Solaris*—Provides guidelines for installing Oracle.
- *Oracle 8i and 9i Designing and Tuning for Performance*—Provides guidelines for Oracle database design, and for tuning and monitoring performance.
- *Oracle 8i interMedia Text Reference*—Provides guidelines for text searching, retrieval, and viewing capabilities.
- *Actuate* documentation.

Product Support

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

- Visit the 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.

- Use the telephone dispatch number associated with your maintenance contract.

So that the technical support staff can best assist you in resolving problems, 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.

Catalog Concepts

This chapter describes the catalog concepts you need to understand to import your catalog data into an iPlanet Market Maker marketplace.

The following sections are contained in this chapter:

- Public Master Catalog (PMC)
- Defining the Catalog Structure
- Mapping Catalog Hierarchies

Public Master Catalog (PMC)

To make their products available in a marketplace, sellers organize product information in catalogs. These catalogs are the basis for buying and selling products. The ways of representing catalog information vary from seller to seller, even among sellers of similar products.

To facilitate searching and comparing products from different seller catalogs, iPlanet Market Maker software uses a Public Master catalog (PMC). The PMC is the result of aggregating and normalizing the information in all the individual seller catalogs. There is one PMC per marketplace. The marketmaker administrator maintains the PMC.

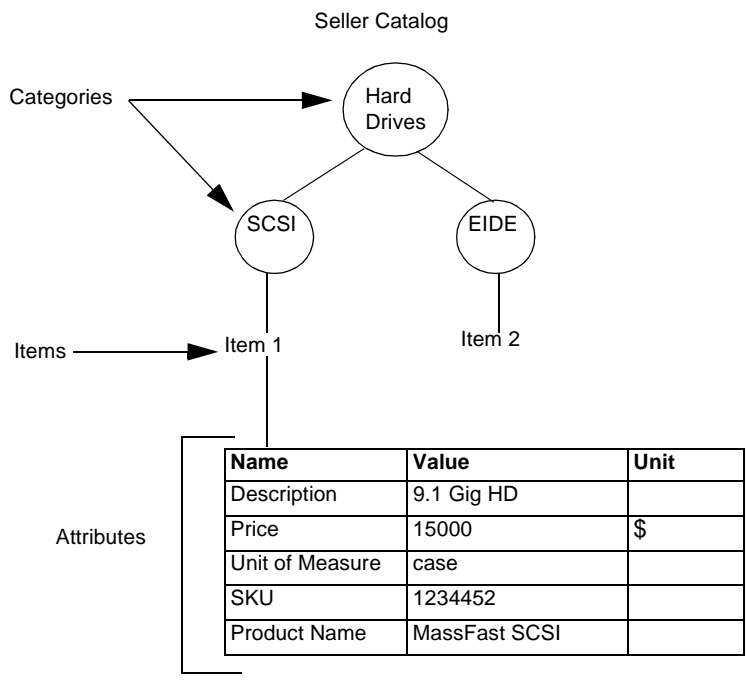
Defining the Catalog Structure

Seller catalogs and the PMC are a hierarchy of categories and items. A category can have other categories and items beneath it. Items are the “leaf” elements in a catalog, which means that there are no categories or items under an item. For example, a category called “office supplies” might have a sub-category called “desks.” An example of an item in this case might be a specific desk.

Defining Item Attributes

Each item under a category has an associated set of attributes. Each attribute has a name, a value, and, (optionally), a unit. These attributes provide information specific to an item, such as its price, product name, and so on. Figure 1-1 shows a sample catalog representation with categories, items, and attributes.

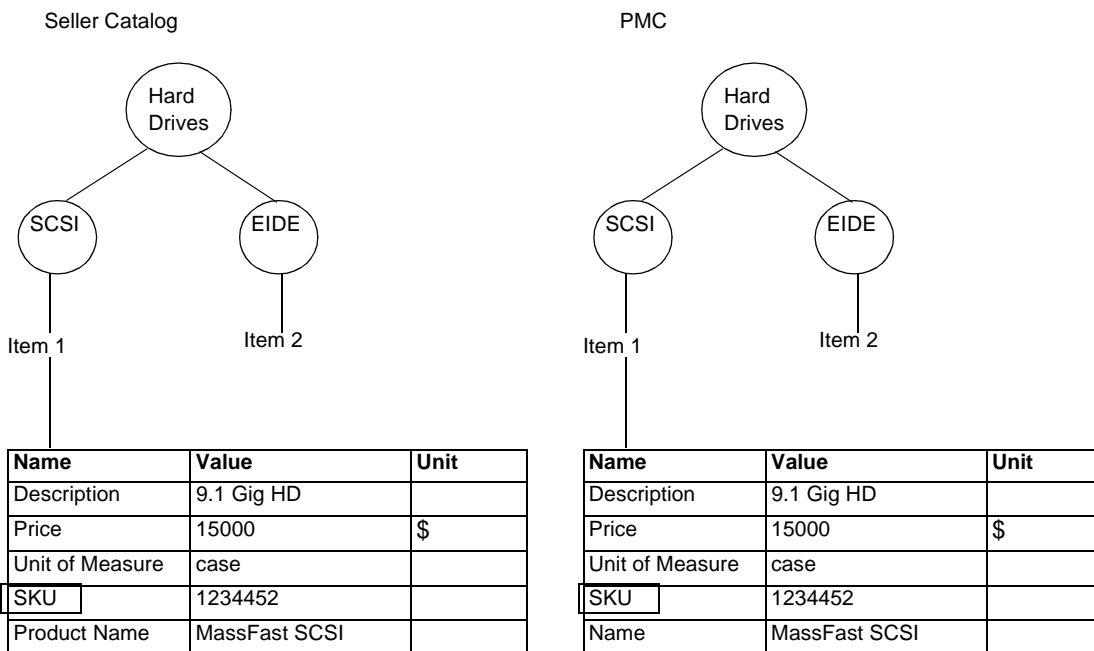
Figure 1-1 Catalog Hierarchy Structure Example



Identifying Items

In a seller catalog, all the items must be uniquely identified using some combination of categories and attributes. Seller catalogs often use the storage keeping unit (SKU) attribute to uniquely identify items, but you can use any combination of categories and attributes to uniquely identify items in your catalog. See Figure 1-2.

Figure 1-2 SKU Attribute Identifier

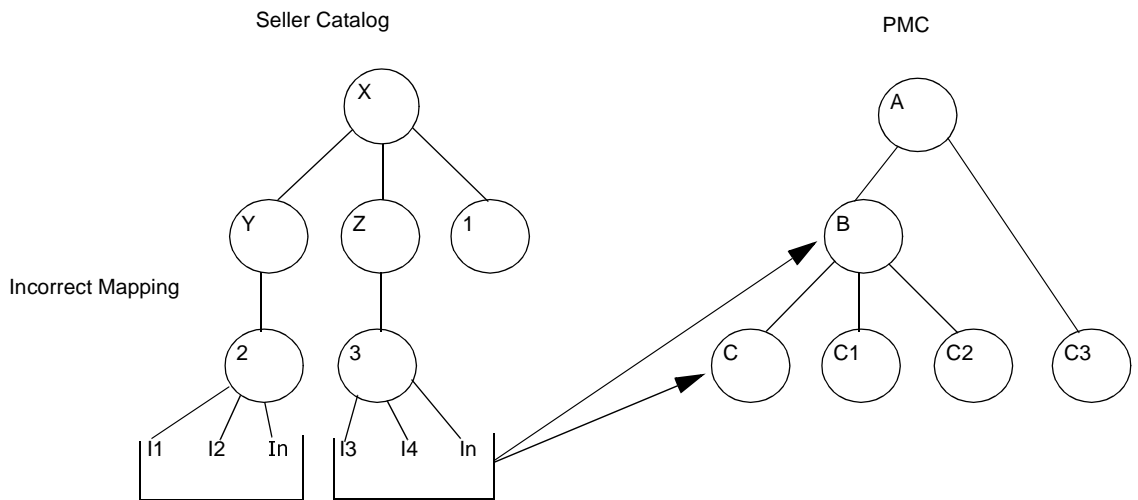
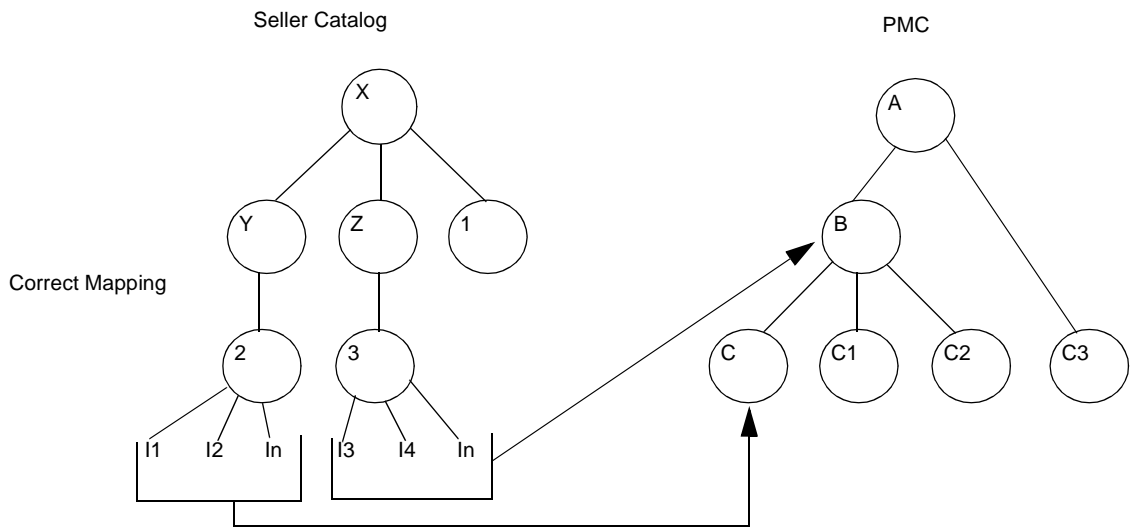


Mapping Catalog Hierarchies

The hierarchy of categories in a catalog is called an ontology. It is common for the ontology of one seller catalog to be different from that of another. The PMC has only one ontology. The process of mapping items from a seller catalog to the PMC is called ontology mapping.

Items in one category in a seller catalog can be mapped to a category in the PMC, but items from one category in a seller catalog cannot be mapped to multiple categories in the PMC. Figure 1-3 shows examples of correct and incorrect ontology mappings. The circles represent categories.

