

iManager 2.7 Developer Kit

Novell® Developer Kit

January 13, 2009

www.novell.com



Legal Notices

Novell, Inc. makes no representations or warranties with respect to the contents or use of this documentation, and specifically disclaims any express or implied warranties of merchantability or fitness for any particular purpose. Further, Novell, Inc. reserves the right to revise this publication and to make changes to its content, at any time, without obligation to notify any person or entity of such revisions or changes.

Further, Novell, Inc. makes no representations or warranties with respect to any software, and specifically disclaims any express or implied warranties of merchantability or fitness for any particular purpose. Further, Novell, Inc. reserves the right to make changes to any and all parts of Novell software, at any time, without any obligation to notify any person or entity of such changes.

Any products or technical information provided under this Agreement may be subject to U.S. export controls and the trade laws of other countries. You agree to comply with all export control regulations and to obtain any required licenses or classification to export, re-export, or import deliverables. You agree not to export or re-export to entities on the current U.S. export exclusion lists or to any embargoed or terrorist countries as specified in the U.S. export laws. You agree to not use deliverables for prohibited nuclear, missile, or chemical biological weaponry end uses. Please refer to the [International Trade Services \(http://www.novell.com/company/policies/trade_services\)](http://www.novell.com/company/policies/trade_services) for more information on exporting Novell software. Novell assumes no responsibility for your failure to obtain any necessary export approvals.

Copyright © 2009 Novell, Inc. All rights reserved. No part of this publication may be reproduced, photocopied, stored on a retrieval system, or transmitted without the express written consent of the publisher.

Novell, Inc. has intellectual property rights relating to technology embodied in the product that is described in this document. In particular, and without limitation, these intellectual property rights may include one or more of the U.S. patents listed on the [Novell Legal Patents Web page \(http://www.novell.com/company/legal/patents/\)](http://www.novell.com/company/legal/patents/) and one or more additional patents or pending patent applications in the U.S. and in other countries.

Novell, Inc.
404 Wyman Street, Suite 500
Waltham, MA 02451
U.S.A.
www.novell.com

Online Documentation: To access the latest online documentation for this and other Novell products, see [the Novell Documentation Web page \(http://www.novell.com/documentation\)](http://www.novell.com/documentation).

Novell Trademarks

For Novell trademarks, see [the Novell Trademark and Service Mark list \(http://www.novell.com/company/legal/trademarks/tmlist.html\)](http://www.novell.com/company/legal/trademarks/tmlist.html).

Third-Party Materials

All third-party trademarks are the property of their respective owners.

Contents

About This Guide	9
1 Getting Started	11
1.1 Requirements	11
1.2 Installing the SDK	11
1.3 Using the iManager SDK	12
1.3.1 Prerequisites	12
1.3.2 Starting iManager	13
1.3.3 Setting Up Role-Based Services	14
1.4 Building the Sample Plug-Ins	15
2 Concepts	17
2.1 How iManager Works	17
2.1.1 Tasks	17
2.1.2 Property Books and Pages	18
2.1.3 Architecture	18
2.2 The iManager Directory Structure	19
2.3 Role-Based Services	20
2.4 Module ID	21
2.5 Connection Modes	22
2.6 iManager Plug-ins	22
3 Creating Tasks	23
3.1 Overview	23
3.2 Creating a Task: An Example	23
3.2.1 Creating the Java Class	24
3.2.2 Creating the UI	26
3.2.3 Creating the Registration File	28
3.2.4 Localizing the Task	29
3.2.5 Deploying the Task	30
3.3 Tips for Creating Task Java Classes	30
3.3.1 Encoding Data	32
3.4 Tips for Creating JSPs	32
3.4.1 Encoding Data Using Tag Libraries	33
3.4.2 Encoding Data Using Java Methods	33
3.5 Creating Registration Files	35
3.5.1 Using the Create XML Install File Task	37
3.6 Specifying Conditions for Task Execution	37
3.7 Launching Tasks and Delegating to Tasks	38
3.7.1 The Launch Service and Launch Actions	38
3.7.2 Life Cycles of Launched and Delegate Tasks	39
3.7.3 Accessing Launching and Delegation Features Programmatically	39
3.7.4 Launching and Delegation Methods in GadgetInstance	40
3.7.5 Task Delegation Example	40
3.7.6 Closing the Window of a Launched Task	41
3.7.7 More Examples	41
3.8 Task Chaining	41

3.8.1	Setting Objects in the Initial Task	42
3.8.2	Getting Objects in the Chained Task	42
3.8.3	Defining the Chained Task in a Registration File	42
3.9	Extending the Object Management Tasks	43
3.9.1	Registering an Object Type for the Create Object Task	43
3.9.2	Extending the Delete, Move, and Rename Tasks	44
3.9.3	Disallowing Operation of an Object Management Task on an Object Type	45
3.10	Enabling a Task for the Object View	45
3.10.1	Register a Task to Work with Specific Object Types	46
3.10.2	Retrieve Objects from iManager	47
3.11	Using the AdminNamespace	48
4	Creating Property Books and Pages	49
4.1	Creating a Property Book	49
4.1.1	Create the Java Class	49
4.1.2	Create the Property Book Pages	49
4.1.3	Create the XML Registration File	50
4.2	Creating a Property Book Page	50
4.2.1	Create the Java Class	50
4.2.2	Create the JSP	51
4.2.3	Create the XML Registration File	52
5	Using the Plug-In Studio	53
5.1	The Plug-In Studio User Interface	53
5.2	Creating a Custom Plug-In	54
5.2.1	Adding Drop Down List Values Dynamically	55
5.3	Control Parameters	56
5.4	What the Plug-In Studio Creates	57
5.5	Customizing Plug-Ins Created with the Plug-In Studio	57
5.5.1	Dynamically Updating Drop-down Lists	58
6	Using the iManager Widgets	61
6.1	The Object Selector Widget	61
6.1.1	Include Files	62
6.1.2	Parameter Variables	63
6.1.3	Advanced Selection XML Syntax	65
6.1.4	Dynamically Enabling and Disabling the Object Selector	65
6.1.5	Pre-Processing and Post-Processing Routines (preOS and postOS)	65
6.1.6	Making the Root Selectable	66
6.1.7	Making Public and This Selectable	66
6.1.8	Filtering on All Container Types	66
6.1.9	Filtering on Containers that Are Partitions	66
6.1.10	Object Selector Support in the iManager Tag Library	66
6.1.11	A JSP Tag Library Example: Delete User	68
6.1.12	Troubleshooting	68
6.2	The Advanced Selection Widget	69
6.2.1	How to Call the AS Widget	70
6.2.2	Include Files	70
6.2.3	Parameter Variables	70
6.2.4	A JSP Example: Delete Users	71
6.2.5	Setting the Options at Runtime Using JavaScript	73
6.2.6	Implementing AdvSelTypeInfoCallback	74
6.2.7	The Resulting XML Selection Criteria	75

6.3	The MVStringEditor Widget	75
6.3.1	How to Use the MVStringEditor Widget	76
6.3.2	Modes	77
6.3.3	Parameters	77
6.3.4	JavaScript API	78
6.3.5	Examples	79
6.4	The Date/Time Widget	79
7	Creating a Plug-In	81
7.1	Overview	81
7.1.1	Plug-In File Structure	81
7.1.2	Plug-In Directory Objects and Attributes	83
7.1.3	Plug-in Update and Uninstallation	83
7.1.4	Plug-In Installers	83
7.2	Creating a Manifest File	83
7.3	Plug-In Information Through XML	85
7.4	Creating a Plug-In Installer	87
7.5	Installing Plug-ins to an Existing iManager	88
7.5.1	Installing a Plug-In Programmatically	89
7.5.2	Installing a Plug-In Manually	89
7.6	Precompiling JSPs for Tomcat 5	90
7.6.1	Tomcat 5 Compiled Java Class File Location	90
7.6.2	Precompiling JSP Pages with Ant	91
7.7	Testing a Plug-In	92
7.7.1	Installation Testing	92
7.7.2	Plug-In Studio Testing	95
7.7.3	Individual Task Testing	96
7.7.4	Object View Testing	96
7.7.5	Generic Operation Testing	96
8	Installing iManager with Your Application	97
8.1	Windows	97
8.2	Linux (SUSE and Red Hat)	98
8.3	Installer.properties File	98
9	Logging Debug Messages	101
10	Customizing iManager for Schema Extensions	103
10.1	Providing an Image for a New Object Class	103
10.2	Providing Translated Names for New Object Classes and Attributes	103
10.3	Providing a Creator for a New Object Class	104
10.4	Handling Deletion, Moving, and Renaming of a New Object Class	104
10.5	Providing Pages for the Modify Object Property Book	104
10.6	Providing Tasks to Interact with a New Object Class	104
11	Creating an iPrint Gateway Plug-In	107
11.1	Introduction to iPrint	107
11.2	Getting Started	108
11.3	Using the Sample Gateway Plug-In	109
11.4	Creating Gateway Plug-In Java Class Files	109

11.5	Creating Gateway Plug-In JSP Files	110
11.6	Creating Gateway Plug-In Registration Files	110
12	Reference	111
12.1	iManager API Documentation	111
12.2	XML Schema for Installation and Registration Files.	111
12.3	Role-Based Services Directory Objects	111
12.3.1	rbsCollection	111
12.3.2	rbsRole	112
12.3.3	rbsModule	112
12.3.4	rbsTask	112
12.3.5	rbsBook	112
12.3.6	rbsScope	113
12.4	eDirectory Access Service XML Formats for eDirectory Attribute Syntax Definitions	113
A	iManager Security Issues	119
A.1	Secure LDAP Certificates	119
A.2	Self-Signed Certificates	120
A.3	iManager Authorized Users and Groups	121
A.4	Preventing Username Discovery	121
A.5	Tomcat Settings	122
A.6	Encrypted Attributes	122
A.7	Secure Connections	122
B	Revision History	125

About This Guide

iManager is a Web-based application you can use to easily build network management services that are accessible by Web browsers, PDAs, phones, and other devices.

This document explains how to use the iManager 2.7 Developer Kit (SDK) to create plug-ins for iManager. The SDK provides tools to develop and test iManager plug-ins. It includes iManager libraries, sample plug-ins, documentation, and a servlet container that is configured for testing iManager 2.7 plug-ins on your workstation. This SDK is designed to provide a standalone iManager plug-in development environment.

This guide contains the following sections:

- ♦ “Getting Started” on page 11
- ♦ “Concepts” on page 17
- ♦ “Creating Tasks” on page 23
- ♦ “Creating Property Books and Pages” on page 49
- ♦ “Using the Plug-In Studio” on page 53
- ♦ “Using the iManager Widgets” on page 61
- ♦ “Creating a Plug-In” on page 81
- ♦ “Installing iManager with Your Application” on page 97
- ♦ “Logging Debug Messages” on page 101
- ♦ “Customizing iManager for Schema Extensions” on page 103
- ♦ “Creating an iPrint Gateway Plug-In” on page 107
- ♦ “Reference” on page 111

Audience

This guide is intended for Java developers interested in creating plug-ins for iManager 2.7 and later.

Feedback

We want to hear your comments and suggestions about this manual and the other documentation included with this product. Please use the User Comment feature at the bottom of each page of the online documentation.

Documentation Updates

For the most recent version of this guide, see the [iManager NDK page \(http://developer.novell.com/ndk/imgrsdk.htm\)](http://developer.novell.com/ndk/imgrsdk.htm).

Additional Documentation

For the developer support postings for the iManager 2.7 SDK, see the related [Developer Support Forum \(http://developer.novell.com/ndk/devforums.htm\)](http://developer.novell.com/ndk/devforums.htm).

For the most recent version of this guide, see the [iManager Developer Kit \(http://developer.novell.com/ndk/doc/imgrsdk/imgr_enu/data/bktitle.html\)](http://developer.novell.com/ndk/doc/imgrsdk/imgr_enu/data/bktitle.html).

Docuentation Conventions

In Novell documentation, a greater-than symbol (>) is used to separate actions within a step and items in a cross-reference path.

A trademark symbol ([®], [™], etc.) denotes a Novell trademark. An asterisk (*) denotes a third-party trademark.

When a single pathname can be written with a backslash for some platforms or a forward slash for other platforms, the pathname is presented with a backslash. Users of platforms that require a forward slash, such as Linux or UNIX, should use forward slashes as required by your software.

Getting Started

1

This section covers the following topics to help you get started with the iManager 2.7 Developer Kit (SDK):

- ♦ [Section 1.1, “Requirements,” on page 11](#)
- ♦ [Section 1.2, “Installing the SDK,” on page 11](#)
- ♦ [Section 1.3, “Using the iManager SDK,” on page 12](#)
- ♦ [Section 1.4, “Building the Sample Plug-Ins,” on page 15](#)

1.1 Requirements

To use this SDK, you must have the following installed on your workstation:

- ☐ Sun* Microsystems [Java* SE Development Kit \(JDK*\) version 1.5.0, update 11, or later](http://java.sun.com/javase/downloads/index_jdk5.jsp) (http://java.sun.com/javase/downloads/index_jdk5.jsp)
- ☐ Microsoft* Internet Explorer 6 SP1 or later, 7.0
- ☐ Firefox* 1.5.x, 2.x, or 3.x
- ☐ Novell® JClient
The SDK includes the JClient runtime library for Windows*.
- ☐ Novell International Cryptographic Infrastructure (NICI) version 2.7.3
The SDK includes the NICI installation files for Windows and Linux*.

You should have access to a server running the following:

- ☐ Novell eDirectory™

You should have at least a basic understanding of the following technologies:

- ☐ Java
- ☐ XML
- ☐ Java Server Pages (JSPs)
- ☐ HTML

1.2 Installing the SDK

The SDK is part of the Novell Developer Kit (NDK), and you can download it from the [iManager SDK Web site](https://twiki.innerweb.novell.com/bin/view/IDApps/IManager27IDE#Setting_up_the_iManager_SDK_envi) (https://twiki.innerweb.novell.com/bin/view/IDApps/IManager27IDE#Setting_up_the_iManager_SDK_envi). The SDK consists of binaries, sample code, and documentation components, which you can download in a single ZIP-compressed archive file or as separate ZIP files. You can also view the documentation on the NDK site.

To install an SDK component, use an unzip utility to decompress the contents of the ZIP file to a directory of your choice. The proper organization of the SDK files on your hard drive is essential for the SDK to work properly; therefore, when you unzip the files, make sure that you select the options that will append the path of each file, as stored in the archive, to the output directory. On Linux, make sure that you also preserve permissions.

The directory where you install the SDK files is referred to as *SDK_HOME* in this documentation.

1.3 Using the iManager SDK

To test and debug your plug-ins, you need access to a server running iManager. The SDK provides a Tomcat* servlet container and locally executed version of iManager, known as iManager Workstation, configured at *sdk_home/tomcat/webapps/nps*.

The following sections discuss how to use the SDK:

- ♦ [Section 1.3.1, “Prerequisites,” on page 12](#)
- ♦ [Section 1.3.2, “Starting iManager,” on page 13](#)
- ♦ [Section 1.3.3, “Setting Up Role-Based Services,” on page 14](#)

1.3.1 Prerequisites

There are some tasks you need to complete before launching the iManager SDK on a workstation that will run the developer version of iManager. Although the tasks are the same, they are performed differently depending on the operating system you are using:

- ♦ [“Linux” on page 12](#)
- ♦ [“Windows” on page 13](#)

Linux

Do the following to start iManager on Linux:

Set the JAVA_HOME Environment Variable: Before starting iManager, make sure that a *JAVA_HOME* environment variable is set to the installation directory of the JDK. Use the *echo* command to display the value of *JAVA_HOME*:

```
echo $JAVA_HOME
```

If you need to set *JAVA_HOME*, use the *export* command to specify the location of the JDK on your system. For example:

```
export JAVA_HOME=/usr/java/jdk1.5.0
```

Set Up NICI: You must install NICI before you can run iManager. Use the *rpm* command to determine whether NICI is installed:

```
rpm -q nici
```

If NICI is installed, the *rpm* command displays the package version. If it is not installed, *rpm* displays the message *package nici not installed*.

To install NICI on Linux, use the *rpm* command as the root user:

```
rpm -Uvh SDK_HOME/NICI/nici.i386.rpm
```

Windows

Do the following to start iManager on Windows:

Set the JAVA_HOME Environment Variable: Before starting iManager, make sure that a JAVA_HOME environment variable is set to the installation directory of the JDK. Check the value of JAVA_HOME in *Control Panel > System > Environment Variables*. If necessary, set JAVA_HOME to the location of the JDK on your system. For example:

```
JAVA_HOME=C:\Program Files\Java\jdk1.5.0_11
```

Set Up NICI: You must install NICI before you can run iManager. If you do not have NICI installed, the NICI installation file is launched automatically the first time you start iManager.

1.3.2 Starting iManager

To start iManager Workstation, use the script file located in `SDK_HOME\imgrsdk`:

Linux: `./startSDK.sh`

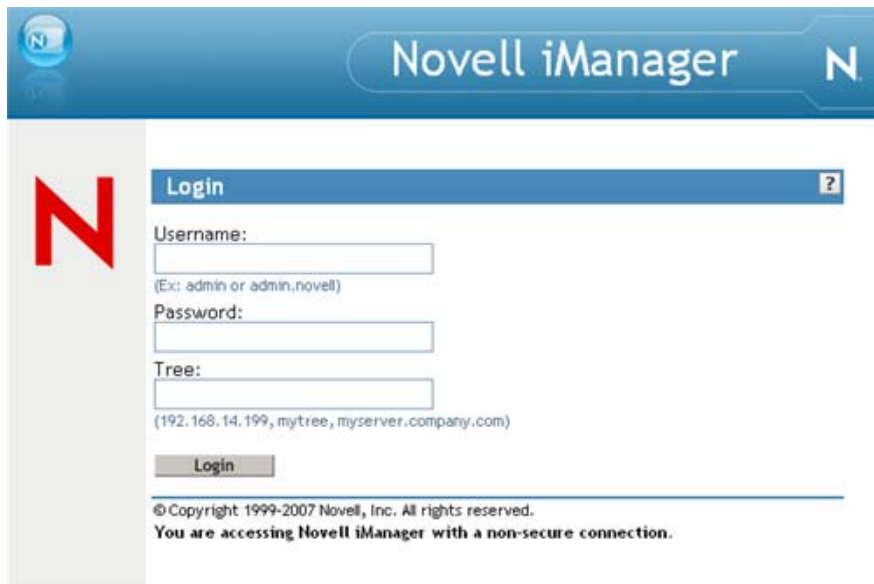
Windows: `startSDK.bat`

This loads the Tomcat servlet container and launches iManager Workstation.

The first thing you see is a login page where you enter a user name, password, and tree. In the tree field, enter the name of the tree or the IP address of a server in the tree that you want to manage.

NOTE: If you don't specify context as part of the username (for example, admin.waltham.novell), iManager performs a contextless lookup to locate the specified user.

Figure 1-1 The iManager Login page.



Novell iManager

N

Login ?

Username:
(Ext: admin or admin.novell)

Password:

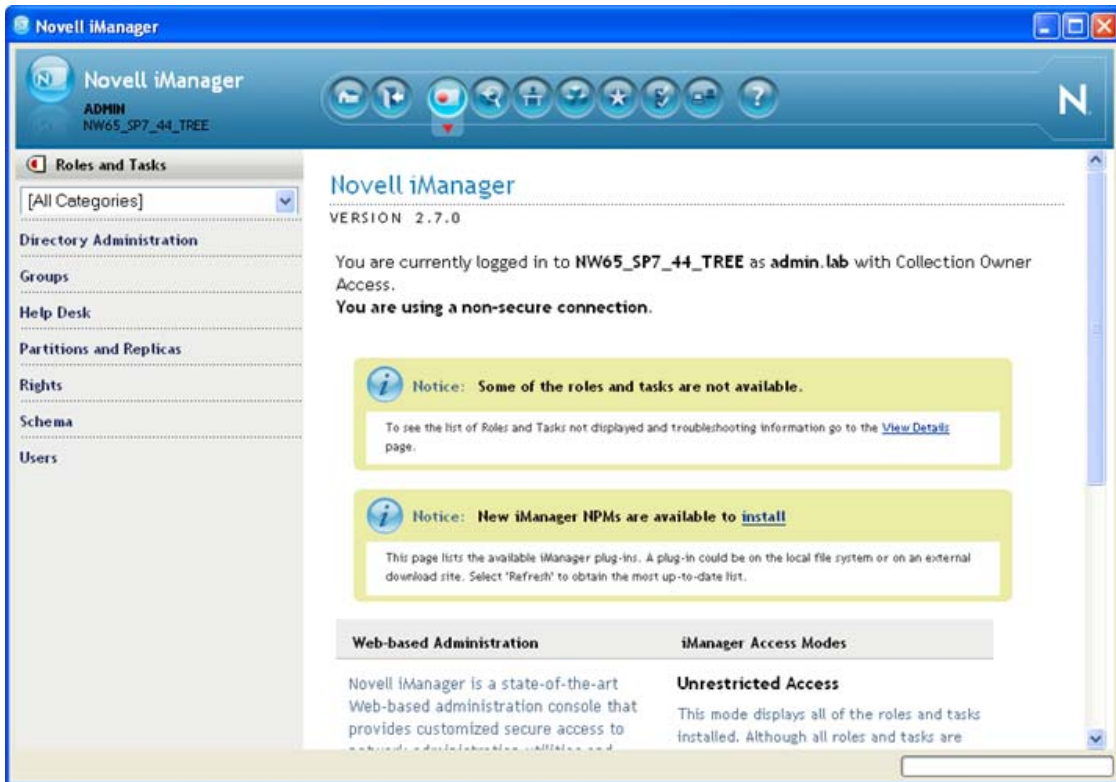
Tree:
(192.168.14.199, mytree, myserver.company.com)

Login

© Copyright 1999-2007 Novell, Inc. All rights reserved.
You are accessing Novell iManager with a non-secure connection.

After you log in, you see the iManager main page. The pane at the top of the page is the banner area. The pane on the lower left side of the main page is the navigation area. It contains a list of all of the roles and tasks available to you. The pane on the lower right side of the page is the content area. It contains information and instructions, and, when a task is selected, it contains the user interface for the task.

Figure 1-2 The iManager main page.



For information about navigating the iManager interface, see the [iManager 2.7 Administration Guide \(http://www.novell.com/documentation/imanager27/imanager_admin_27/data/ahv1qb4.html\)](http://www.novell.com/documentation/imanager27/imanager_admin_27/data/ahv1qb4.html).

NOTE: Closing iManager automatically unloads Tomcat as part of the shutdown process. No further action is necessary.

1.3.3 Setting Up Role-Based Services

Before you can use a task that requires Role-Based Services (RBS), such as the Plug-In Studio (see “[Using the Plug-In Studio](#)” on page 53), you must configure RBS. For more information about RBS, see [Section 2.3, “Role-Based Services,”](#) on page 20.

You can develop and test plug-ins without setting up RBS by using the Unrestricted Access connection mode. However, this lets users view all roles and tasks even if they do not have rights to use them.

You must set up RBS if you want to use the Plug-In Studio. For more information about connection modes, see [Section 2.5, “Connection Modes,”](#) on page 22.

To configure RBS:

1 Create a new Collection.

1a From the Configure view, select *Role-Based Services > RBS Configuration* to open the RBS Configuration page in the Content frame.

1b In the RBS Configuration page, select *New > Collection*.

1c In the Create Collection page, specify a name and a container for the new collection, then click *OK*.

2 Install RBS modules into the new collection.

Each RBS module contains roles and tasks that you can assign to users. The RBS Configuration page lists the number of modules available for the collection you just created.

2a In the Not-Installed column, select the number next to the collection you just created.

2b In the Collection page, select *iManager Base Content* and any other modules you want to install, then click *Install*.