Figure 1-3 Mapping Items in the Seller Catalog to the PMC



Attribute Normalization

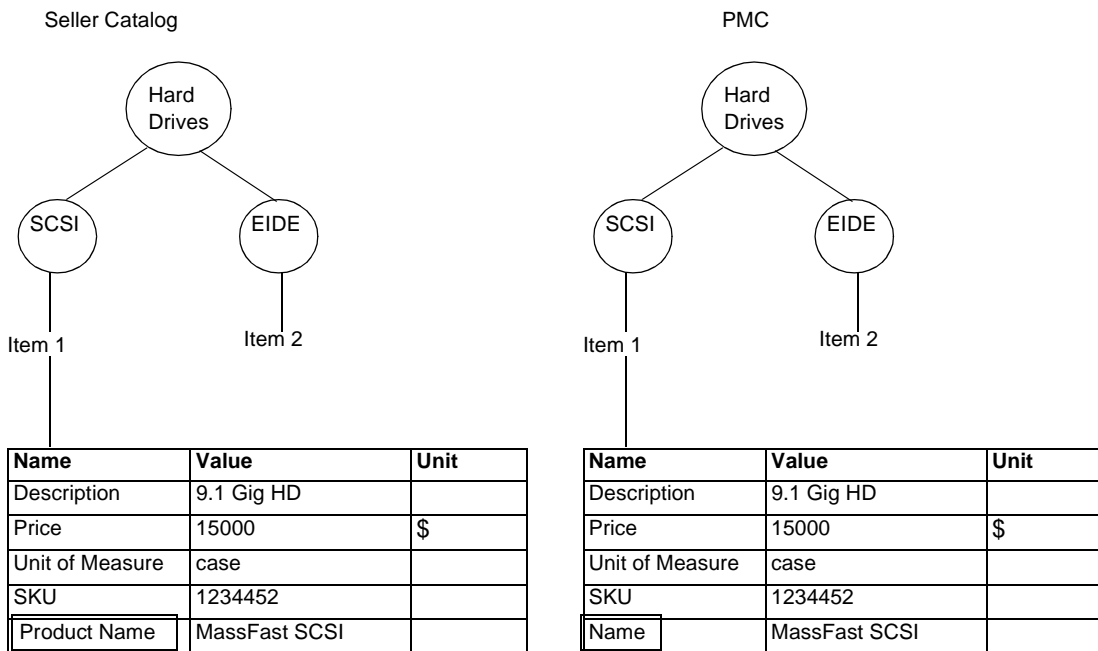
The PMC can impose conventions on attribute names and units. For example, in the PMC, all the attributes associated with the list price of an item are named “Price”. But a seller catalog might not conform to this convention, so the Import utility needs to map the attribute names and units from the seller catalog to those in the PMC. This mapping process is called attribute normalization.

The attribute names in a seller catalog drive attribute name and unit normalization. The Import utility applies the same normalization rules to all the items in a seller catalog. To determine which rule to apply, it finds the attribute name with a corresponding normalization rule.

Attribute Name Normalization

An attribute name in the seller catalog might have a different name in the PMC. For example, an attribute name of “Product Name” in the seller catalog might have an attribute name of “Name” in the PMC, as shown in Figure 1-4.

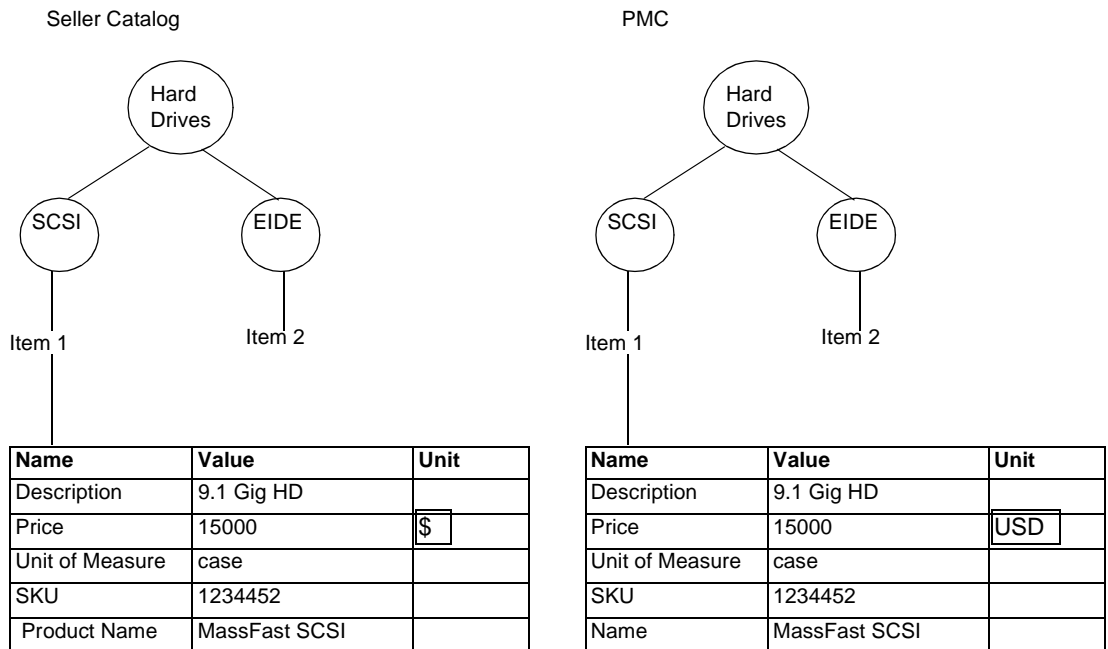
Figure 1-4 Renaming Attribute Names



Renaming Attribute Units

Like attribute names, attribute units in the seller catalog might have a different name in the PMC. An attribute might have the same name in both the seller and PMC, but might have different attribute units. For example, an attribute unit of “\$” in the seller catalog might have an attribute type of “USD” in the PMC, as shown in Figure 1-5.

Figure 1-5 Renaming Attribute Units



Specifying Attribute Types

To relate a certain type to an attribute value based on its attribute name, you can create an XML file that maps the attribute name to the type. For each attribute name, you specify an attribute value and the type to be associated with the value. This file must be correct with respect to the `attribute-types.dtd` file.

Common uses of attribute types are as follows.

- To relate an attribute name of “Price” to the type “currency.” For example,

```
<attribute-types> <attribute-type name="Price" type="currency">  
</attribute-type> </attribute-types>
```

- To specify an attribute value type as `URL`. This means that within the iPlanet Market Maker PMC the attribute value is shown as a link. The URL specified in your catalog must be a path relative to the iPlanet Market Maker directory:

```
<market_maker_install_dir>/docs
```

- To specify an attribute value type of `Integer`. You specify this type to allow for greater than and less than advanced searching capabilities with an iPlanet Market Maker market place. You must have marketmaker administrator privileges to use this capability.

Representing Catalog Information

This chapter describes the conventions the Import utility uses to represent the catalog ontologies and do the mapping.

The following sections are contained in this chapter:

- Representing Catalogs
- XML Representation Issues
- Using the Human-Readable Difference Report Generator (HRDRG)

Chapter 1, “Catalog Concepts” described the concepts that relate to mapping the categories from seller catalogs to the Public Master catalog (PMC). You need to be familiar with these concepts before you can prepare to import your catalog. The next step is to understand the conventions the Import utility uses to represent the catalog ontologies and do the mapping.

Each catalog ontology is a hierarchy of categories and sub-categories. The Import utility uses the extensible markup language (XML) to represent the ontologies of a seller catalog and the PMC.

Representing Catalogs

A seller catalog is represented in two files. The first file contains the unstructured catalog data, which is the standard iPlanet Market Maker catalog input. The second file provides the structural information to interpret the data in the first file.

Table 2-1 describes these files.

Table 2-1 Seller Catalog Files

Seller Catalog File	Format	Content
Character-separated fields (CSF)	Text	Contains the catalog data in which the hierarchy is represented as a table. Each row is a line and each column is field. Each field is separated by a special character, such as a tilde (“~”).
Column structure specification (CSS)	XML	Contains the structural information that the Import utility uses to interpret the catalog data in the CSF file.

The result of interpreting the CSF and CSS files is a load file that the Import utility uses to create the catalog database for an iPlanet Market Maker marketplace. The seller catalog categories are represented by XML elements in the load file. The XML category names are derived from the seller catalog category names. The XML element hierarchy represents the seller catalog hierarchy.

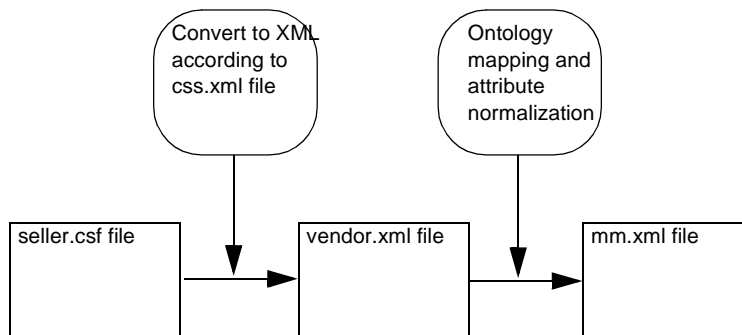
The load file contains the most compact representation of the seller catalog categories. The default load file name is `mm.xml`.

Creating the Seller Catalog and PMC in XML

There are two steps to convert the seller catalog data to an XML file that the Import utility can load into the iPlanet Market Maker catalog database. The first step is to create an XML representation of the seller catalog using the CSF and CSS files. The Import utility creates this XML representation in a file called `vendor.xml`.

The second step is to apply ontology mapping and attribute normalization rules to the `vendor.xml` file. The result of this step is an `mm.xml` file. The Import utility uses UTF8 encoding to process the `vendor.xml` and `mm.xml` files.

Figure 2-1 summarizes the process of creating the XML files.

Figure 2-1 Flow to Create the Seller Catalog and PMC in XML Format

Adding Column Structure to the CSF File

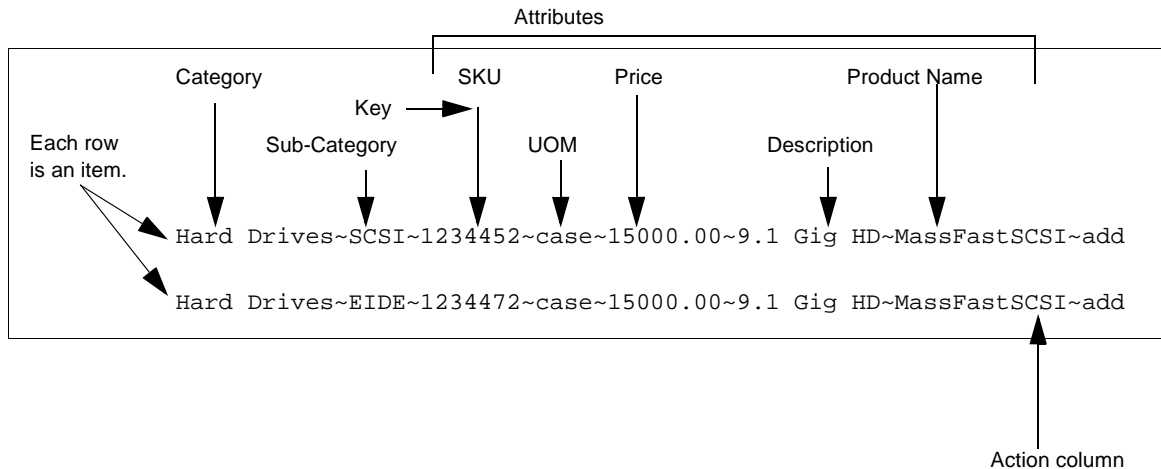
An example of a CSF file is shown in Figure 2-2. Note that this text file does not contain any structural information.