WARNING: Only select modules with versions 2.7.0 or later. Older modules might install but they will not work correctly with the SDK.

2c When the module installation completes, click *OK*.

3 Assign users to roles.

Before users can perform tasks, they must be assigned to the appropriate role. Each role object has a list of members to which you can assign objects in order to grant users access to the tasks associated with the role.

3a In the RBS Configuration page, select the collection you created in [Step 1](#).

3b In the Collection page, select a role to which you want to add users, then click *Actions > Member Associations*.

3c In the *Name* field, specify the objects that you want to assign to the role.

Click the *Browse* button search for the objects you want to add to the member list. Supported objects include users, groups, organizations, organizational units, and domains.

3d In the *Scope* field, specify the directory container in which the specified objects can perform the tasks associated with the role. Click the *Browse* button to search for the container you want to specify as the member scope.

3e Click *Add* to add the designated objects to the member list.

3f Click *OK* to save your changes, then click *OK* to return to the Collection page.

After adding members to a role, you should be able to log in as a role member and see the role and its associated tasks in the iManager navigation frame.

1.4 Building the Sample Plug-Ins

The SDK includes several sample plug-ins and an Apache* Ant build environment for building them.

NOTE: To build the sample plug-ins on any platform, you must have the `JAVA_HOME` environment variable set, as described in [“Prerequisites” on page 12](#).

To build the sample plug-ins, use the build script appropriate to your platform. The build scripts are located at *SDK_HOME/imgrsdk/samples*:

Linux: `./build.sh`

Windows: `build.bat`

This section explains the following iManager development concepts:

- ♦ [Section 2.1, “How iManager Works,” on page 17](#)
- ♦ [Section 2.2, “The iManager Directory Structure,” on page 19](#)
- ♦ [Section 2.3, “Role-Based Services,” on page 20](#)
- ♦ [Section 2.4, “Module ID,” on page 21](#)
- ♦ [Section 2.5, “Connection Modes,” on page 22](#)
- ♦ [Section 2.6, “iManager Plug-ins,” on page 22](#)

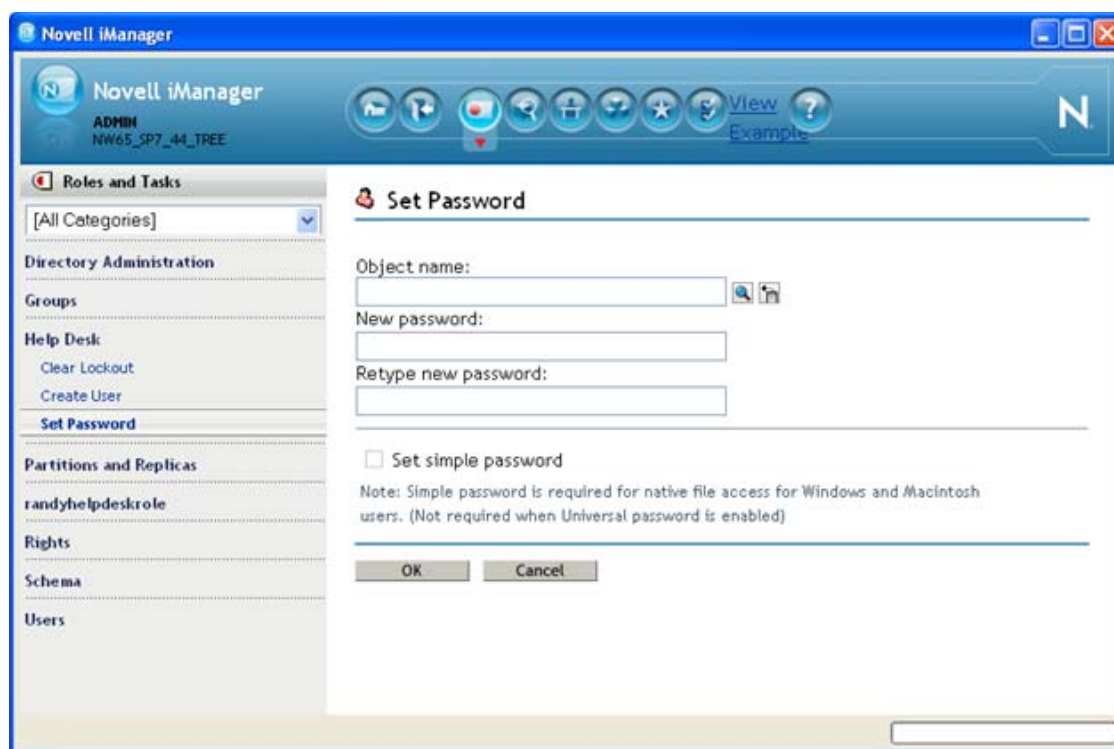
2.1 How iManager Works

iManager is based on the J2EE servlet specification and runs on the Tomcat* servlet container. iManager itself is a framework for extensions called plug-ins. Tasks, property books, and property book pages are types of plug-ins.

2.1.1 Tasks

Plug-ins that perform a distinct management function, such as creating a user or setting a password, are called tasks. iManager organizes tasks by role in the Navigation frame.

Figure 2-1 Tasks in the iManager UI



In [Figure 2-1](#), the *Set Password* task is selected, and the task's UI displays in the Content frame.

Tasks consist of Java class files, Java Server Pages (JSPs), registration files, and resource files that are placed in the appropriate directories within the document root directory of the Novell Portal Services (NPS) Web Application servlet. For more information about the iManager directory structure, see [Section 2.2, "The iManager Directory Structure," on page 19](#).

2.1.2 Property Books and Pages

A property book displays a group of pages that allow a user to view or modify the properties of an object or set of objects of the same type. Each page of the book has a tab that the user can click to switch to that page.

For example, a property book that modifies the attributes of User objects might have a page that allows an administrator to specify a user's login script. Another page might allow an administrator to change a user's e-mail address and telephone number.

Property books can be assigned to roles and appear in the list of tasks for a role.

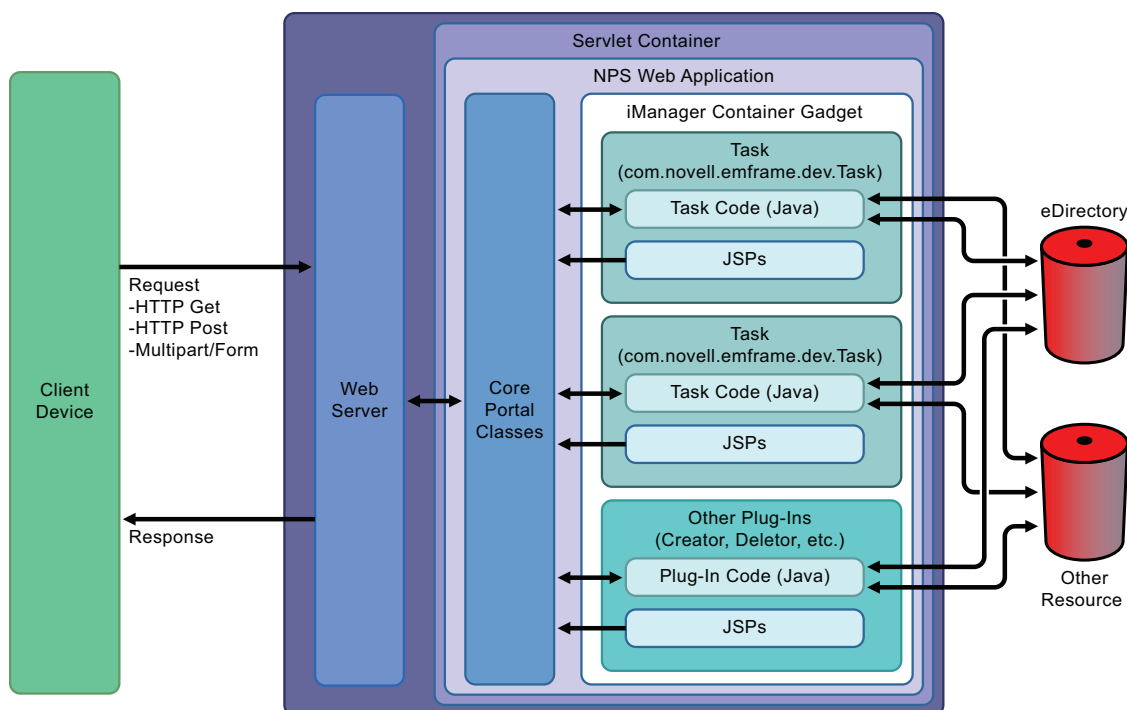
Property book pages are similar to tasks. However, they are for displaying and modifying attributes in a single view. For more complex, wizard-like UI, you should create a task.

Like tasks, property book pages consist of Java class files, JSPs, registration files, and resource files that are placed in the appropriate directories within the document root directory of the NPS Web Application servlet. For more information about the iManager directory structure, see [Section 2.2, "The iManager Directory Structure," on page 19](#).

2.1.3 Architecture

The following figure illustrates the iManager architecture.

Figure 2-2 The iManager 2.7 architecture



iManager tasks can use the Role-Based Services (RBS) technology product, which is part of iManager. You can use RBS to base the availability of your tasks on the user's role in an organization. For more information about RBS, see [Section 2.3, "Role-Based Services," on page 20](#).

For detailed instructions on how to create iManager plug-ins, see [Chapter 3, "Creating Tasks," on page 23](#).

2.2 The iManager Directory Structure

iManager files are placed in the appropriate directories within the NPS Web Application servlet document root directory. NPS follows the Java Servlet 2.3 Specification defined by [JSR 53 of the Java Community Process](http://jcp.org/aboutJava/communityprocess/first/jsr053/index.html) (<http://jcp.org/aboutJava/communityprocess/first/jsr053/index.html>). The Java Servlet Specification prescribes a general directory structure for Web applications implemented as Java servlets. If you are not familiar with this directory structure, we recommend that you study Chapter 9 of the Java Servlet 2.3 Specification.

The following table describes the directories in the document root of the NPS Web Application, which is `SDK_HOME\tomcat\webapps\nps`. Everything in the `nps` directory is directly accessible to client devices, with the exception of the `WEB-INF` subdirectory and its contents.

Table 2-1 Directories of the NPS Web application

Directory	Description
portal/modules	Module files, such as JSPs and registration files. Each subdirectory represents a module and is named with the module ID. For information about module IDs, see Section 2.4, "Module ID," on page 21 .

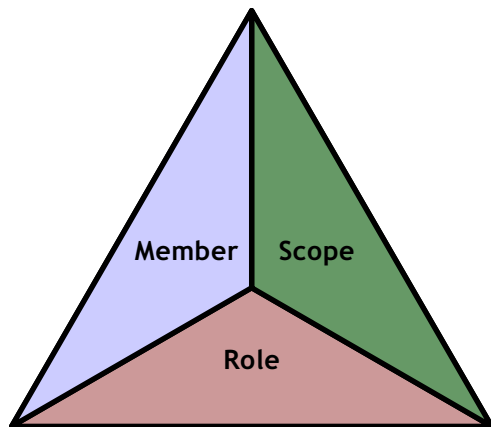
Directory	Description
portal\modules\module name\applets	Applets that are used by plug-ins.
portal\modules\module name\css	Cascading style sheets that are used to create the user interface of plug-ins.
portal\modules\module name\errors	Error messages that are displayed to the user when iManager encounters an error.
portal\modules\module name\help	Plug-in help files. This directory contains a subdirectory named with the ISO language code for each language for which help files exist.
portal\modules\module name\images	Images used by plug-ins and the iManager framework. This directory contains a subdirectory named with the ISO language code for each language for which image files exist.
portal\modules\module name\install	Registration files for plug-ins that are registered in eDirectory for role-based usage.
portal\modules\module name\javascripts	JavaScript* files that can be inserted into JSPs.
portal\modules\module name\plugins	Registration files for plug-ins that are for global, non-role-based usage.
portal\modules\module name\skins	Plug-in UI pages and JSPs. A skin modifies the look and feel of the plug-in. There is a subdirectory for each skin supported by the plug-in, and each skin directory contains a subdirectory for each device supported by the plug-in. Must have, at a minimum, a default skin directory, and each skin must have a default device directory, each named "default."
WEB-INF	Configuration and application files. The content of this directory is accessible only by the Web application.
WEB-INF\bin	Binary files required by iManager.
WEB-INF\classes	Plug-in Java class files that are not in Java archive (JAR) files. The classes in this directory are automatically added to the classpath.
WEB-INF\lib	JAR files required by plug-ins and the iManager framework. If you archive your plug-in Java class files into a JAR file, you should copy that JAR file to this directory. The JAR files in this directory are automatically added to the iManager classpath.
WEB-INF\logs	Log files generated by iManager.