Figure 2-2 CSF File Example

```
Hard Drives~SCSI~1234452~case~15000.00~9.1 Gig HD~MassFastSCSI~add
Hard Drives~EIDE~1234472~case~15000.00~9.1 Gig HD~MassFastSCSI~add
```

The Import utility treats each row in the CSF file as an item. The columns define the categories and attributes. To define the column structure of the CSF file in Figure 2-2, you need to determine which columns represent the categories and attributes. Figure 2-3 shows an example of how you might apply column names to a CSF file.

Figure 2-3 CSF File Annotated with Structural Information



To represent this structure in XML format, it is useful to think of the column structure in terms of a table, such as the example shown in Table 2-2. Note that the “Key” column identifies the SKU attribute as the unique identifier for the items. You must define at least one attribute as a key in the CSS file, but you can use multiple columns to define a key.

Table 2-2 Table Format Representation of the CSF File

Column Name	Column Number	Column Type	Key
Hard Drives	1	Category	
SCSI/EIDE	2	Sub-category	
SKU	3	Attribute	X
Unit of Measurement (UOM)	4	Attribute	
Price	5	Attribute	
Description	6	Attribute	
Product Name	7	Attribute	
Action column	8		

To implement a column structure for the CSF file such as the one shown in Table 2-2, you create a CSS file in XML format. The CSS file must conform to the `ColumnStructureSpecification.dtd` file in the `<imm_install_dir>/catalogtools/dtd` directory.

Figure 2-4 shows an example of a CSS file that describes the column structure of the CSF file shown in Figure 2-2. The columns are numbered from left to right. Each column specification defines a category or attribute, except for column 8 (see the following section).

Adding, Deleting, and Updating Items

Column 8 in Table 2-2 is an action column that defines which action to take when loading this item in the PMC. The result of the “add” action is an item defined in the PMC just as it was defined in the load file. You can also specify “delete” or “update” in an action column to delete an item or update an existing item in the database.

You can specify only one action per column row. If you do not specify an action column, the default action is to add items.

Suppose, for example, that you define an “add” action for item “x” in load file, and item “x” already exists in the PMC with the same key as defined in the load file. As a result of the “add” action, the attributes of item “x” are changed in the PMC as necessary so that they match the attributes of item “x” in the load file. For more information about how the actions work, see Table 2-3.

Table 2-3 Action Results

Action	Result		Comments
	PMC Contains an Item with the Same Key Defined in the Load File	PMC Does Not Contain an Item with the Same Key Defined in File	
add	Item attributes in the PMC are replaced by the item attributes in the load file.	A new item is created in the PMC.	As a result of the “add” action, the PMC contains an item with the identical load file attribute names, numbers, and units. This means that exiting attributes in the PMC can be added, deleted, and updated to match those in the load file.
delete	Item is deleted from the PMC.	An error message	
update	Item attributes in the PMC are updated to match the attributes in the load file.	An error message	Only the attribute values and units changed in the load file are updated in the PMC. Existing attributes in the PMC are not added or deleted.

Figure 2-4 CSS File Example

```

<?xml version="1.0" ?>
<!DOCTYPE column-specifications SYSTEM
"/netscape/server4/imm/catalogtools/dtd/ColumnStructureSpecification.dtd">
<column-specifications>
  <column-specification number="1" >
    <category level="1" />
  </column-specification>
  <column-specification number="2" >
    <category level="2" />
  </column-specification>
  <column-specification number="3" >
    <value key="yes"><name><fixed>SKU</fixed></name></value>
  </column-specification>
  <column-specification number="4" >

```

```
        <value><name><fixed>UOM</fixed></name></value>
    </column-specification>

    <column-specification number="5" >
        <value>
            <name><fixed>Price</fixed></name>
        </value>
    </column-specification>

    <column-specification number="6" >
        <value>
            <name><fixed>Description</fixed></name>
        </value>
    </column-specification>

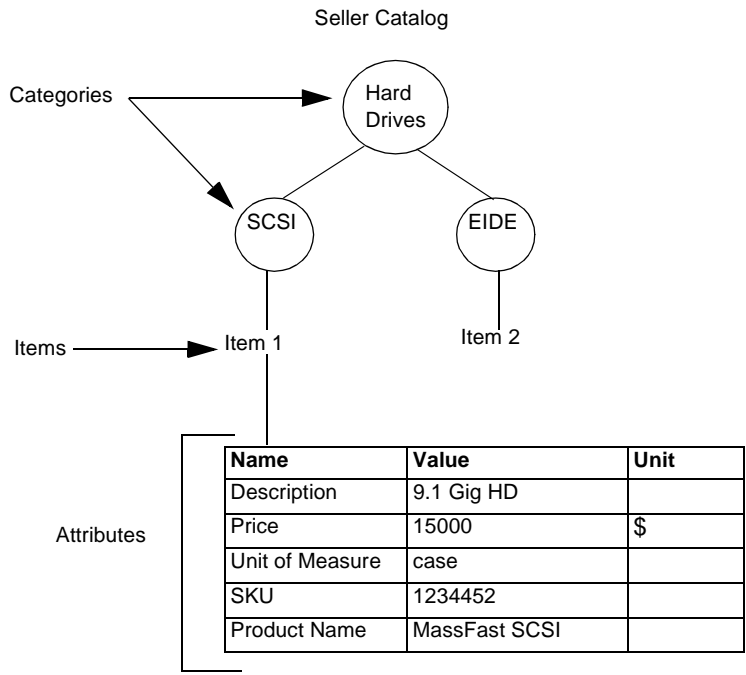
    <column-specification number="7" >
        <value>
            <name><fixed>Product Name</fixed></name>
        </value>
    </column-specification>

    <column-specification number="8" >
        <action />
    </column-specification>

</column-specifications>
```

Figure 2-5 shows a graphical representation of the hierarchy structure implemented by the CSS file in Figure 2-4.

Figure 2-5 Seller Catalog Hierarchy Structure



Using Column References

The CSS file shown in Figure 2-4 specifies fixed columns, which means that the Import utility interprets all the attribute values or units in a column the same way for every row. But there are cases when the attribute names or units in a column are different in each row, such as when there are different currencies for different items. When the attribute names or units are different in each row, you use a column reference.

Suppose, for example, that you wanted to specify different currency types for the items shown in Figure 2-6.

Figure 2-6 CSF File Example with Different Currency Types

1	2	3	4	5	6	7	8
Hard	Drives~SCSI~1234452~case~15000.00~9.1	Gig	HD~MassFastSCSI~USD				
Hard	Drives~EIDE~1234472~case~45000.00~9.1	Gig	HD~MassFastSCSI~JPY				

Assume that in Figure 2-6 column 5 contains the values of the “Price” attribute, and column 8 contains the units (currencies) for this same attribute. See Figure 2-7.

Figure 2-7 CSF File with Structural Information

Categories	Price Attribute Value	Unit
Hard Drives~SCSI~1234452~case~15000.00~9.1	Gig	HD~MassFastSCSI~USD
Hard Drives~EIDE~1234472~case~15000.00~9.1	Gig	HD~MassFastSCSI~JPY

Figure 2-8 shows the section of the CSS file entry to specify that the prices in column 5 have their units (USD for U.S. dollars, JPY for Japanese yen) defined in column 8.

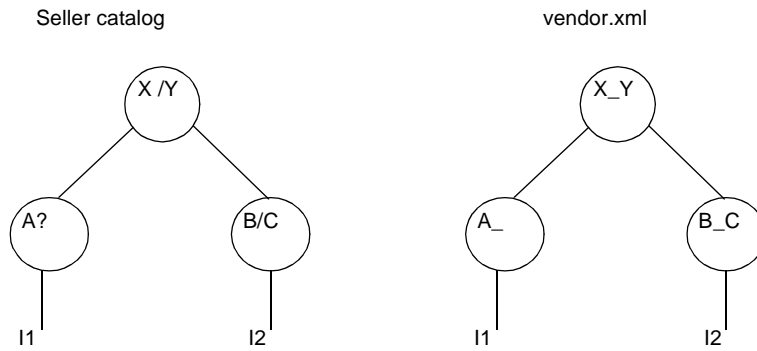
Figure 2-8 Column Reference Section of a CSS File

```
<column-specification number="5" >
  <value>
    <name><fixed>Price<\fixed><\name>
    <unit><column-ref column-number="8"/></unit>
  <value>
</column-specification>
```

Creating the Seller Catalog

The seller catalog created from processing the CSF and CSS file is a `vendor.xml` file. This file maps categories to XML elements. If the category names contain spaces or special characters, those characters are replaced with an underscore (“_”) character. See Figure 2-9.

Figure 2-9 Seller Catalog Represented in the `vendor.xml` File



XML Representation Issues

In the process of representing a seller catalog in XML, the following two issues can arise.

- Sibling categories with unique names in the seller catalog are renamed to have identical names in the `vendor.xml` file. This renaming causes what is known as a *name clash*.
- When two or more categories in the `vendor.xml` file have identical names, the result is what is known as a *path clash*. This situation can occur because two categories in the CSF file have the same name, or as a consequence of renaming categories in the `vendor.xml` file.

To identify clashes, run the Import utility with the `CHECK` option. If your catalog has clashes, this option creates `name_clashes.xml` and `path_clashes.xml` report files that contain instructions for resolving the clashes. In the report files, the names with clash problems are identified with a `<tag-name>` tag.

Detecting Clashes

The following two types of clashes can occur.

- Name clashes

Occurs when sibling categories are renamed to the same name. For a list of special characters that cause this renaming, see the following URL.

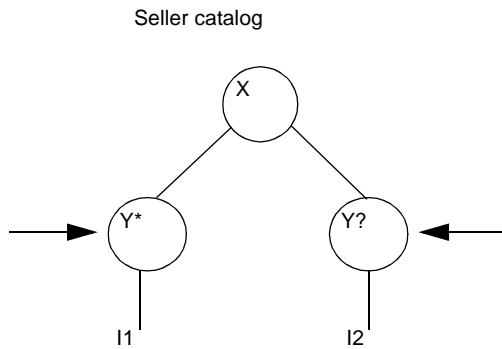
<http://www.w3.org/TR/1998/REC-xml-19980210.html#NT-Name>
- Path clashes

Occurs when categories have the same name, but their “child” categories have different names.

Resolving Name Clashes

If there are name clashes in your hierarchy, you need to resolve them to create the PMC.

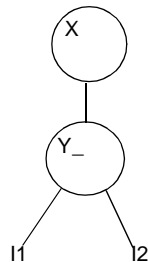
Figure 2-10 shows an example of a hierarchy with a name clash, where the names “Y*” and “Y?” create a name clash.

Figure 2-10 Name Clash

By default, the “Y*” and “Y?” categories are renamed to the same category in the `vendor.xml` file. See Figure 2-11 and Figure 2-12.

Figure 2-11 Result of Name Clash

`vendor.xml`

**Figure 2-12** The `vendor.xml` File with a Name Clash

```

<?xml version="1.0" encoding="UTF-8"?>
<root>
  <X vortex-type="category" name="X">
    <Y_ vortex-type="category" name="Y*">
      <Item vortex-type="item" action="add">
        <Attribute vortex-type="attribute" value="1" name="SKU" key="yes" />
      </Item>
    </Y_>
  </X>
</root>

```

```
        </Item>
    </Y_>
</X>
<X vortex-type="category" name="X">
    <Y_ vortex-type="category" name="Y?">
        <Item vortex-type="item" action="add">
            <Attribute vortex-type="attribute" value="2" name="SKU" key="yes" />
        </Item>
    </Y_>
</X>
</root>
```

If name clashes exist, the Import utility creates a report file called `name_clashes.xml`. This file contains an XML description of the name clashes that you can use as a template file to resolve the clashes. Figure 2-13 shows an example of such a file, edited to perform the renaming shown in Figure 2-14. The `<tag-name>` tags that define the renamed names are in **bold** in Figure 2-13.

Figure 2-13 Edited Name Clash File

```

<?xml version="1.0" encoding="UTF-8"?>
  <names>
    <name>
      <category>Y*</category>
      <tag-name>Y_star</tag-name>
    </name>
    <name>
      <category>Y?</category>
      <tag-name>Y_q</tag-name>
    </name>
  </names>

```

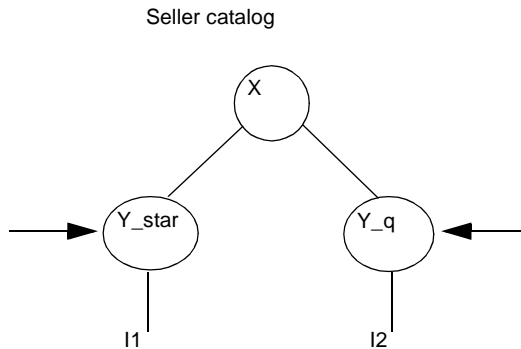
Figure 2-14 Name Clash Resolution

Figure 2-15 shows the `vendor.xml` file with resolved name clashes.

Figure 2-15 The `vendor.xml` File with a Resolved Name Clash

```

<?xml version="1.0" encoding="UTF-8"?>
<root>
  <X vortex-type="category" name="X">
    <Y_star vortex-type="category" name="Y*">
      <Item vortex-type="item" action="add">
        <Attribute vortex-type="attribute" value="1" name="SKU" key="yes" />
      </Item>
    </Y_star>
  </X>
  <X vortex-type="category" name="X">
    <Y_q vortex-type="category" name="Y?">

```

```

<Item vortex-type="item" action="add">
  <Attribute vortex-type="attribute" value="2" name="SKU" key="yes" />
</Item>
</Y_q>
</X>
</root>

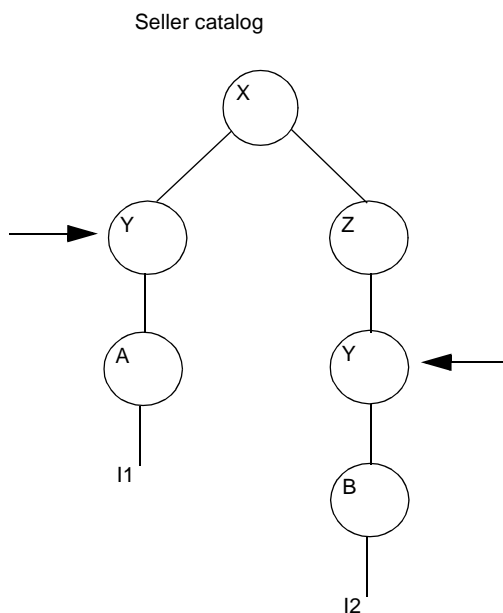
```

Resolving Path Clashes

If there are path clashes in your hierarchy, you need to resolve them to create an `mm.xml` file you can load into the catalog database. To find out if there are path clashes, always specify the `CHECK` option the first time you run the Import utility.

Figure 2-16 shows an example of a path clash.

Figure 2-16 Path Clash



In Figure 2-16, the two “parent” categories named “Y” have different sets of “child” categories. Under XML validation rules, the two “Y” categories contain the “child” categories “A” and “B”. As a result of validation, the “Y” categories are considered to be the same category represented as <Y> in the `vendor.xml` file shown in Figure 2-12.

Figure 2-17 The `vendor.xml` File with a Path Clash

```
<?xml version="1.0" encoding="UTF-8"?>
<root>
  <X vortex-type="category" name="X">
    <Y vortex-type="category" name="Y/">
      <A vortex-type="category" name="A">
        <Item vortex-type="item" action="add">
          <Attribute vortex-type="attribute" value=" 1" name="SKU" key="yes" />
        </Item>
      </A>
    </Y>
  </X>
  <X vortex-type="category" name="X">
    <Z vortex-type="category" name="Z">
      <Y vortex-type="category" name="Y?">
        <B vortex-type="category" name="B">
          <Item vortex-type="item" action="add">
            <Attribute vortex-type="attribute" value="2" name="SKU" key="yes" />
          </Item>
        </B>
      </Y>
    </Z>
  </X>
</root>
```

The category hierarchy in Figure 2-17 is defined too broadly. To apply strict XML validation, the “Y” categories need to be renamed so that they are unique. For example, see Figure 2-19.

If path clashes exist, the Import utility creates a report file called `path_clashes.xml`. This file contains an XML description of the path clashes that you can use as a template file to resolve the clashes. Figure 2-18 shows an example of such a file, edited to perform the renaming shown in Figure 2-19. The `<tag-name>` tags that define the renamed names are in **bold** in Figure 2-18.

Figure 2-18 Edited Path Clash File

```

<?xml version="1.0" encoding="UTF-8"?>
<paths>
  <path-clash clashing-tag="Y">
    <path-ref>
      <path>/root/X/Y</path>
      <tag-name>Y_slash</tag-name>
    </path-ref>
    <path-ref>
      <path>/root/X/Z/Y</path>
      <tag-name>Y_star</tag-name>
    </path-ref>
  </path-clash>
</paths>

```

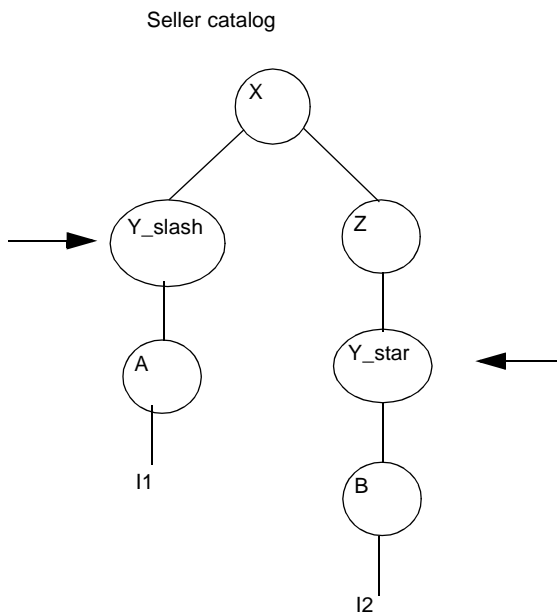
Figure 2-19 Path Clash Resolution

Figure 2-20 shows the `vendor.xml` file with resolved path clashes.

Figure 2-20 The vendor.xml file with Resolved Path Clashes

```
<?xml version="1.0" encoding="UTF-8"?>
<root>
  <X vortex-type="category" name="X">
    <Y_slash vortex-type="category" name="Y/">
      <A vortex-type="category" name="A">
        <Item vortex-type="item" action="add">
          <Attribute vortex-type="attribute" value=" 1" name="SKU" key="yes" />
        </Item>
      </A>
    </Y_slash>
  </X>
  <X vortex-type="category" name="X">
    <Z vortex-type="category" name="Z">
      <Y_star vortex-type="category" name="Y?">
        <B vortex-type="category" name="B">
          <Item vortex-type="item" action="add">
            <Attribute vortex-type="attribute" value="2" name="SKU" key="yes" />
          </Item>
        </B>
      </Y_star>
    </Z>
  </X>
</root>
```


Defining the Ontology Mapping

The “Mapping Catalog Hierarchies” section in Chapter 1 describes the concept of mapping items from a seller catalog to the PMC. Seller ontologies often vary from seller to seller, but the PMC has only one ontology. The PMC ontology is likely to be different from a seller catalog. So a mapping mechanism is necessary to ensure that a seller catalog ontology conforms to the PMC ontology. The instructions to do this mapping are described in an ontology mapping description file named `omd.xml`.

When the ontology in the PMC is the same as that of a seller catalog, the Import utility can generate the `omd.xml` file automatically. The Import utility stills needs to do the mapping in this case because the load file is a compact version of the catalog input data, and the load program uses this compact version to optimize processing.

When the ontology of the PMC is different from that of a seller catalog, you must create the file `omd.xml` manually.

Changing the Ontology Mapping

The Import utility creates an ontology mapping file, `omd.xml`, that assumes the `mm.xml` and `vendor.xml` ontologies are identical. Figure 2-21 shows an example of a `vendor.xml` ontology.

Figure 2-21 A `vendor.xml` Hierarchy Example

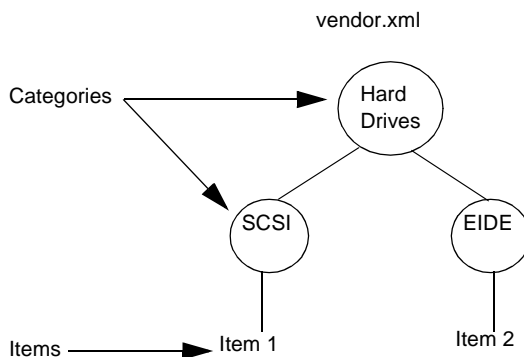


Figure 2-22 shows the resulting `omd.xml` file that the Import utility creates. The categories defined in the `omd.xml` file are the categories that the Import utility creates in the `mm.xml` file. The `<path>` tags tells the Import utility where in the `vendor.xml` file to find the items to be contained in a category in the `mm.xml` file.

In Figure 2-22, for example, there is a category named “SCSI”:

```
<SCSI vortex-type="category" name="SCSI">
```

In this example, the “SCSI” category in the `vendor.xml` file also appears in the `mm.xml` file. The `<path>` tag defines the path to locate the items (in this case only one item) in the `vendor.xml` file to be contained in the “SCSI” category in the `mm.xml` file. The “Hard_Drives” category is the root category.

```
<path>/root/Hard_Drives/SCSI/Item</path>
```

Figure 2-22 Resulting `omd.xml` Ontology Mapping File

```
<?xml version="1.0" encoding="UTF-8"?>

<root version="1.0">

  <Hard_Drives vortex-type="category" name="Hard Drives">

    <path>/root/Hard_Drives/Item</path>

    <SCSI vortex-type="category" name="SCSI">

      <path>/root/Hard_Drives/SCSI/Item</path>

    </SCSI>

    <EIDE vortex-type="category" name="EIDE">

      <path>/root/Hard_Drives/EIDE/Item</path>
```

```

    </EIDE>

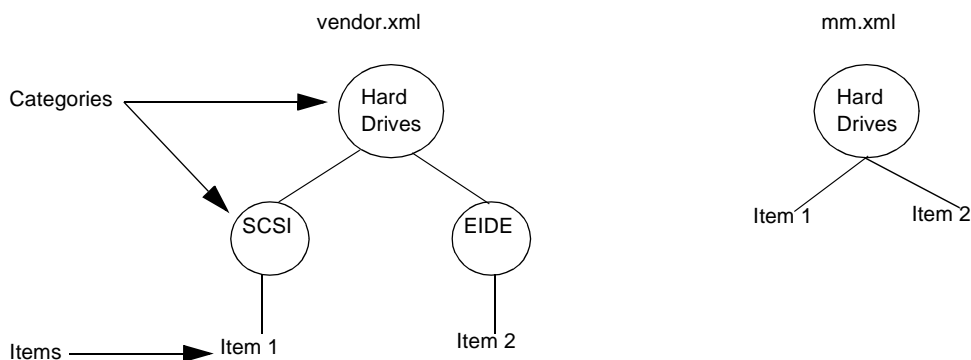
  </Hard_Drives>

</root>

```

When the ontologies in the `vendor.xml` and `mm.xml` files are different, you need to edit the `omd.xml` file. Figure 2-23 illustrates such a case.