2.3 Role-Based Services

Role Based Services (RBS) is a set of extensions to the eDirectory schema. RBS defines several object classes and attributes that provide a mechanism for administrators to grant a user access to management tasks based on the user's role in the organization. This gives users access to only those tasks that the users need to perform. RBS grants only the rights necessary to perform assigned tasks.

Furthermore, users are associated with roles in a specified scope—a container in the tree in which the user has the requisite permissions to perform a task. A role requires this ternary association of role, members, and scope to be complete. The following figure illustrates the relationship of roles, members, and scopes.

Figure 2-3 *The relationship of roles, members, and scope.*



An RBS role object creates an association between users and tasks. An administrator grants a user access to a task by making the user a member of the role to which the task is assigned.

A user can be assigned to a role in the following ways:

- ♦ Directly
- ♦ Through group and dynamic group assignments. If a user is a member of a group or a dynamic group that is assigned to a role, then the user has access to the role.
- ♦ Through organizational role assignments. If a user is an occupant of a organizational role that is assigned a role, then the user has access to the role.
- ♦ Through container assignment. A user object has access to all of the roles that its parent container is assigned. This could also include other containers up to the root of the tree.

A user can be associated with a role multiple times, each with a different scope.

For information about RBS directory objects, see [Section 12.3, “Role-Based Services Directory Objects,” on page 111](#).

2.4 Module ID

Because iManager has an extendable architecture, its directories must be organized in a way that enables iManager to locate and distinguish the files, tasks, and session keys of plug-ins. This is accomplished through use of modules. You should group related plug-ins and other files into modules, and give each module a unique ID. The module ID is used to group module files within the `webapps/nps/portal/modules` directory. It is also used as a prefix to task IDs, and as a prefix to session keys.

Core iManager modules include framework (fw), base, and dev. The fw module contains the core iManager framework content and should not be modified by plug-in developers. The base module contains the base eDirectory administration content provided by Novell. The dev module contains shared “widgets” that can be used by developers to easily add features and common user interface elements.

2.5 Connection Modes

When you connect to iManager, you are connected in one of the modes described in the following table:

Table 2-2 *iManager connection modes*

Mode	Description
Unrestricted	All roles and tasks that are in the file system on the Web server are displayed to all users regardless of whether the user has sufficient rights to use them. This is the mode used when the iManager SDK is first run.
Assigned	Users see only the roles to which they have been assigned. To enable connections in this mode, you must configure set up RBS, and assign users to roles. For information, see “Setting Up Role-Based Services” on page 14 .
Collection Owner	Users see the roles and tasks that are installed into any collections of which they are the owner. To enable connections in this mode, you must follow the same procedure as for Assigned mode.

2.6 iManager Plug-ins

Because of changes to class structure and organization, plug-ins for previous versions of iManager must be recompiled to work with iManager 2.7. The iManager 2.7 Web site contains all currently available plug-ins, and is regularly updated with additional plug-ins as they become available.

WARNING: Do not install older plug-ins from a local drive into iManager 2.7. While the plug-in might install, it will not run and can be very difficult to remove.

This section explains how to create task plug-ins. It covers the following topics:

- ♦ [Section 3.1, “Overview,” on page 23](#)
- ♦ [Section 3.2, “Creating a Task: An Example,” on page 23](#)
- ♦ [Section 3.3, “Tips for Creating Task Java Classes,” on page 30](#)
- ♦ [Section 3.4, “Tips for Creating JSPs,” on page 32](#)
- ♦ [Section 3.5, “Creating Registration Files,” on page 35](#)
- ♦ [Section 3.6, “Specifying Conditions for Task Execution,” on page 37](#)
- ♦ [Section 3.7, “Launching Tasks and Delegating to Tasks,” on page 38](#)
- ♦ [Section 3.8, “Task Chaining,” on page 41](#)
- ♦ [Section 3.9, “Extending the Object Management Tasks,” on page 43](#)
- ♦ [Section 3.10, “Enabling a Task for the Object View,” on page 45](#)
- ♦ [Section 3.11, “Using the AdminNamespace,” on page 48](#)

3.1 Overview

Developing a task plug-in involves the following general steps, which are explained in more detail later in this section:

- 1 Create a Java class that extends `com.novell.emframe.dev.Task`.
- 2 Override the `execute(TaskContext context, Properties resultStrings)` method.
- 3 Compile your Java class.
- 4 Create JSPs for the UI. If the task has more than one state, you must provide a JSP for each state unless the JSP is flexible enough to adapt to each state.
- 5 Create a registration file to register your task with `iManager`.
- 6 Copy the Java class files, JSPs, and the registration file to the appropriate directories within the `iManager` directory structure.
- 7 Start `iManager` and test your task.

3.2 Creating a Task: An Example

To show how to create a task plug-in, we will use the `BasicTaskExample` sample task that is included in the `iManager 2.7 SDK`. The source files for the sample plug-ins are in `SDK_HOME/samples/src`. For information on how to build the sample plug-ins, see [Section 1.4, “Building the Sample Plug-Ins,” on page 15](#).

This section contains information on the following:

- ♦ [Section 3.2.1, “Creating the Java Class,” on page 24](#)
- ♦ [Section 3.2.2, “Creating the UI,” on page 26](#)
- ♦ [Section 3.2.3, “Creating the Registration File,” on page 28](#)

- ♦ [Section 3.2.4, “Localizing the Task,” on page 29](#)
- ♦ [Section 3.2.5, “Deploying the Task,” on page 30](#)

3.2.1 Creating the Java Class

To create a Java class for your task, extend the `com.novell.emframe.dev.Task` class and override its `execute(TaskContext context, Properties resultStrings)` method. The `BasicTaskExampleTask.java` sample task shows how to do this:

```
package com.company.plugins;

import com.novell.emframe.dev.*;
import java.util.Properties;

public class BasicTaskExampleTask extends Task
{
    public boolean execute(TaskContext context, Properties resultStrings)
    {
        setUIPage("sdk/BasicTaskExampleTemplate.jsp");
        return true;
    }
}
```

The `execute` method calls the inherited `setUIPage(String UIPage)` method to tell `iManager` which JSP to load in the content area of the `iManager` UI. You can add conditional statements to specify different JSPs for different task states. Tasks might have states that correspond with steps in the flow of the UI. For example, suppose a task has an initial page that asks for the name of an object that is to be created. After entering a name and clicking a `Create` button, users proceed to the next step in which they are presented with either a confirmation message or a message indicating that an error occurred. A `Cancel` button causes the task to end without creating the object.

The `execute` method should

- ♦ Check the value of the `nextState` parameter from the HTTP request, if the task has more than one state.
- ♦ Perform any processing required for the current state.
- ♦ Set the next JSP by calling `setUIPage(String UIPage)`.

The following example shows how you might modify the `BasicTaskExampleTask.java` file so that it can handle two states:

```
package com.company.plugins;

import com.novell.emframe.dev.*;
import java.util.Properties;
import javax.servlet.http.HttpServletRequest;

public class BasicTaskExampleTask extends Task
{
    private HttpServletRequest req = null;

    public boolean execute(TaskContext context, Properties resultStrings)
    {
        req = context.getRequest();
        String nextState = req.getParameter(eMFrameConsts.NEXTSTATE);
```



```

        if (nextState.equalsIgnoreCase(eMFrameConsts.INITIALSTATE))
        {
            setUIPage("sdk/BasicTaskExampleTemplate.jsp");
        }
        else if(nextState.equalsIgnoreCase("state2"))
        {
            req.setAttribute("paramName", "paramValue");
            setUIPage("sdk/BasicTaskExampleTemplate2.jsp");
        }
        return true;
    }
}

```

In the preceding example, the `eMFrameConsts.NEXTSTATE` parameter is set to the current state of the task. If the task is in its initial state, the initial JSP is set. If the task is in state 2, the JSP for state 2 is set. Note that the initial JSP must specify the next state of the task. This is typically done using a hidden input field named `nextState`:

```
<INPUT type="hidden" name="nextState" value="state2">
```

Use the methods of the `HttpServletRequest` class, such as `getAttribute(String name)` or `getParameter(String name)`, to retrieve data from the JSP, as shown in the preceding example. To send data to the JSP, use `HttpServletRequest.setAttribute(String name, Object o)`.

The preceding examples do little more than display a UI. To add the business logic required by your task, you might add methods to your class or create other classes. The following example adds a `doSomething` method to the `BasicTaskExampleTask` sample task:

```

package com.company.plugins;

import com.novell.emframe.dev.*;
import java.util.Properties;
import javax.servlet.http.HttpServletRequest;

public class BasicTaskExampleTask extends Task
{
    private HttpServletRequest req = null;

    public boolean execute(TaskContext context, Properties resultStrings)
    {
        req = context.getRequest();
        String nextState = req.getParameter(eMFrameConsts.NEXTSTATE);

        if (nextState.equalsIgnoreCase(eMFrameConsts.INITIALSTATE))
        {
            setUIPage("sdk/BasicTaskExampleTemplate.jsp");
        }
        else if(nextState.equalsIgnoreCase("state2"))
        {
            String value = doSomething();
            req.setAttribute("paramName", value);
            setUIPage("sdk/BasicTaskExampleTemplate2.jsp");
        }
        return true;
    }

    private String doSomething()

```

```

{
    String result = "paramValue";
    return result;
}
}

```

3.2.2 Creating the UI

To create the UI for a task, create one or more JSPs. Use the JSPs to display forms, instructions, and the results of the processing performed by your task. Avoid putting the business logic of your task in the JSP. The *SDK_HOME/samples/web/portal/modules/skd/skins/default/devices/default/BasicTaskExampleTemplate.jsp* file shows the JSP for the BasicTaskExample sample task:

```

<%@ page pageEncoding="utf-8" contentType="text/html; charset=utf-8" %>

<%@ taglib uri="/WEB-INF/iman.tld" prefix="iman" %>
<%@ taglib uri="/WEB-INF/c.tld" prefix="c" %>
<%@ taglib uri="/WEB-INF/x.tld" prefix="x" %>

<iman:stringtable bundle="DevResources"/>
<iman:stringtable bundle="FwResources"/>

<HTML>
<HEAD>
    <TITLE><iman:string key="ProductName"/></TITLE>
    <LINK rel="stylesheet" href="<c:out value="\${ContextPath}" />/portal/modules/
dev/css/hf_style.css">
    <iman:eMFrameScripts/>
</HEAD>

<BODY>
    <iman:taskHeader title="Basic Task Example"/>
    This task simply exposes the minimum code needed to create an iManager task,
specifically: Java code, JSP template, XML install file, and a Properties file for
string management.
    <br>
    <br>
    The code for this task can be found in the "BasicTaskExample" folder found in the
iManager SDK.
    <BR><BR>
    <iman:bar/>
    <iman:button key="OK" type="input"/>
    <iman:cancelBtn/>
</BODY>
</HTML>

```

This example does nothing more than display text. It uses the iman tag library to insert common UI elements: a header, a horizontal rule, an OK button, and a Cancel button.