Figure 2-23 Different `vendor.xml` and `mm.xml` Hierarchies



In Figure 2-23, the items in `vendor.xml` are mapped to a different ontology in `mm.xml` in which there is only one category. This means that the “SCSI” and “EIDE” categories are not in the `mm.xml` file. The items under these two categories need to be mapped to the “Hard Drives” category. To do this mapping, you need to edit the `omd.xml` file.

Figure 2-24 shows the edited version of the `omd.xml` file needed to map the items in the `vendor.xml` file to the `mm.xml` file. Note that there is only a “Hard Drives” category. The `<path>` tags tell the Import utility to take the items under the “SCSI” and “EIDE” categories in the `vendor.xml` file and map them to the “Hard Drives” category in the `mm.xml` file.

```

<path>/root/Hard_Drives/SCSI/Item/</path>

<path>/root/Hard_Drives/EIDE/Item/</path>

```

Figure 2-24 PMC omd.xml Ontology Mapping File

```

<?xml version="1.0" encoding="UTF--8"?>

<root version="1.0"

  <Hard_Drives vortex-type="category" name="Hard Drives">

    <path>/root/Hard_Drives/SCSI/Item/</path>

    <path>/root/Hard_Drives/EIDE/Item/</path>

  </Hard_Drives>

</root>

```

As a result of ontology mapping, the `mm.xml` file defines the hierarchy of category, item, and attribute elements. Figure 2-25 shows an example of an `mm.xml` file. The elements are highlighted in bold.

Figure 2-25 Sample `mm.xml` File

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE vortex-data-load SYSTEM
"file:///usr/netscape/server4/imm/labs/mm.dtd">
<vortex-data-load version="1.0">
  <Hard_Drives name="Hard Drives" vortex-type="category">
    <Item vortex-type="item" action="add">
      <Attribute vortex-type="attribute" value="1234452" name="SKU"
key="yes"/>
      <Attribute vortex-type="attribute" value="case" name="UOM"
key="no"/>
      <Attribute vortex-type="attribute" value="15000.00"
dataType="currency" name="Price" key="no"/>
      <Attribute vortex-type="attribute" value="9.1 Gig HD"
name="Description" key="no"/>
      <Attribute vortex-type="attribute" value="MassFastSCSI"
name="Name" key="no"/>
    </Item>
  <Item vortex-type="item" action="add">

```

```
        <Attribute vortex-type="attribute" value="1234472" name="SKU"
key="yes"/>
        <Attribute vortex-type="attribute" value="case" name="UOM"
key="no"/>
        <Attribute vortex-type="attribute" value="15000.00"
dataType="currency" name="Price" key="no"/>
        <Attribute vortex-type="attribute" value="9.1 Gig HD"
name="Description" key="no"/>
        <Attribute vortex-type="attribute" value="MassFastSCSI"
name="Name" key="no"/>
    </Item>
</Hard_Drives>
</vortex-data-load>
```

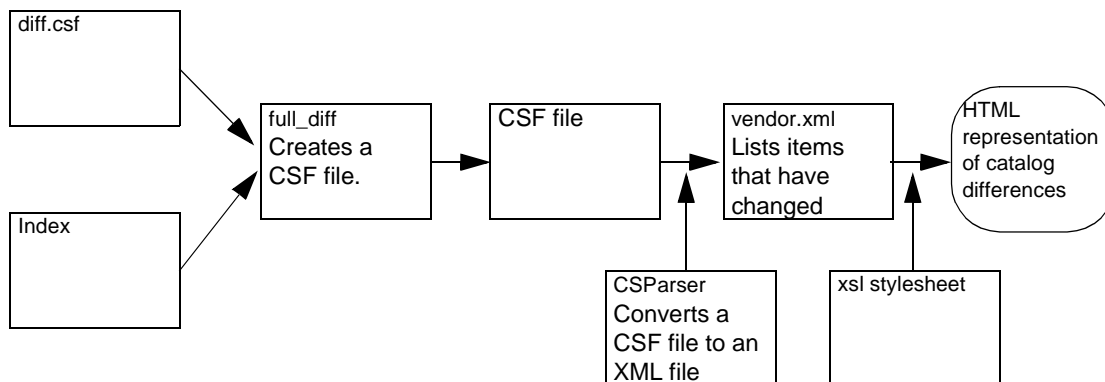
Using the Human-Readable Difference Report Generator (HRDRG)

The human-readable difference report generator (HRDRG) provides a set of utilities, which can be customized by iPlanet Professional Services, to build a system that generates HTML reports showing the minimum set of meaningful differences between two CSF files. Suppose, for example, that a seller changes an existing catalog and submits the changes to a marketmaker. The marketmaker might want to review the changes in relation to the existing catalog to find out which items have been added, replaced, updated, or deleted. The HRDRG scripts provide a framework to allow the changes to be displayed in an HTML browser. These scripts are in the following directory.

```
<imm_install_dir>/catalogtools/bin
```

The `hrdrng` script runs the `full_diff` utility and the `CSFParser` utility. See Figure 2-26. The `full_diff` utility produces an annotated CSF file that describes the changes to the existing CSF file. The `CSFParser` utility converts the annotated CSF file to XML format. For more information about these utilities, see the “Inputs to the HRDRG Flow” section.

Figure 2-26 HRDRG Flow



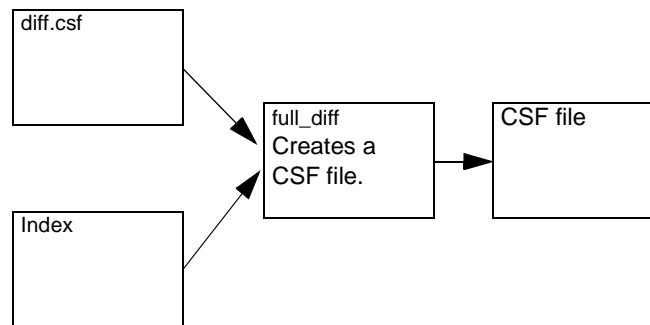
Inputs to the HRDRG Flow

There are two inputs to the HRDRG flow: the `diff.csf` file and the index file. The `diff.csf` file contains the differences between an existing CSF file and a new CSF file. There are a number of ways to create the `diff.csf` file. One way is to keep the changes made in the existing `.csf` file in a separate `diff.csf` file. You can also provide an existing and new index as input to the `difference` utility to create the `diff.csf` file.

The index file is the indexed representation of the existing `.csf` file.

The `full_diff` utility reads the `diff.csf` file and the index file and creates a `.csf` file similar to the `.csf` file that the `difference` utility creates. Figure 2-27 shows the input to the HRDRG flow.

Figure 2-27 HRDRG Input



Unlike the `.csf` file that the `difference` utility creates, the `.csf` file output by the `full_diff` utility lists the items that have been added, replaced, updated, or deleted. For each item changed in the existing catalog, the `full_diff` utility adds a corresponding new line that notes the type of change. For example, see Figure 2-28.

Figure 2-28 A `full_diff` Utility `.csf` Example

```

Hard Drives~SCSI~1234472~case~15000.00~11 Gig HD~MassFastSCSI~update
Hard Drives~SCSI~1234472~case~15000.00~9.1 Gig HD~MassFastSCSI~updateold
  
```

Table 2-4 shows the types of changes that can appear in the `.csf` file generated by the `full_diff` utility.

Table 2-4 The `full_diff` Utility Keywords

Keyword	Description
<code>add</code>	Specifies that a new item is added to the existing <code>.csf</code> file.
<code>delete</code>	Specifies that a new item is deleted from the existing <code>.csf</code> file.
<code>deleteincorrect</code>	Specifies that a new item is specified to be deleted but is not in the existing <code>.csf</code> file.
<code>update</code>	Specifies that a new item is updated the existing <code>.csf</code> file.
<code>updateold</code>	Specifies the existing item to be updated in the existing <code>.csf</code> file.
<code>updateincorrect</code>	Specifies that a new item is specified to be updated but is not in the existing <code>.csf</code> file.
<code>replace</code>	Specifies that a new item is replaced in the existing <code>.csf</code> file.
<code>replaceold</code>	Specifies the existing item to be replaced in the existing <code>.csf</code> file.
<code>replaceincorrect</code>	Specifies that a new item is specified to be replaced but is not in the existing <code>.csf</code> file.

Output of the HRDRG Flow

From the annotated CSF file created by the `full_diff` utility, the `hrdrng` utility creates a `vendor.xml` file with the same changes propagated from the annotated CSF file described in Table 2-4. Figure 2-29 shows a `vendor.xml` file example.

Figure 2-29 A `vendor.xml` File with Changes Noted

```
<?xml version="1.0" encoding="UTF-8"?>
<root>
  <X vortex-type="category" name="X">
    <Y vortex-type="category" name="Y/">
      <A vortex-type="category" name="A">
        <Item vortex-type="item" action="update">
          <Attribute vortex-type="attribute" value=" 1" name="SKU" key="yes"
        />
      </Item>
    </A>
  </X>
</root>
```



```

    </Y>
  </X>
<root>
  <X vortex-type="category" name="X">
    <Y vortex-type="category" name="Y/">
      <A vortex-type="category" name="A">
        <Item vortex-type="item" action="updateold">
          <Attribute vortex-type="attribute" value=" 1" name="SKU" key="yes"
/>
        </Item>
      </A>
    </Y>
  </X>

```

After the `hrdrng` script creates the file `vendor.xml` file, you can customize an xsl stylesheet to format and display the information in the `vendor.xml` file in an HTML browser. Example of the stylesheets are in the following directory.