The following example shows how you might modify BasicTaskExampleTemplate.jsp to enable the BasicTaskExample task to handle two states:

```

<%@ page pageEncoding="utf-8" contentType="text/html; charset=utf-8" %>

<%@ taglib uri="/WEB-INF/iman.tld" prefix="iman" %>
<%@ taglib uri="/WEB-INF/c.tld" prefix="c" %>
<%@ taglib uri="/WEB-INF/x.tld" prefix="x" %>

<iman:stringtable bundle="DevResources"/>
<iman:stringtable bundle="FwResources"/>

<HTML>
<HEAD>
  <TITLE><iman:string key="ProductName"/></TITLE>
  <LINK rel="stylesheet" href="<c:out value="{ContextPath}" />/portal/modules/
dev/css/hf_style.css">
  <iman:eMFrameScripts/>
</HEAD>

<BODY>
  <iman:taskHeader title="Basic Task Example - State 1"/>
  This is the UI for the first state of the task.
  <BR><BR>
  <form name="next" method="post" action="webacc?Autoparse=true"
enctype="multipart/form-data">
    <INPUT type="hidden" name="error" value="dev.GenErr">
    <INPUT type="hidden" name="User.context" value="<c:out value="{User.context}"
/>">
    <INPUT type="hidden" name="taskId" value="sdk.BasicTaskExampleTask">
    <INPUT type="hidden" name="nextState" value="state2">
    <iman:bar/>
    <iman:button key="Next" type="input"/>
    <iman:cancelBtn/>
  </form>
</BODY>
</HTML>

```

This version encloses Next and Cancel buttons in a form element, and it uses hidden input fields to pass data back to the task Java class. The nextState field identifies the next state of the task. The Java class uses the value of the nextState parameter to determine which state it is in so it can display the JSP for the current state.

The following example is the JSP for the second state of the task, which displays the data passed to it from the Java class:

```

<%@ page pageEncoding="utf-8" contentType="text/html; charset=utf-8" %>

<%@ taglib uri="/WEB-INF/iman.tld" prefix="iman" %>
<%@ taglib uri="/WEB-INF/c.tld" prefix="c" %>
<%@ taglib uri="/WEB-INF/x.tld" prefix="x" %>

<iman:stringtable bundle="DevResources"/>
<iman:stringtable bundle="FwResources"/>

<HTML>
<HEAD>
  <TITLE><iman:string key="ProductName"/></TITLE>
  <LINK rel="stylesheet" href="<c:out value="{ContextPath}" />/portal/modules/
dev/css/hf_style.css">
  <iman:eMFrameScripts/>

```

```

</HEAD>

<BODY>
  <iman:taskHeader title="Basic Task Example - State 2"/>
  This is the UI for the second state of the task.
  <BR><BR>
  Param: <c:out value="\${paramName}"/>
  <BR><BR>
  <iman:bar/>
  <iman:button key="OK" type="input"/>
</BODY>
</HTML>

```

The JSP for the second state uses the `c` tag library to print the value of the `paramName` parameter.

3.2.3 Creating the Registration File

Tasks and other plug-ins must be registered with iManager. You register a plug-in by creating an XML file that describes the plug-in and placing the file in the appropriate directory. A single registration file can contain descriptions for multiple tasks. The registration file for the sample plug-ins is `SDK_HOME/samples/web/portal/modules/sdk/install/iManagerCodeExamplesInstall.xml`:

```

<install>
  <module>
    <id>sdk</id>
    <version>2.0.1</version>
    <required-version>2.7.0</required-version>
    <resource-properties-file>com.company.plugins.iManagerCodeExamplesResources</
resource-properties-file>
    <display-name-key>iManagerCodeExamples.InstallDisplayName</display-name-key>
    <description-key>iManagerCodeExamples.Description</description-key>
  </module>
  <role>
    <id>iManager Code Examples</id>
    <version>2.0.0</version>
    <display-name-key>iManagerCodeExamples.RoleDisplayName</display-name-key>
    <resource-properties-file>com.company.plugins.iManagerCodeExamplesResources</
resource-properties-file>
  </role>
  <task>
    <id>sdk.BasicTask</id>
    <version>2.0.0</version>
    <required-version>2.7.0</required-version>
    <class-name>java.com.company.plugins.BasicTaskExampleTask</class-name>
    <display-name-key>BasicTaskExample.TaskDisplayName</display-name-key>
    <description-key>BasicTaskExample.Description</description-key>
    <resource-properties-file>com.company.plugins.iManagerCodeExamplesResources</
resource-properties-file>
    <role-assignment>iManager Code Examples</role-assignment>
  </task>

  . . . (Other tasks removed)

</install>

```

This registration file defines a module, a role, and the sample tasks. The definitions use the `<id>` tag to specify the ID of the module, role, and tasks. The task definitions use the ID of the role (“iManager Code Examples”) as the value of the `<role-assignment>` tag to specify that the tasks are assigned to the role:

```
<role-assignment>iManager Code Examples</role-assignment>
```

The task ID should use the module ID as a prefix to its ID. For the `BasicTaskExample` task, the ID is:

```
<id>sdk.BasicTask</id>
```

The module ID is the name of the directory under `SDK_HOME/tomcat/webapps/nps/portal/modules` where iManager expects to find files for plug-ins that are part of the module. For more information about modules, see [Section 2.4, “Module ID,” on page 21](#).

The definition of the `BasicTaskExample` task includes several other tags besides ID. The `<id>`, `<version>`, `<required-version>`, and `<class-name>` tags are required. The `<version>` tag is used to specify the task version. The `<required-version>` tag specifies the version of iManager that the task requires. The `<class-name>` tag identifies the Java class for the task.

For more information about registration files, see [Section 3.5, “Creating Registration Files,” on page 35](#). For more information about the available elements for registration files, see [Section 12.2, “XML Schema for Installation and Registration Files,” on page 111](#).

3.2.4 Localizing the Task

To localize your task, you need to

- ♦ Create properties files that contain translated strings.
- ♦ Specify the name of the default properties file in the task registration file.

The registration file for `BasicTaskExample` includes a `<resource-properties-file>` element, which identifies a properties file that contains strings used by the task:

```
<resource-properties-file>com.company.plugins.iManagerCodeExamplesResources</resource-properties-file>
```

Multiple plug-ins can share a properties file and the strings it contains. The `SDK_HOME/samples/resources/com/company/plugins/iManagerCodeExamplesResources.properties` file contains strings for most of the SDK sample plug-ins. The strings in the properties file are in the default language for the task, organized in key-value pairs. For example, `iManagerCodeExamplesResources.properties` contains the following entry:

```
BasicTaskExample.TaskDisplayName=Basic Task Example
```

You can insert this string in your JSP by referencing the key:

```
<iman:string key="BasicTaskExample.TaskDisplayName"/>
```

In fact, iManager uses the `<display-name-key>` and `<description-key>` elements of registration files to retrieve the task display name and description:

```
<display-name-key>BasicTaskExample.TaskDisplayName</display-name-key>
<description-key>BasicTaskExample.Description</description-key>
```

To retrieve localized strings, you must create a properties file for each language you want to support, including English. The keys are not translated, but the values change for each language. For each translated properties file, name it by appending an underscore and the language code to the name of the default properties file before the `.properties` extension. For example, a Spanish version of `iManagerCodeExamplesResources.properties` would be named `iManagerCodeExamplesResources_es.properties`.

iManager automatically selects the string value from the appropriate properties file, based on the language settings of the browser.

For a complete example, see the `StringLocalizationExample` sample task included in the SDK.

3.2.5 Deploying the Task

To deploy a task, you must copy the task files to several locations in the iManager directory structure, depending on the file type, as explained in the following table:

Table 3-1 *iManager task file locations*

File Type	Location
Java class and properties files	If your class and properties files are in a JAR file, copy the JAR file to <code>webapps/nps/WEB-INF/lib</code> ; otherwise, copy them to <code>webapps/nps/WEB-INF/classes</code> .
JSPs	Copy JSPs to <code>webapps/nps/portal/modules/module_id/skins/skin_name/devices/device_name</code> .
Registration files	Copy <code><install></code> registration files to <code>webapps/nps/portal/modules/module_id/install</code> . Copy <code><plugin></code> registration files to <code>webapps/nps/portal/modules/module_id/plugins</code> .
Other files	If you have other files, such as UI pages, JavaScript, and images, copy them to subdirectories of <code>webapps/nps/portal/modules/module_id</code> . Follow the pattern used by the modules that ship with iManager.

The sample tasks included in the SDK are deployed automatically when you use the build script included with the SDK.

3.3 Tips for Creating Task Java Classes

Be careful when using static variables. Static variables are shared across all sessions, so every logged-in user will use them. If you want to save information that can be shared by multiple tasks, use the authentication context. Authentication context attributes can be set using the `setAuthAttribute()` method, and retrieved using the `getAuthAttribute()` method on the task context. The task context can be retrieved from the `HttpServletRequest()` with the `getContext()` method. You can also store information in local variables on the task.

When using authentication context attributes, prepend your key with your module ID to avoid overwriting keys of other plug-ins. For example:

```
request.getContext().setAuthAttribute("acme.CurrentCount", "5");
```

When building a string with variables included, use the message formatting methods instead of concatenating. There is functionality in Java and JavaScript to do this correctly. In JavaScript use `formatMessage()`, found in `eMFrameScripts`. In Java use `eMFrameUtils.formatMessage()`.

If you want certain tasks to be performed whenever there is a switch-over from one running task to another, or if you want to do cleanup when your task is released for garbage collection, override the `release()` method of the task that is currently running.

When catching Exceptions, use `eMFrameUtils.setErrorMessage(Throwable t, TaskContext context)` to report them to the user. Currently, many plug-ins catch an exception, pull out the message, then return it to the user. This may work in some cases, but generally is not enough information. Instead, catch the exception, pass it to `setErrorMessage(t, context)`, and return false. `GenErr` must be the error template for this to work properly. This shows one of two things: If it is an `SPIException` it shows the NDS Error message with a link to get more details about the error. If it is another type of exception, it shows a general error message with a details link to show the exception class, the exception message, and a stack trace. This is preferable to showing just the error message because, for Java exceptions, this often does not make sense without the other information. For pages, throwing new `PageException(t)` provides similar functionality.

If your plug-in manages classes and attributes in `eDirectory` that are not part of the base schema, provide a graphic and translated names for these classes and attributes. To provide a graphic that shows up in the object selector, place a GIF file in the `SDK_HOME/tomcat/webapps/nps/portal/modules/moduleID/images/dir` directory. The GIF file must have the same name as the object class with all non-alphanumeric characters converted to underscores. For example, the graphic for the NDPS:Printer object is `SDK_HOME/tomcat/webapps/nps/portal/modules/dev/images/dir/NDPS_Server_Domain.gif`.

To provide translated names, create an XML file in the plugins directory that specifies what resource bundle and what attributes and classes it translates, then put values in that resource bundle for those translations. The keys for the translated class and attributes should use underscores instead of non-alphanumeric characters and should begin with `ObjectType` or `Attribute`.

Example XML file:

```
<dir-translator>
<resource-properties-file>FwResources</resource-properties-file>
<object-type-name>AFP Server</object-type-name>
<object-type-name>Alias</object-type-name>
<object-type-name>NLS:Product Container</object-type-name>
  <object-type-name>NLS:License Certificate</object-type-name>
  <object-type-name>NLS:License Server</object-type-name>
  <attribute-name>L</attribute-name>
  <attribute-name>CN</attribute-name>
<attribute-name>Full Name</attribute-name>
<attribute-name>Surname</attribute-name>
</dir-translator>
```

Example resource file:

```
ObjectType.AFP_Server=AFP Server
ObjectType.Alias=Alias
ObjectType.NLS_Product_Container=License Product Container
ObjectType.NLS_License_Certificate=License Certificate
ObjectType.NLS_License_Server=License Service Provider
Attribute.L=Location
Attribute.CN=Common name
Attribute.Full_Name=Full name
Attribute.Surname=Surname
```

If your plug-in manages classes and attributes in eDirectory that are not part of the base schema, plug in to the iManager creator, deleter, move, and rename tasks. These allow you to plug in to the general create, delete, move and rename tasks and also allow you to make tasks that just create, delete, move or rename your specific object or set of objects. If you do not plug in this way, users cannot create your objects from the general create object task. For delete, rename, and move, it gives you the ability to do additional operations when the general tasks are used on your type of object. See [Section 3.9, “Extending the Object Management Tasks,” on page 43](#).