```
<imm_install_dir>/catalogtools/xsl
```

Using the Human-Readable Difference Report Generator (HRDRG)

Running the Import Utility

This chapter describes how to set up and run the Import utility to create the load file and load it in the PMC.

The following sections are contained in this chapter:

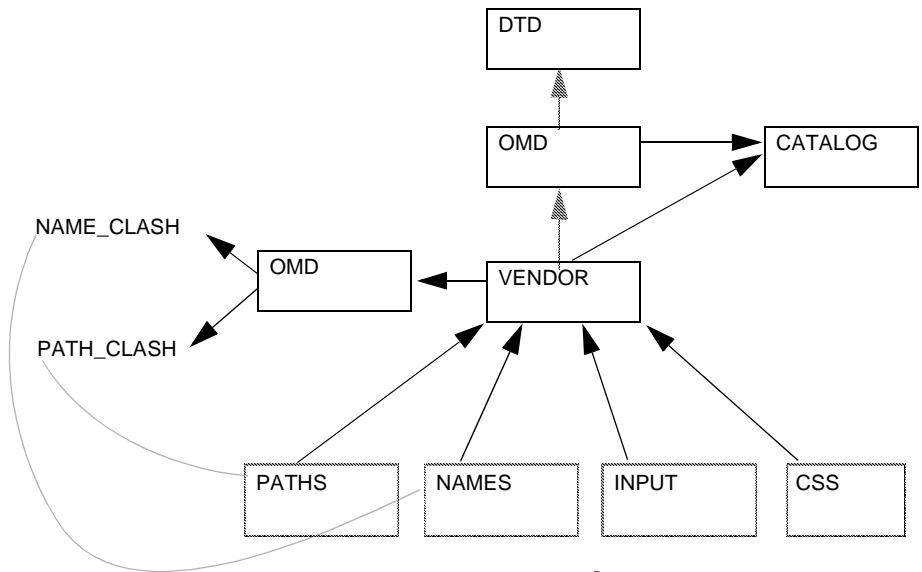
- Dependency Trees
- Setting Up Your Environment
- Preparing To Run the Import Utility
- An Example of Running the Import Utility
- Loading the Load File in the PMC
- Exporting Catalogs

Dependency Trees

After you create CSF and CSS files, you are ready to specify the information that the Import utility needs to create the load file to load in the Public Master catalog (PMC).

The Import utility uses the `make` command to create the load file. The `make` command inputs and the output targets are shown as dependency trees in Figure 3-1 and Figure 3-2. Figure 3-1 shows the dependency tree to create a seller ontology when there is not an exiting PMC ontology. Figure 3-2 shows the dependency tree to update an existing PMC ontology.

Figure 3-1 Dependency Tree for Creating a Seller Ontology with No Existing PMC Ontology



Legend

DTD - mm.dtd

PATHS - Paths for Items

OMD - omd.xml

NAMES - Attribute names


CATALOG - mm.xml

INPUT - CSF file

VENDOR - vendor.xml


CSS - CSS file

NAME_CLASH - Name clash report

 Resolve name and path clashes

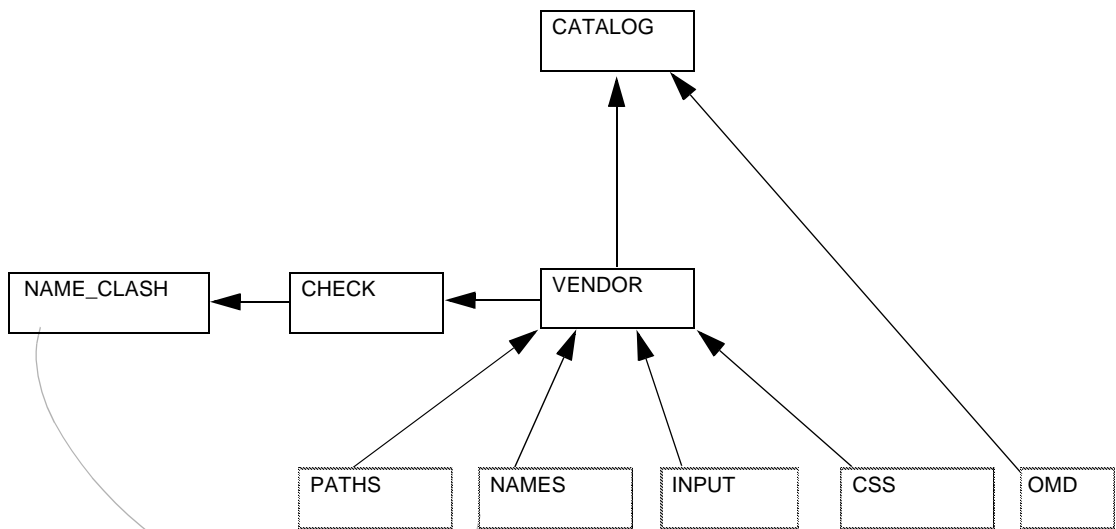
PATH_CLASH - Path clash Report

 Optional steps

 Output targets

 Inputs

Figure 3-2 Dependency Tree for Updating an Existing PMC Ontology




Legend

CATALOG - mm.xml

VENDOR - vendor.xml

CHECK - Check for path Clashes

NAME_CLASH - Name clash report

 Output Target


PATHS - Paths for items

NAMES - Attribute names

INPUT - CSF File

CSS - CSS File

OMD - omd.xml

 Resolve name clashes

 Inputs

Setting Up Your Environment

The Import utility uses the `make` command. To be sure that the `make` command runs properly, check the following items.

- Be sure that the `make` command is in your command path. This command is in the `/usr/ccs/bin` directory.
- Source either the `mm.kshrc` or `mm.cshrc` files, depending on whether you run the Import utility from a Korn shell or C-shell. These files are in the iPlanet Market Maker installation directory.

If you get a `word too long` error message when you source the `mm.cshrc` file, you need to create a symbolic link from your root directory to the iPlanet Market Maker installation directory (usually `imm`). To create this link, go to your root directory and type the command `ln -s <install_dir> <link_name>`, where `<install_dir>` is the path to your iPlanet Market Maker installation directory and `<link_name>` is the name of the symbolic link. After you create this link, replace all instances of the iPlanet Market Maker installation directory name in your `mm.cshrc` file with the directory name defined by the symbolic link.

Preparing To Run the Import Utility

Before you can create the load file, you need to specify the input file names. To specify this information, you edit a file named `makefile` in a text editor. To create the `makefile` file, you must start with the `filenames.mk` file in the following directory.

```
<imm_install_dir>/catalogtools/etc/filenames.mk
```

Copy the `filenames.mk` file to your working directory. Then rename `filenames.mk` to `makefile`. This section describes the information you need to provide in the `makefile` file. This file contains all the information the Import utility needs to run properly, so make sure that you enter this information completely and correctly. Figure 3-3 shows a sample `makefile` file.

The “#” character in a `makefile` file is a comment. The Import utility ignores all the lines that begin with this character. Be sure to remove the “#” character when you specify an entry in the `makefile` file.

The `makefile` file contains the following sections.

- Output targets
- Required source files
- Optional files
- Ultimate targets
- Ontology Mapping
- Optional variables
- Error reports
- Command arguments
- Internal targets
- Directory locations

Specifying Output Targets

This section of the `makefile` contains the output targets that the Import utility uses to create the load file. You do not need to edit this section of the `makefile` file. You specify the output targets on the Import utility command line. See Table 3-1.

Table 3-1 Output Targets

Target	Description
ALL	Creates or updates the following files: <ul style="list-style-type: none"> • Ontology definition (default is <code>omd.xml</code>) • Document type definition (default is <code>mm.dtd</code>) • PMC (default is <code>mm.xml</code>)
CATALOG	Creates or updates the PMC (default is <code>mm.xml</code>).
ONTOLOGY_MAP	Creates or updates the ontology definition file (default is <code>omd.xml</code>).
DTD	Creates or updates the document type definition file (default is <code>mm.dtd</code>).
CHECK	Checks for both name and path clash problems. The default report files names are <code>name_clashes.xml</code> and <code>path_clashes.xml</code> , respectively. These reports tell you what to do if your catalog has name or path clashes.

Table 3-1 Output Targets

CLEAN	Creates or updates the name and path clash reports.
CLOBBER	Removes all files except the source input files, column structure specification, ontology map definition, and the name and path files. You use this output target to remove the intermediate files from a previous run of the Import utility.

Specifying the Required Source File Names

Table 3-2 shows the required source input file names you provide in the `makefile` file.

Table 3-2 Required Source Input File Names

Required Input	Default File Name or Path	Description
PWD=		For Windows compatibility only. Not required for UNIX systems.
INPUT=input.csf	input.csf	Specifies the name of your CSF file. You can use a relative path name.
CSS=\$(PWD)/css.xml	css.xml	Specifies the name of your CSS file. You must use an absolute path name.
IMPORT_ROOT=\$(IMM_HOME)/catalogtools	\$(IMM_HOME)/catalogtools	Specifies the base location of the iPlanet Market Maker Import utility files. The default path is specified when you install the iPlanet Market Maker software. You must use an absolute path.

Specifying the Optional File Names

Table 3-3 shows the optional source input file names you can provide in the `makefile` file.

Table 3-3 Optional Input File Names

Option	Description
ENCODING=	Encoding of the input CSF file. For information about encoding rules, see the following URL. http://java.sun.com/products/jdk/1.2/docs/guide/internat/encoding.doc.html
NAMES=	Name of the file you specify to resolve name clashes. For an example of this file, see the “Resolving Name Clashes” section in Chapter 3.
PATHS=	Name of the file you specify to resolve path clashes. For an example of this file, see the “Resolving Path Clashes” section in Chapter 3.
UNITS=	Name of the XML file you use to perform attribute unit normalization. For more information about unit normalization, see the “Renaming Attribute Units” section in Chapter 2.
NAME-MAP=	Name of the XML file you use to perform attribute name normalization. For details, see the “Attribute Name Normalization” section in Chapter 2.
ATTRIBUTE-TYPES=	Name of the XML file you use to specify attribute types. For details, see the “Specifying Attribute Types” section in Chapter 2.

Specifying the Ultimate Targets

The ultimate targets are the result of running the Import utility. The `CLEAN` option does not remove these targets. You can use the default targets to run the Import utility, or you can change them. See Table 3-4.

Table 3-4 Ultimate Target File Names

Target	Default File Name	Description
DTD=	mm.dtd	Name of the DTD file for the load file. You must specify an absolute path. The default is <code>\$(PWD)/mm.dtd</code> .
OUTPUT=	mm.xml	Name of the load file to load into the iPlanet Market Maker catalog database to create or update the PMC. The default file name is <code>mm.xml</code> .

Specifying the Ontology Mapping File

The Import utility can generate the ontology mapping file automatically, but you might need to modify this file depending on the ontologies of your seller catalog and load file. The default file name is `omd.xml`. For details about ontology mapping, see the “Defining the Ontology Mapping” section in Chapter 3.

Specifying Optional Variables

The optional variables define the characteristics of your CSF file. You can change the defaults.

Table 3-5 Optional Variable Names

Variable	Description
SEPARATOR=	Name of the character that separates the text fields in the CSF. The default is the tilde (“~”) character.
SKIP=	Specifies the number of header lines to skip (ignore) in the CSF file. The default is 0.

Specifying Error Reports

The error reports tell you if your seller catalog has name or path clash problems. You can change the default report file names.

Table 3-6 Error Report File Names

Report	Default File Name	Description
NAME_CLASH_REPORT=	name_clashes.xml	Specifies the file name of the name clash report that the Import utility generates when you use the CHECK option. For an example of this file, see the “Resolving Name Clashes” section in Chapter 3.
PATH_CLASH_REPORT=	path_clashes.xml	Specifies the file name of the path clash report that the Import utility generates when you use the CHECK option. For an example of this file, see the “Resolving Path Clashes” section in Chapter 3.

Specifying Command Arguments

The command argument specify whether or not to suppress the output messages from the programs that the Import utility uses. See Table 3-8.

Table 3-7 Command Arguments

Target	Description
# QUIET=Q	Uncomment this argument if you want to suppress the output messages from the XSLT process that the Import utility uses. The default is not to suppress the output messages.
# .SILENT	Uncomment this argument if you want to suppress the output messages from the make command that runs the Import utility uses. The default is not to suppress the output messages.

Specifying Internal Targets

The internal targets are the internal files that the Import utility uses. You can change the default names. See Table 3-8.

Table 3-8 Internal Targets

Target	Description
VENDOR=	Specifies the seller catalog file name that the Import utility maps to the PMC. The default file name is <code>vendor.xml</code> .
CM=	Specifies the catalog map file name. The default file name is <code>cm.xml</code> .

Specifying Directory Locations

The Import utility accesses these directories for XSLT and XML processing. You should not change these directory paths unless you are creating a customized environment to run the Import utility.

Table 3-9 Directory Locations

Directory	Location
XSLT_DIR=	Specifies the path of the XSLT directory. The default path is <code>\$(IMPORT_ROOT)/xsl</code> .
XML_DIR=	Specifies the path of the XML directory. The default path is <code>\$(IMPORT_ROOT)/xml</code> .
DTD_DIR	Specifies the path to the directory that contains the standard Import utility <code>.dtd</code> files. The default path is <code>\$(IMPORT_ROOT)/dtd</code> .

Figure 3-3 shows a sample `makefile` file.

Figure 3-3 Sample makefile File