3.3.1 Encoding Data

You do not need to unencode parameters that are fetched from the `HttpServletRequest` object using the `getParameter` method. This is true without regard to the source of the parameter, whether using `<href>`, `<action>`, or `<input>` tags. The `com.novell.emframe.dev.eMFrameUtils` class provides `urlencode`, `urldecode`, `toDisplay`, and `toScript` methods, but we discourage encoding the data in your Java class unless there is no other way. You should perform all encoding in your JSP, as explained in [Section 3.4.1, “Encoding Data Using Tag Libraries,” on page 33](#).

3.4 Tips for Creating JSPs

When specifying merge and error UI Pages and other files on the file system, be sure to use the proper case so that your tasks can run on platforms that are case sensitive.

The `MVStringEditor` widget should be used in most cases where multiple strings or DNs are being edited. It works for browser and simple UI Pages. It has modes for showing on a single line and as a list box. It automatically shows the search when editing a DN list. It allows the user to type values for adding instead of forcing the user to search in DN mode. For more information, see [Section 6.3, “The MVStringEditor Widget,” on page 75](#).

To use bold text for part of a message when using `GenConf` and `GenErr`, use the `eMFrameUtils.setMessage()` method that takes the `plainText` and `boldText` parameters. For the `plainText` String, use `{0}` for the position of the bold text.

We suggest that all JSPs include the `<iman:eMFrameScripts />` tag from the iManager tag library at the top to make future changes more compatible. This file contains methods for encoding and decoding as well as other miscellaneous utility methods. It also includes method for packing an array of strings into one string to be sent across the wire and then unpacking them. The `com.novell.emframe.dev.eMFrameUtils` class contains `pack` and `unpack` methods you can use in your Java code.

3.4.1 Encoding Data Using Tag Libraries

Typically, a JSP retrieves, displays, sends, and otherwise deals with data, which often contains characters that are incompatible with the languages and protocols that are part of the iManager architecture. To avoid errors, you need to make sure that data is correctly converted or escaped. It is usually best to perform all data manipulation in the JSP and none in your Java code, thereby separating the display and use of the data from your business logic.

Novell recommends using tag libraries for accessing data in your JSPs. Tags such as `c:out` and `x:out` perform XML encoding automatically. So when you are displaying data in a table or on a page, the encoding is taken care of for you. For example:

```
<c:out value="${myData}" />
```

Encoding Data in URLs

When you insert data into a URL (for example, in an `href`), it must be properly encoded. To do this, turn off XML-encoding and use the `iman:urlEncode` tag to encode it. For example:

```
<a href="frameservice?myData=<iman:urlEncode><c:out value="${myData}"  
escapeXml="false" /></iman:urlEncode>"
```

Encoding Data in JavaScript

When you insert data into a JavaScript context, it must be properly encoded. To do this, turn off XML-encoding and use the `iman:toScript` tag to encode it. This automatically escapes backslashes, quotes, new lines, etc.

```
<script>  
  var a = "<iman:toScript ><c:out value="${myData}" escapeXml="false"/></  
  iman:toScript>";  
</script>
```

3.4.2 Encoding Data Using Java Methods

Novell discourages using Java code to encode data in your Java classes or JSPs. However, if circumstances require you to use Java methods to encode data, it is possible, although significantly more complex, than using tag libraries.

The `com.novell.emframe.dev.eMFrameUtils` class contains methods for translating data. These methods are described in the following table:

Table 3-2 *com.novell.emframe.dev.eMFrameUtils class data translation methods*

Utility Method	Description
<code>toTag(String)</code>	Translates strings for inclusion in the tag portions of your HTML code. Characters that are part of the HTML syntax are translated to entities like <code>&lt;</code> ; so that they do not confuse the HTML parser.
<code>urlEncode(String)</code> <code>urlEncode(String, String)</code> <code>urlEncode(String, PluginContext)</code>	Translates query parameters on the URL links for transmission across the network. Spaces are translated to plus signs (+), and other characters are translated into three-character sequences like "%xx."

Utility Method	Description
toDisplay(String)	Translates strings that are displayed to the user. New line characters are converted to , multiple spaces are converted to , and characters that are part of the HTML syntax are translated to entities like < so that they do not confuse the HTML parser.
toScript(String)	Escapes quotes, apostrophes, and backslashes in strings that are placed in JavaScript code. The toScript routine places a backslash in front of these characters.
xmlEncode(String)	Escapes characters that are part of the XML syntax. These characters are translated to entities like < so that they will work properly with XML parsers.

Encoding in JSP or HTML Code

The following table shows HTML attributes and other parts of an JSP/HTML document where dynamic data is often inserted. For each of these, the table shows the utility method that should be used to encode the data and example. In the examples, the data is represented by a String-type variable named var.

The following elements do not need to be encoded:

- ♦ taskId
- ♦ image names
- ♦ UI page names
- ♦ template names

Table 3-3 JSP/HTML document components that accept dynamic data

HTML	Method	Example
action=""	urlEncode()	<%= eMFrameUtils.urlEncode(var) %>
alt=""	toTag()	<%= eMFrameUtils.toTag(var) %>
href=""	urlEncode()	<%= eMFrameUtils.urlEncode(var) %>
onClick=""	toTag()	<%= eMFrameUtils.toTag(var) %>
onClick="javascript:"	toTag(toScript())	<%= eMFrameUtils.toTag(eMFrameUtils.toScript(var)) %>
onLoad=""	toTag()	<%= eMFrameUtils.toTag(var) %>
onLoad="javascript:"	toTag(toScript())	<%= eMFrameUtils.toTag(eMFrameUtils.toScript(var)) %>
src="URL"	urlEncode()	<%= eMFrameUtils.urlEncode(var) %>
src="directory path"	toTag()	<%= eMFrameUtils.toTag(var) %>
value="" tag	toTag()	<%= eMFrameUtils.toTag(var) %>
window.location=url	toScript()	<%= eMFrameUtils.urlEncode(var) %>
javascript var=""	toScript()	<%= eMFrameUtils.toScript(var) %>
Strings in HTML body	toDisplay()	<%= eMFrameUtils.toDisplay(var) %>

Encoding for JavaScript (Non-JSP)

If a string is included in a URL, it must be encoded before it is transmitted to the server. This causes special characters like commas, colons, quotation marks, plus signs, and spaces, which have special significance to the URL parser, to be encoded into a three-character string, such as %2B, that has no significance to the URL parser.

Some browsers cannot handle spaces in parameters, so you must encode parameters that include only spaces as well as strings that include the other special characters. All <href> and <action> data is automatically unencoded when it reaches the server. To encode for JavaScript, include eMFrameScripts in your header and call the encodeURIComponent() method. Use the iman tag library to include eMFrameScripts:

```
<%@ taglib uri="/WEB-INF/iman.tld" prefix="iman" %>
<HEAD>
    . . .
    <iman:eMFrameScripts/>
</HEAD>
```

eMFrameScripts also includes urlDecode, toDisplay(), and toScript() functions.

Do not use the standard JavaScript escape and unescape methods. They do not work when using UTF-8.

Posting Data Through <input> Fields

Information passed back and forth between the client and web server by way of form <input> fields (whether hidden or not) do not need to be URL encoded before transmission to the server because they are not automatically decoded on arrival at the web server. Only action, href, window.location, and src="url" tags need to be URL encoded.

Using JavaScript in href Tags

Different browsers are not consistent in the way they handle URL encoded strings when they are passed to JavaScript routines. Internet Explorer completely decodes strings before sending them to the JavaScript routine. Netscape does nothing with the strings. To avoid using browser-specific code, do not use the following syntax:

```
<a href="javascript:<functionName>('<%= Utils.urlEncode(var) %>')" >
```

Use the following syntax instead:

```
<a href="#" onClick="javascript:<functionName>('<%=
Utils.toTag(Utils.toScript(var)) %>');return false">
```

Using href="#" instructs the browser to not go anywhere when the link is accessed. The onClick processing is then used to decide what to do when the link is clicked.

3.5 Creating Registration Files

Tasks and other plug-ins must be registered with iManager to be available to users. You register a plug-in by creating an XML file that describes the plug-in and placing the file in the appropriate directory. iManager provides a task for simplifying the creation of registration files. For more information see [Section 3.5.1, "Using the Create XML Install File Task," on page 37](#).

A plug-in can be registered in one of two places: eDirectory or the file system. Plug-ins registered in eDirectory can take advantage of Role-Based Services to limit access to the plug-in tasks to those users who are assigned to the appropriate role. These types of plug-ins must be installed using the iManager Configuration Wizard. The registration files for these plug-ins require the `<install>` tag as the root element. They are stored in `SDK_HOME/tomcat/webapps/nps/portal/modules/moduleID/install`. These plug-ins usually contain only roles and tasks.

Plug-ins that are registered in the file system are available to anyone. They appear as property book pages in the ModifyObject task of the Object Management role. These global plug-ins are detected by iManager when it starts up, and they do not need to be installed. The registration files for file system plug-ins require the `<plugin>` tag as the root element. These files are stored in `SDK_HOME/tomcat/webapps/nps/portal/modules/moduleID/plugins`.

NOTE: If you have not set up RBS, you access all plug-ins in Unrestricted Access mode, which allows you to view all roles and tasks even if you do not have sufficient rights to use them. You must set up RBS if you want to use the Plug-In Studio. For instructions, see [“Setting Up Role-Based Services” on page 14](#). For more information about connection modes, see [Section 2.5, “Connection Modes,” on page 22](#).

The following example is part of an install file for a plug-in that is registered in eDirectory:

```
<install>
  <module>
    <id>base</id>
    <version>1.0</version>
    <required-version>2.7.0</required-version>
    <resource-properties-file>BaseResources</resource-properties-file>
    <description-key>EDirModuleDescription</description-key>
  </module>
  <role>
    <id>eDirectory Administration</id>
    <version>1.0</version>
    <display-name-key>eDirectoryAdministrationDisplayName</display-name-key>
    <resource-properties-file>BaseResources</resource-properties-file>
  </role>
  <task>
    <id>base.CloneObjects</id>
    <version>1.0</version>
    <required-version>2.7.0</required-version>
    <type>snapinTask</type>
    <class-name>com.novell.imanage.base.CloneObjectTask</class-name>
    <merge-template>base.CloneObject</merge-template>
    <display-name-key>CloneObjectTaskDisplayName</display-name-key>
    <description-key>CloneObjectsDescription</description-key>
    <error-template>dev.GenConf</error-template>
    <resource-properties-file>BaseResources</resource-properties-file>
    <role-assignment>eDirectory Administration</role-assignment>
    <rights-assignment>
      <attribute-name>[Entry Rights]</attribute-name>
      <privilege>Browse</privilege>
      <privilege>Write</privilege>
    </rights-assignment>
  </task>
</install>
```

```

    </task>
    .
    .
    .
</install>

```

The following is an example of an install file for a global plug-in:

```

<plugins>
  <task>
    <id>dev.Book</id>
    <class-name>com.novell.emframe.dev.Book</class-name>
  </task>
  <task>
    <id>dev.Page</id>
    <class-name>com.novell.emframe.dev.Page</class-name>
  </task>
  <task>
    <id>dev.Empty</id>
    <class-name>com.novell.emframe.dev.EmptyTask</class-name>
  </task>
</plugins>



```

For more information about the available elements for registration files, see [Section 12.2, “XML Schema for Installation and Registration Files,” on page 111](#).

3.5.1 Using the Create XML Install File Task

The iManager Base Content plug-in provides a task that simplifies the creation of plug-in registration files. It collects information about your plug-in, creates a registration file, and enables you to install the plug-in in the current tree. It can create new registration files or update existing files.

To access the Create XML Install File task:

- 1 Log in to iManager.
- 2 Click the Developer icon .
- 3 Click *iManager Development > Create XML Install File*.
- 4 Follow the Wizard steps. For help during the task creation process, click Help . When you have saved the registration file, click *Next*.
- 5 If you want to install the plug-in that you have just defined into the current tree, specify the collection into which the plug-in will be installed, then click *Install*.
- 6 Click *Finish* to complete the wizard.

3.6 Specifying Conditions for Task Execution

The Task class can allow conditional execution of the task using the `shouldRun(HttpServletRequest, String)` method inherited from `BaseGadgetInstance`.

To check conditions to determine whether the task should run, overwrite the `shouldRun` method. Be sure to call `super.shouldRun()` to let the system check device types. The `shouldRun` method is a static method, which means that it can be run before your task is instantiated, only when the class is loaded. The following is the declaration for `shouldRun`:

```
static public void shouldRun(HttpServletRequest req, String serviceName)
    throws Exception
```

3.7 Launching Tasks and Delegating to Tasks

An `iManager` task can launch another task in a separate window or delegate its area to another task. A task launched in a new window can close when the task returns. A delegate task takes over the area of the delegating task until the delegate task returns control to the delegating task, the parent, the `fw.HomePage` task, or any other task. For most situations, you will use task delegation.

3.7.1 The Launch Service and Launch Actions

The Launch service is a public service you can call to access the launch and delegation features from a URL. In the URL, you specify one of the following actions:

- ♦ “Launch” on page 38
- ♦ “Delegate” on page 38
- ♦ “ReturnFromLaunch” on page 38
- ♦ “ReturnFromDelegate” on page 39

Launch

This action causes a task to appear in a new window. It has the following syntax:

```
nps/servlet/portalservlet?NPService=LaunchService
&NPAction=Launch
&launch=serviceNameToLaunch
&launcher=serviceIdOfLauncher&lifecycle={New | Reuse | Reset | Existing |
Recreate}
```

Delegate

This action causes another task to take over the area of the task. It has the following syntax:

```
nps/servlet/portalservlet?NPService=LaunchService
&NPAction=Delegate
&delegate=serviceNameToLaunch
&launcher=serviceIdOfLauncher&lifecycle={New | Reuse | Reset | Existing |
Recreate}
```

ReturnFromLaunch

This action causes a launched task to return. It has the following syntax:

```
nps/servlet/portalservlet?NPService=LaunchService
&NPAction=ReturnFromLaunch
&returnID=serviceNameOfLaunchedService
```

ReturnFromDelegate

This action causes a delegate task to restore its area to a designated task. It has the following syntax:

```
nps/servlet/portalservlet?NPService=LaunchService
&NPAction=ReturnFromDelegate
&returnID=serviceNameOfDelegateService
```

3.7.2 Life Cycles of Launched and Delegate Tasks

The Launch and Delegate actions require that you specify the life cycle of the launched or delegate task in the URL using the lifecycle parameter. The lifecycle parameter determines how the task instance is instantiated and freed. The following table describes the possible values of lifecycle:

Table 3-4 Possible lifecycle values

Value	Description	Code Name	Code Value
New	Creates a new instance with the name of the service appended to a four-digit random number.	LIFECYCLE_TYPE_NEW	1
Reuse	Like New, but uses a pool to reuse the tasks. (Must support Reset to use the pool.)	LIFECYCLE_TYPE_REUSE	2
Reset	Uses the name of the service but calls Reset on the task. (Must support Reset to use this.)	LIFECYCLE_TYPE_RESET	3
Existing	Uses the existing instance in its current state.	LIFECYCLE_TYPE_EXISTING	4
Recreate	Creates a new instance, and any existing instance is freed for garbage collection.	LIFECYCLE_TYPE_RECREATE	5

3.7.3 Accessing Launching and Delegation Features Programmatically

The `com.novell.nps.gadgetManager.GadgetManager` class provides the following methods for programmatic access to the launching and delegation features:

launchGadget

```
public static GadgetInstance launchGadget(GadgetInstance launcher, PortalSession
session, HttpServletRequest req, String launchServiceName, int delegationType)
throws PortalException
```

delegateToGadget

```
public static GadgetInstance delegateToGadget(GadgetInstance delegator,
PortalSession session, HttpServletRequest req, String delegateServiceName, int
delegationType) throws PortalException
```

returnToLauncher

```
public static void returnToLauncher(GadgetInstance secondary, Object results,
PortalSession session)
```

returnToDelegator

```
public static void returnToDelegator(GadgetInstance secondary, Object results,
PortalSession session)
```

3.7.4 Launching and Delegation Methods in GadgetInstance

The `com.novell.emframe.dev.Task` class implements the following methods of `com.novell.nps.gadgetManager.GadgetInstance` that support task launching and delegation by developers:

Methods Related to Launching or Delegating:

- ♦ `processSecondaryGadgetResults`

Methods Related to Being Launched:

- ♦ `returnToPrimary`

Life Cycle-Related Methods:

- ♦ `reset`
- ♦ `release`
- ♦ `getLifecycleType`

3.7.5 Task Delegation Example

The JSP responsible for populating the Roles and Tasks frame uses task delegation. The task links set the target to the Content frame and cause the framework to delegate to the selected task and make the launcher the `fw.HomePage` task. The `fw.HomePage` task is not the task that delegates. Another task—the one responsible for populating the Roles and Tasks frame—is the delegating task. However, this task—the Roles and Tasks task—is setting up the delegation between the selected task and the `fw.HomePage` task.

For example, the Create User task has the following anchor tag:

```
<a target="Content"
href="frameservice?NPService=fw.LaunchService&NPAction=Delegate&delegate=base.CreateUser&launcher=fw.HomePage&lifecycle=Recreate">CreateUser</a>
```

Notice that this URL does not refer to the Roles and Tasks task. This is perfectly legal. Any task can launch any other task and designate any other task as the launcher.

Let's examine the elements of the URL specified in the anchor tag more closely:

```
NPService=fw.LaunchService
```

To launch or delegate, everything must go through `fw.LaunchService`. (`NPService=` is similar to `taskId=`, as used in previous versions of `iManager`.)


```
NPAction=Delegate
```

The value of NPAction can be either Delegate or Launch.

```
delegate=base.CreateUser
```

This parameter specifies the task you want to run. If NPAction=Launch, then you would use launch=base.CreateUser instead to specify the task to run in the new window.

```
launcher=fw.HomePage
```

With either NPAction=Delegate or NPAction=Launch, this parameter specifies which task is the parent task. When the delegate or launched task returns, control switches to the parent.

```
lifecycle=Recreate
```

The lifecycle parameter specifies how to create or find an existing instance of the task. For launching, it is usually best to use Recreate. The possible values are listed in [Section 3.7.2, “Life Cycles of Launched and Delegate Tasks,”](#) on page 39.

3.7.6 Closing the Window of a Launched Task

If you want your task started in a new window, and you want the window closed when the task returns, make fw.LaunchService the launcher. fw.LaunchService has code that closes the window if it is the parent. For example:

```
<a target="_blank"
href="frameservice?NPService=fw.LaunchService&NPAction=Launch&launch=Task1&launcher=fw.LaunchService&lifecycle=Recreate">Task2</a>
```

If you do not make fw.LaunchService the launcher, the launching task is responsible for closing the window. This might be desirable if the launching task needs to get results back from the launched task before closing the window itself or delegating to some other task.

3.7.7 More Examples

For another example of task delegation and launching, see the *SDK_HOME\tomcat\webapps\nps\portal\modules\debug\skins\default\devices\default\TaskLaunchingDemo.jsp* file.

3.8 Task Chaining

You can create a task that automatically executes upon completion of another task—a process called task chaining. Task chaining can help you avoid duplicating work when you have a task that already accomplishes part of what you need your task to do. You can chain a task that does the remaining work to an existing task. For example, suppose you need a task that creates a user and assigns the user to an organizational unit. iManager includes the Create User task for creating users. You can create a task that assigns a user to an organizational unit and chain it to the Create User task.

To chain tasks you need to

- ♦ Add code to the initial task to pass objects to the chained task.

- ♦ Add code to the chained task to get objects from the initial task.
- ♦ Include an entry for the chained task in a registration file using the <chained-task> element.

3.8.1 Setting Objects in the Initial Task

To pass objects from the initial task, you need to call `com.novell.nps.gadgetManager.JobData.put(Object key, Object value)` or `com.novell.nps.gadgetManager.JobData.setObjectNames(String[] names)` in the first task in the chain. For example:

```
import com.novell.nps.gadgetManager.JobData;

JobData data = JobData.getJobData(this);
if (null != data)
{
    data.put(ANYKey, anyValue);
}
```

The Create User task provided by iManager is already enabled for serving as the initial task in a task chain. The name of the User object created is set using the following code:

```
JobData data = JobData.getJobData(this);
if (null != data)
{
    data.setObjectNames(new String[]{targetOE.getFullName()});
}
```

3.8.2 Getting Objects in the Chained Task

To get an object reference from the initial task, call `com.novell.nps.gadgetManager.JobData.getJobData(GadgetInstance gadget)`:

```
JobData data = JobData.getJobData(this);
if (null != data)
{
    String [] objNames = data.getObjectNames();
}
```

3.8.3 Defining the Chained Task in a Registration File

Use the <chained-task> element to define the chained task in a registration file:

```
<plugin>
  <chained-task>
    <id>moduleId.ChainedTaskId</id>
    <chaining-initial-task-id>base.CreateUser</chaining-initial-task-id>
    <order>300</order>
    ... (Everything else your task needs, just like a normal task,
        <class-name>, <display-name>, etc.)
    </chained-task>
  </plugin>
```

The `<chained-task>` element is identical to the `<task>` element, except that it allows for an additional child element, `<chaining-initial-task-id>`, which is the element you use to specify the task that the chained task will follow. The `<order>` element specifies the order of appearance of tasks in a role.

3.9 Extending the Object Management Tasks

The iManager Base Content module provides a role called eDirectory Administration. If you are assigned to the role, you have access to tasks that allow you to create, delete, move, and rename eDirectory objects. These tasks are generic: they work on almost any type of object and they operate on each object in the same way. However, you can extend these tasks to customize the management action for certain types of objects. For example, you could create a task that deletes groups only, keeps a log of deletions, and sends a notice when the task is used to delete a group. You can also disallow deletion of certain types of objects.

The object management tasks that you can extend are:

- ♦ Create Object (com.novell.emframe.dev.CreateObjectTask)
- ♦ Delete Object (com.novell.emframe.dev.Deletor)
- ♦ Move Object (com.novell.emframe.dev.Move)
- ♦ Rename Object (com.novell.emframe.dev.Rename)

3.9.1 Registering an Object Type for the Create Object Task

The Create Object task only creates objects of types that are registered with iManager for the creator task. This is different from the behavior of the Delete, Move, and Rename tasks, which operate on all objects unless you specify otherwise (see [Section 3.9.2, “Extending the Delete, Move, and Rename Tasks,”](#) on page 44).

iManager provides a generic creator task, com.novell.emframe.fw.GenericCreator. The generic creator task shows the user the default fields for the mandatory attributes of the object type that the user has chosen to create. If you need to enable the create object task for a class that is not registered, and all you need to do is collect information for the mandatory attributes, you can enable your class by simply adding an `<object-creator>` section to the `SDK_HOME/tomcat/webapps/nps/portal/fw/plugins/creator.xml` file using com.novell.emframe.fw.GenericCreator as the value of the `<class-name>` element, as shown in the following example:

```
<object-creator>
  <id>dbm.CreateServer</id>
  <version>1.0</version>
  <required-version>2.7</required-version>
  <class-name>com.novell.emframe.fw.GenericCreator</class-name>
  <object-type-name>Database Server</object-type-name>
</object-creator>
```

If the generic creator doesn't work for you, create a task that creates objects the way you need it to work then specify the class as the value of the `<class-name>` element in `SDK_HOME/tomcat/webapps/nps/portal/fw/plugins/creator.xml`. This causes the creator to launch your task instead of the generic creator. For example:

```

<object-creator>
  <id>dbm.CreateServer</id>
  <version>1.0</version>
  <required-version>2.7.0</required-version>
  <class-name>com.company.imanager.CreateServer</class-name>
  <object-type-name>Database Server</object-type-name>
</object-creator>

```

3.9.2 Extending the Delete, Move, and Rename Tasks

- 1 Create a Java class that extends the class of the task you want to use.