```

#####
# Targets
# ALL - this will update the ontology map, the DTD and the catalog
# from the input.
# CATALOG - this will use the current ontology map to make a new
# catalog. The ontology map will not be updated.
# ONTOLOGY_MAP - this will update the ontology map from the input.
# DTD - this will update the DTD from the current ontology map. The
# ontology map will not be updated
# CHECK - this will check for both name clashes and validity problems
# document
# CLEAN - a phony target that cleans up report files
# CLOBBER - a phony target that will remove everything but the source files.
#####
#
#####
# SOURCE FILES
#####
#
### REQUIRED
# PWD is the name of the directory that this makefile is located in
#PWD =
# INPUT is the name of the CSF file - it can be a relative name
#INPUT=input.csf
# CSS is the name of the Column Structure Specification file. Must be
# an absolute path name
#CSS=$(PWD)/css.xml
#
# IMPORT_ROOT - the base location of the transform system. Must be an
# absolute file name.
#IMPORT_ROOT=$(IMM_HOME)/catalogtools
#include $(IMPORT_ROOT)/defaults.mk
### OPTIONAL
# ENCODING is the name of the character encoding of the input
# file. It defaults to whatever the platform default is. This must be
# an encoding understood by Java 1.2.
# See
# http://java.sun.com/products/jdk/1.2/docs/guide/internat/encoding.doc.html
# for a complete list
#ENCODING=
#
# NAMES is the name of the file containing a list of name
# aliases. It must be an absolute path. This file must be valid wrt
# $(DTD_DIR)/names.dtd. Optional. Default is no file.
#NAMES=
#
# PATHS is the name of the file which will translate a path to
# a category name. This must be an absolute path. This file must be
# valid wrt $(DTD)/paths.dtd. Optional. Default is no file
#PATHS=
#
# UNITS is the name of the file containing vendor-specific unit
# aliases. This must be an absolute path. This file must be valid wrt
# $(DTD)/units.dtd. Optional. Default is no file.

```

```
#####
#UNITS=
#
# NAME-MAP is the name of the xml file containing a mapping between
# supplier's attribute names and those to be used by the market
# maker. This file must be valid wrt
# $(DTD)/attribute-renaming.dtd. This must be an absolute
# path. Optional. Default is no file
#NAME-MAP=
#
# ATTRIBUTE-TYPES is the name of the file containing a mapping between
# attribute names and attribute types. This must be an absolute
# path. This file must be valid wrt
# $(DTD)/attribute-types.dtd. Optional. Default is no file
#ATTRIBUTE-TYPES=$(PWD)/attr-type.xml
#
#####
# ULTIMATE TARGETS - these are never removed by the CLEAN operation
# DTD - the name of the dtd file. Optional. Absolute path. Defaults to
# $(PWD)/mm.dtd
#DTD=
#
# OUTPUT is the name of the market maker catalog
#file. Optional. Defaults to mm.xml
#OUTPUT=
#
#####
# Intermediate (generated) files
#####
# OMD is the name of the Ontology Mapping Description. This might be
# modified by hand to support a particular mapping. Optional. Defaults
# to omd.xml
#OMD=
#
#####
# OPTIONAL variables
#####
#
# SEPARATOR - this is the character that separates fields in the
# $(INPUT) file - defaults to '~'
#SEPARATOR=
#
# SKIP - this is the number of header lines to be skipped in the
# $(INPUT) file - defaults to 0
#SKIP=
#
#
#####
# Error reports
#####
# Error reports can be generated from the "CHECK" target.
#
# NAME_CLASH_REPORT is the name of the file which will contain the name check
# output. Optional. Defaults to "name_clashes.xml"
# NAME_CLASH_REPORT=
#
# PATH_CLASH_REPORT is the name of the file which will contain the path
# problem report. Optional. Defaults to "path_clashes.xml"
```

```
#####
# PATH_CLASH_REPORT=
#
#####
# Command arguments
#####
# Uncomment the following if you don't want any "noise" from the XSLT
# processor. Defaults to no value (i.e. noisy)
# QUIET=-Q
#
# Uncomment the following if you don't want any "noise" from
# make. Defaults to not defined (i.e. noisy)
# .SILENT :
#
#####
# INTERNAL TARGETS - these files are used internally and you can
# change them as you like
#####
# VENDOR is the name of the vendor vertical file. Optional. Defaults
# to "vendor.xml"
#VENDOR=
# CM is the name of the catalog map file. Optional. Defaults to cm.xml
#CM=
#
#####
# DIRECTORY LOCATIONS
#
# These variables define which directories various utilities are
# located in. Only change them if you know what you're doing
#####
#
# XSLT_DIR - the location of the XSLT directory. Optional. Defaults to
# $(IMPORT_ROOT)/xsl
# XSLT_DIR=
#
# XML_DIR - the location of the XML directory. Optional. Defaults to
# $(IMPORT_ROOT)/xml
# XML_DIR=
#
# DTD_DIR - the location of the directories holding the "standard"
# dtds. Optional. Defaults to $(IMPORT_ROOT)/dtd
# DTD_DIR=
#
#
include $(IMPORT_ROOT)/etc/makefile
```

An Example of Running the Import Utility

This section provides an example using some sample files you can copy to your working directory and use to run the Import utility. This example uses the CSF and CSS files shown in Figure 3-4 and Figure 3-5.

To run the Import utility with the sample files, do the following steps.

1. Make sure that your environment is set up correctly.
2. Create a working directory to contain your sample files.
3. Copy the sample `input.csf` CSF file shown in Figure 3-4 to your working directory.

Make sure that your CSF file has separate lines for each of the two items. Also make sure that you do not have a blank line at the end of your CSF file.

Figure 3-4 An `input.csf` Example

```
Hard Drives~SCSI~1234452~case~15000.00~$~9.1 Gig HD~MassFastSCSI~add  
Hard Drives~EIDE~1234472~case~15000.00~JPY~9.1 Gig HD~MassFastSCSI~add
```

4. Copy the sample `css.xml` CSS file shown in Figure 3-5 to your working directory.

Make sure that you specify the correct path to the `ColumnStructureSpecification.dtd` file in the header of your CSS file.

Replace `<imm_install_dir>` with your iPlanet Market Maker installation directory.

Figure 3-5 A `css.xml` Example

```

<?xml version="1.0" ?>
<!DOCTYPE column-specifications SYSTEM
"imm_install_dir/catalogtools/dtd/ColumnStructureSpecification.dtd">
<column-specifications>
  <column-specification number="1" >
    <category level="1" />
  </column-specification>

  <column-specification number="2" >
    <category level="2" />
  </column-specification>

  <column-specification number="3" >
    <value key="yes"><name><fixed>SKU</fixed></name></value>
  </column-specification>

  <column-specification number="4" >
    <value><name><fixed>UOM</fixed></name></value>
  </column-specification>

  <column-specification number="5" >
    <value>
      <name><fixed>Price</fixed></name>
      <unit><column-ref column-number="6"/></unit>
    </value>
  </column-specification>

  <column-specification number="7" >
    <value>
      <name><fixed>Description</fixed></name>
    </value>
  </column-specification>

  <column-specification number="8" >
    <value>
      <name><fixed>Product Name</fixed></name>
    </value>
  </column-specification>

  <column-specification number="9" >
    <action />
  </column-specification>
</column-specifications>

```

5. To map the category name “Product Name” in the seller catalog to “Name” in the load file, use the `attr-renaming.xml` file shown in Figure 3-6.

Make sure that you specify the correct path to the `attribute-renaming.dtd` file in the header of your file. **Replace `<imm_install_dir>` with your iPlanet Market Maker installation directory.**

Figure 3-6 An `attr-renaming.xml` Example

```
<?xml version="1.0" ?>
<!DOCTYPE attributes SYSTEM
"<imm_install_dir>/catalogtools/dtd/attribute-renaming.dtd">
<attributes>
  <attribute>
    <s_name>Product Name</s_name>
    <m_name>Name</m_name>
  </attribute>
</attributes>
```

6. To map the attribute unit of “\$” in the seller catalog to “USD” in the load file, uses the `units.xml` file shown in Figure 3-7.

Make sure that you specify the correct path to the `units.dtd` file in the header of your file. **Replace `<imm_install_dir>` with your iPlanet Market Maker installation directory.**

Figure 3-7 A `units.xml` Example

```

<?xml version="1.0" ?>

<!DOCTYPE units SYSTEM
"<imm_install_dir>/catalogtools/dtd/units.dtd">

<unit>

    <name>${</name>

    <alias>USD</alias>

</unit>

```

7. Copy the `filenames.mk` file from the `<imm_install_dir>/catalogtools` directory to your working directory.
8. Rename `filenames.mk` in your working directory to `makefile`.
9. Edit the `makefile` file in your working directory and specify the input file names.

Be sure to remove the “#” character when you specify an entry in the `makefile` file. See Figure 3-8. Note that Figure 3-8 omits the parts of the `makefile` that you do not need to edit for this Import utility run.

Figure 3-8 Edited `makefile` file

```

...
INPUT=input.csf
# CSS is the name of the Column Structure Specification file. Must be
# an absolute path name
CSS=$(PWD)/css.xml
#
...
# NAME-MAP is the name of the xml file containing a mapping between
# supplier's attribute names and those to be used by the market
# maker. This file must be valid wrt
NAME-MAP=$(PWD)/attr-renaming.xml
#
...
#

```

```
# UNITS is the name of the file containing vendor-specific unit
# aliases. This must be an absolute path. This file must be valid wrt
# $(DTD)/units.dtd. Optional. Default is no file.
UNITS=$(PWD)/units.xml
#
...
```

10. To create the load file from a UNIX command line, type

```
make ALL
```

If successful, this command creates the `mm.xml` file shown in Figure 3-9.

Figure 3-9 The `mm.xml` File Created by the Import Utility

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE vortex-data-load SYSTEM
"file:///imm_install_dir/catalog/labs/mm.dtd">
<vortex-data-load version="1.0">
  <Hard_Drives name="Hard Drives" vortex-type="category">
    <SCSI name="SCSI" vortex-type="category">
      <Item vortex-type="item" action="add">
        <Attribute vortex-type="attribute" value="1234452" name="SKU"
key="yes" />
        <Attribute vortex-type="attribute" value="case" name="UOM"
key="no" />
        <Attribute vortex-type="attribute" value="15000.00"
dataType="currency" name="Price" key="no" />
        <Attribute vortex-type="attribute" value="9.1 Gig HD"
name="Description" key="no" />
        <Attribute vortex-type="attribute" value="MassFastSCSI"
name="Name" key="no" / unit="USD" />
      </Item>
    </SCSI>
    <EIDE name="EIDE" vortex-type="category">
      <Item vortex-type="item" action="add">
        <Attribute vortex-type="attribute" value="1234472" name="SKU"
key="yes" />
        <Attribute vortex-type="attribute" value="case" name="UOM"
key="no" />
        <Attribute vortex-type="attribute" value="15000.00"
dataType="currency" name="Price" key="no" />
        <Attribute vortex-type="attribute" value="9.1 Gig HD"
name="Description" key="no" />
        <Attribute vortex-type="attribute" value="MassFastSCSI"
name="Name" key="no" / unit="JPY" />
      </Item>
    </EIDE>
  </Hard_Drives>
</vortex-data-load>
```

11. If you want to display an image file with an item in the iPlanet Market Maker catalog user interface, such as an image to go with one of hard drives in Figure 3-9, use the following format in your `mm.xml` file.

```
<Item ...>
<Attribute vortex-type="attribute" value="image_file" name="value">
</Item>
```

The `image_file` can be an absolute or relative path. If you specify a relative path, that path must be under the following directory.

```
<server_root_dir>/docs
```

The `value` for the `name` specifies the image size, which can be `imagefile_sm` for a thumbnail size image, `imagefile` for a regular size image, or `imagefile_lg` for a large size image only.

12. To load the `mm.xml` file in the iPlanet Market Maker catalog database, type the following information on a UNIX command line. Substitute your user name, password, and company name for the entries within the brackets (“<>”).

```
run -xms64M -Xmx512M
com.iplanet.ecommerce.vortex.catalogtools.loader.Loader -USER
<your_user_name> -PASSWORD <your_password> -COMPANYID
<your_company_name> -IN mm.xml -ontology
```

Or:

```
$IMM_HOME/catalogtools/bin/load.sh
```

For information about your load run, see the `<-COMPANYID>.error` and `<-COMPANYID>.log` files.

For more information about how to load the load file in the PMC, see the following “Loading the Load File in the PMC” section.

Loading the Load File in the PMC

After you have created your `mm.xml` file, you can load it into the iPlanet Market Maker catalog database. Note that you must register your user and company name in an iPlanet Market Maker marketplace to load catalog information.

To load an `mm.xml` file into the catalog database, use one of the following commands:

```
run -xms64M -Xmx512M
com.iplanet.ecommerce.vortex.catalogtools.loader.Loader [inputs]
```

Or:

```
$IMM_HOME/catalogtools/bin/load.sh
```

The inputs to this command line are described in the following table. The required inputs to the loader are in **bold** in the left column in Table 3-10.

Table 3-10 Inputs to the Loader Command Line

Loader Input	Definition
<code>-xms64M</code>	Increases the default initial heap size for better performance loading large catalogs.
<code>-Xmx512M</code>	Increases the default maximum heap size for better performance loading large catalogs.
-USER	The user name.
-PASSWORD	The user password.
-COMPANYID	Company login name or ID of the user running the loader. If you do not specify the <code>-VENDORID</code> option, the loader adds items under this company name or ID.
<code>-VENDORID</code>	Optional vendor identification, typically a company name.
<code>-LOCALE</code>	Specifies an optional locale other than the default. If you specify a locale, use the form <code><language code><country code></code> , where the language and country code comply to the ISO 639 standard. The locale for French, for example, is <code>fr-FR</code> .
-IN XML_file_name	Name of the load file. The default file name is <code>mm.xml</code> .
<code>-LOG log_file_name</code>	Log file name. The default file name is the <code><-VENDORID></code> name with a <code>.log</code> extension.
<code>-ERROR error_file_name</code>	Error file name. The default file name is the <code><-VENDORID></code> name with an <code>.error</code> extension.

Table 3-10 Inputs to the Loader Command Line

-ITEM item_number	Optional item number to begin processing. The default item number is 0.
-ontology	Creates a new ontology. This option is only for marketmaker administrators and should be used very carefully. This option allows the loader to make ontology changes by adding new categories in the PMC. You should turn this option on only when you want to create a new ontology, such as when you create a catalog database for the first time. It is turned off by default.
-approve	Approves the items after loading.
-THREADS	Specifies the number of threads. The default is 1. In some cases increasing the number of threads can improve performance.
-MAXITEMBATCHSIZE	Specifies the number of items in a category to process in a transaction. The default is the number of items in a category as specified in the Public Master catalog XML file.
-noIndex	
-reIndex	
-approveCatalog	Approve all pending catalog items after loading.
-D vortex_debugLevel	Specifies a Java property that can have the following values. 0 - No debugging output. 3 - Exceptions in output. 7 - Sql statements in output.
stats file_name	Outputs statistics. For iPlanet Market Maker internal use.
-effective Start Date	yyyy_mm_dd
-effective End Date	yyyy_mm_dd

Exporting Catalogs

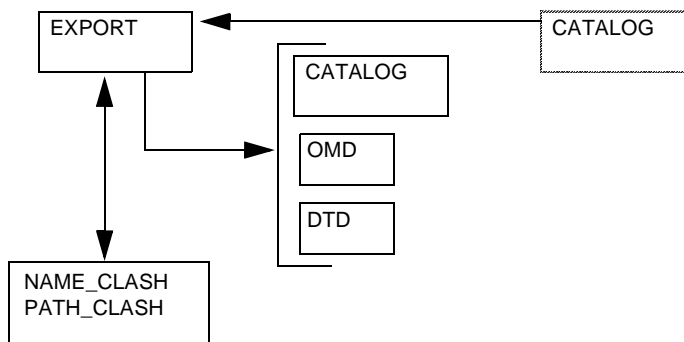
In addition to importing catalogs, you can also export them. You might want to export catalogs for the following reasons.

- To use the Export utility to provide the existing PMC ontology information to a new seller who wants to import a new catalog. The new seller can use the `omd.xml` file generated by the Export utility to ensure that the new seller catalog ontology matches the existing PMC ontology.
- The Export utility lets you export a seller catalog for backup and archiving purposes.

Given an existing PMC, you can export an `mm.xml` file that contains the catalog information for a specified seller. If there are name or path clash violations after your first Export utility run, you must specify the `-n` or `-p` options to resolve the violations the next time you run the Export utility. If there are no name and path clashes after subsequent Export utility runs, the name or path clash report files are empty.

For more information about name and path clashes, see the “Detecting Clashes” section in Chapter 3.

The Export utility outputs `mm.dtd`, `mm.xml`, `omd.xml`, and name and path clash files. See Figure 3-10.

Figure 3-10 Export Utility Flow**Legend**

EXPORT utility

CATALOG - PMC

OMD - omd.xml



Inputs

DTD - mm.dtd



Output targets

NAME_CLASH - Name lash
reportPATH_CLASH - Name clash
report

Running the Export Utility

To export an `mm.xml` seller catalog file from the catalog database, use one of the following commands:

```
run -xms64M -Xmx512M
com.iplanet.ecommerce.vortex.catalogtools.export.Main [inputs]
```

Or:

```
$IMM_HOME/catalogtools/bin/export_cat.sh
```

The inputs to this command line are described in the following table. The required inputs to the loader are in **bold** in the left column in Table 3-11.

Table 3-11 Inputs to the Export Utility Command Line

Export Utility Input	Definition
-xms64M	Increases the default initial heap size for better performance loading large catalogs.
-Xmx512M	Increases the default maximum heap size for better performance loading large catalogs.
-ROOT	Root directory for the files that the export utility needs to access., which is <imm_install_dir>/catalogtools.
-PASSWORD	The user password.
-COMPANYID	Marketmaker administrator login name or ID of the user running the Export utility. If you do not specify the <code>-seller</code> option, the Export utility exports items under this company name or ID.
-USER	The user name.
-s seller_id	Optional seller identification, typically a company name.
-n name_file	Name of the name clash report file. For information about resolving name clashes, see the “Resolving Name Clashes” section in Chapter 3.
-p path_file	Name of the path clash report file. For information about resolving path clashes, see the “Resolving Path Clashes” section in Chapter 3.
-l	Specifies an optional locale other than the default. If you specify a locale, use the form <language code>-<country code>, where the language and country code comply to the ISO 639 standard. The locale for French, for example, is <code>fr-FR</code> .
-v	Enables the verbose messaging mode, which can be useful for debugging purposes.

Index

A

- Action column entries 27
- Adding items 28
- Attributes
 - hierarchy structure 16
 - normalization 20, 24
 - renaming 21
 - types 21

C

- Catalog hierarchy 16
- Categories 16
- Character-separated fields (CSF) file 24
- Clashes
 - detecting 33
 - resolving name clashes 33
 - resolving path clashes 37
- Column references 30
- Column structure specification (CSS) file 24
- Column structure, implementing 27
- Command arguments 59
- Comments in the makefile file 54
- CSF file, row and column structure 26

D

- Deleting items 28
- Detecting clashes 33
- Different currencies, specifying 30
- Directory locations 60

E

- Editing the makefile file 54
- Error reports 59
- Extensible markup language (XML) 23

G

- Graphical representation of catalog hierarchy structure 30

H

- Hierarchy
 - categories 18
 - categories and sub-categories 23

seller catalogs 16

I

Implementing column structure 27

Internal targets 60

Interpreting attribute values 30

iPlanet

customer support 12

Items 16

adding 28

deleting 28

mapping 18

uniquely identifying 17

updating 28

K

Keys 26

L

Load file 24, 41

M

make command 54

Mapping items 18

mm.xml file 24, 44

N

Name clashes 33, 57

Normalization rules 20

O

omd.xml file 42

Ontology mapping 18, 24, 41, 43

Optional input files 57

Optional variables 58

Output targets 55

P

Path clashes 33, 57

Public Master catalog (PMC) 15, 23

R

Renaming

attribute names 20

attribute units 21

Representation issues, XML 33

Required input files 56

Resolving clashes 33, 37

Running the Import utility with CHECK 33

S

Sample seller catalog representation 16

Seller catalogs 15

Setting up and running the Import utility 51

SKU attribute 26

Specifying different currencies 30

Steps to convert catalog data to XML 24

Storage keeping unit (SKU) 17

Structural information, CSS file 24

U

Ultimate targets 58

Updating items 28

V

vendor.xml file 32, 42

W

website

for iPlanet customer support 12

X

XML

elements 24, 32

representation issues 33

representation of a seller catalog 24