For example, if you want to use the Delete Object task, your class must extend `com.novell.emframe.dev.Deletor`.

- 2 In your class, implement the `doDelete(ObjectEntry, TaskContext)`, the `doRename(ObjectEntry, String, TaskContext)`, or the `doMove(ObjectEntry, ObjectEntry, TaskContext)` method to perform the desired action. In the appropriate method, either complete the operation and return or throw a `PluginException` constructed with the header and body of the message to display to the user.

NOTE: The `doXXX()` method names may be overloaded, so make sure that you pass in the correct set of arguments in order to get the result you are expecting.

- 3 Register your plug-in with iManager by adding an `<object-deletor>`, `<object-move>`, or `<object-rename>` section to the XML file in `SDK_HOME/tomcat/webapps/nps/portal/modules/fw/plugins` that corresponds to the task to which you are plugging in. Use the `<object-type-name>` element to designate object types that your task works with, and use the `<class-name>` element to specify the class name of your Java class. For example, the following `<object-deletor>` section registers a deletor plug-in when added to `deletor.xml`:

```

<object-deletor>
  <id>dbm.DeleteServer</id>
  <version>1.0</version>
  <required-version>2.7.0</required-version>
  <object-type-name>Database Server</object-type-name>
  <class-name>com.company.imanager.DeleteServer</class-name>
</object-deletor>

```

The following XML registers a move plug-in when added to `move.xml`:

```

<object-move>
  <id>dbm.MoveServer</id>
  <version>1.0</version>
  <required-version>2.7.0</required-version>
  <object-type-name>Move Server</object-type-name>
  <class-name>com.company.imanager.MoveServer</class-name>
</object-move>

```

- 4 Create a registration file for the plug-in. Specify the base task class (for example, `DeleteObjectTask`) and the object type you are working with.
- 5 To veto performance of the action on an object type, throw an exception in the `doXXX()` method with the message to be displayed to the user when the user tries to act on that object type.

3.9.3 Disallowing Operation of an Object Management Task on an Object Type

You can extend the Create, Delete, Move, or Rename tasks to disallow, or “veto,” the operation of an object management task on objects of a particular type. However, iManager provides classes that perform a basic veto on the Create, Delete, Move, and Rename tasks. You can register an object type using one of these veto classes to disallow operation of the task on that object type. For example, the following section in `deletor.xml` disallow deletion of the Database Server type:

```
<object-deletor>
  <id>dbm.DeleteServer</id>
  <version>1.0</version>
  <required-version>2.7.0</required-version>
  <object-type-name>Database Server</object-type-name>
  <class-name>com.novell.emframe.dev.VetoDeletor</class-name>
</object-deletor>
```

The veto classes are as follows:

- ♦ `com.novell.emframe.dev.VetoCreatorTask`
- ♦ `com.novell.emframe.dev.VetoDeletor`
- ♦ `com.novell.emframe.dev.VetoMove`
- ♦ `com.novell.emframe.dev.VetoRename`

If the veto plug-ins provided with iManager do not suit your needs, you can create your own. For example, you might want to create a plug-in that sends a notice to an administrator when a user attempts to delete objects of a certain type.

To veto the creator task, create a custom task that extends `Task` and register it as explained in [Section 3.9.1, “Registering an Object Type for the Create Object Task,” on page 43](#). The task should implement the `execute` method and return `false` after displaying an error message.

To veto the deletor, move, or rename tasks, extend the task for the action you want to veto (Delete, Move, or Rename) and register the task, as explained in [Section 3.9.2, “Extending the Delete, Move, and Rename Tasks,” on page 44](#). In the `doXXX()` method, throw a `PluginException` with the message to be displayed to the user when the user tries to act on the object type for which your class is registered.

3.10 Enabling a Task for the Object View

The Object view provides tools so users can manage by first selecting one or more objects, then selecting the task to perform. When you select objects in the Object view, iManager provides a way to view the list of available tasks for that object. To do this, tasks must be able to accept object names programmatically, rather than through the UI; inform the iManager framework on which object types it can work; and specify whether they can accept more than one object at a time.

To configure your tasks to work properly with the Object view requires the following:

- ♦ [Section 3.10.1, “Register a Task to Work with Specific Object Types,” on page 46](#)
- ♦ [Section 3.10.2, “Retrieve Objects from iManager,” on page 47](#)

3.10.1 Register a Task to Work with Specific Object Types

To specify the object types on which your task can operate, add the names of the object classes to the task registration file using the `<object-type-name>` element, as shown in the following example:

```
<task>
  <id>base.ClearIntruderLockout</id>
  <version>2.0.0.0</version>
  <required-version>2.7.0</required-version>
  <type>snapinTask</type>
  <class-name>java:com.novell.emframe.base.ClearLockoutTask</class-name>
  <merge-template>base.ClrLock</merge-template>
  <error-template>dev.GenFatal</error-template>
  <description>This task clears the intruder lockout flag</description>
  <resource-properties-file>BaseResources</resource-properties-file>
  <display-name-key>ClearLockoutTaskDisplayName</display-name-key>
  <object-type-name>User</object-type-name>
  <rights-assignment>
    <attribute-name>Locked By Intruder</attribute-name>
    <privilege>Supervisor</privilege>
  </rights-assignment>
  <rights-assignment>
    <attribute-name>Login Intruder Attempts</attribute-name>
    <privilege>Supervisor</privilege>
  </rights-assignment>
  <role-assignment>Help Desk Management</role-assignment>
  <gadget-assignable>>false</gadget-assignable>
  <frame-type>Full</frame-type>
</task>
```

In this example, the `<object-type-name>User</object-type-name>` line enables the task for User objects. All eDirectory classes and auxiliary classes are valid values. The special types, [root], [containers], and [leafs] are also valid types. Specifying [root] enables a task to operate on the root of the tree, specifying [containers] enables a task to operate on all container-type objects, and specifying [leafs] enables a task to operate on all leaf-type objects.

If your task operates on multiple object types, use multiple `<object-type-name>` elements in the task registration file. For example:

```
<task>
  . . .
  <object-type-name>User</object-type-name>
  <object-type-name>Group</object-type-name>
  . . .
</task>
```

If your task can modify multiple objects at the same time, specify that the task is enabled for multiple-object operations (moo) using the `<url-param>` element. For example:

```
<task>
  . . .
  <url-param>
    <param-key>mooEnabled</param-key>
    <param-value>true</param-value>
  </url-param>
  . . .
</task>
```

3.10.2 Retrieve Objects from iManager

In the Object view, users can select one or more objects. The selected objects are passed to the task as a packed string in the `targetNames` parameter.

Single Object Selection

If your task works on only a single object, use the `com.novell.emframe.dev.eMFrameUtils.getSingleTarget` method to get the name of the selected object from iManager. If the object was not selected in the Object view, `getSingleTarget` returns an empty string.

Typically the first screen of a task allows the user to specify the object on which the task should operate. If the object name is the only information collected on the first screen, you should skip the first screen, as shown in the following example:

```
public boolean execute(TaskContext context, Properties resultStrings)
{
    this.context=context;
    this.resultStrings=resultStrings;
    req=context.getHttpServletRequest();

    String nextState=req.getParameter(eMFrameConsts.NEXTSTATE);
    String targetName = eMFrameUtils.getSingleTarget(req);
    if(targetName.length() > 0)
    {
        nextState = "doClearLockout";
    }

    if(nextState.equalsIgnoreCase(eMFrameConsts.INITIALSTATE))
    {
        return showInitialForm();
    }
    else if(nextState.equalsIgnoreCase("doClearLockout"))
    {
        if(targetName.length() == 0)
        {
            targetName = req.getParameter("single");
        }
        if (doClearLockout(targetName))

    {

        setUIPage("dev/GenConf.jsp");

        return true;
    }
    else

    {

        setUIPage("dev/GenFatal.jsp");

        return true;
    }
}
```

```

    }

    throw new RuntimeException("Task received invalid nextState:"+nextState);
}

```

In this example, the value returned by `getSingleTarget(targetName)` is used to determine whether the object has already been selected by testing whether the length of the name is greater than zero. If so, the `nextState` is set to “doClearLockout,” which clears the lockout for the specified object without stopping at the first screen, where the object is specified. Because the object was selected in the Object view, the first screen is unnecessary. If the Object view was not used (`targetName.length() == 0`), the task displays the first screen, and the `getParameter` method returns the object selected.

If the first screen collects information in addition to the object name, you should display the first screen, but populate the object name field with the name of the object selected in the Object view. For a task that operates on a single object at a time, you can accomplish this by calling the `getSingleTarget` method within the JSP:

```

<INPUT type=text name="single"
value="<%= eMFrameUtils.toTag(eMFrameUtils.getSingleTarget(request)) %>" />

```

Multiple Object Selection

If your task operates on multiple objects, use the `TargetObjects` class to enumerate the selected objects, as shown in the following example:

```

String targetNames = req.getParameter("targetNames");
TargetObjects targetObjects = new TargetObjects(targetNames, context);

if(targetObjects.isMultiple())
{
    ObjectEntryEnumeration enum = targetObjects.getObjectEntryEnumeration();
    // do work with multiple objects
}
else
{
    ObjectEntry oe = targetObjects.getObjectEntry();
    // do work for single object
}

```

3.11 Using the AdminNamespace

`iManager` uses the `AdminNamespace` to access Novell eDirectory™ objects and objects in other directories. The `AdminNamespace` API documentation is included in the Javadoc-generated `iManager` Framework API documentation. See [Section 12.1, “iManager API Documentation,” on page 111](#).

Creating Property Books and Pages

A property book displays a group of pages that allows a user to view or modify the properties of an object or set of objects of the same type. Each page of the book has a tab that the user can click to switch to that page.

This section explains how to create property books and property book pages. It covers the following topics:

- ♦ [Section 4.1, “Creating a Property Book,” on page 49](#)
- ♦ [Section 4.2, “Creating a Property Book Page,” on page 50](#)

4.1 Creating a Property Book

Property books are types of tasks used to display and modify the attributes of a single object or a set of objects of the same type. Unlike tasks, where you decide what you want to do then follow a flow of UI screens until the task is accomplished, with property books you pick an object, view its attributes, and make changes to the attributes. The attribute data is displayed on pages. You can navigate among the pages by clicking on the tabs along the top of the property book or selecting the page in a list, depending on the type of device used to access iManager.

To create a property book, you need to:

- ♦ [Section 4.1.1, “Create the Java Class,” on page 49](#)
- ♦ [Section 4.1.2, “Create the Property Book Pages,” on page 49](#)
- ♦ [Section 4.1.3, “Create the XML Registration File,” on page 50](#)

4.1.1 Create the Java Class

To create a custom property book, create a Java class that extends `com.novell.emframe.dev.PropertyBook`.

NOTE: For property books that deal with eDirectory™ objects, use `com.novell.emframe.dev.DirPropertyBook` instead of creating your own class.

4.1.2 Create the Property Book Pages

Unlike tasks, property books do not use JSPs directly to create their user interfaces. Instead, property books display pages that are assigned to the property book. Pages are simple tasks with their own Java code and JSPs for their user interfaces. Pages can run outside of a book. For instructions on creating pages, see [“Creating a Property Book Page” on page 50](#).

4.1.3 Create the XML Registration File

Property books must be defined in an XML registration file. Store the registration file in a `\plugins` directory inside your module. When iManager reads the registration file, it uses the `<book>` entry to create an `rbsBook` object in eDirectory.

Property books must be defined in a registration file using the `<book>` element.

For example:

```
<book>
  <id>sdk.PropertyPageExample</id>
  <version>2.0.0</version>
  <required-version>2.7.0</required-version>
  <class-name>java:com.novell.emframe.dev.DirPropertyBook</class-name>
  <display-name-key>PropertyPageExample.BookDisplayName</display-name-key>
  <resource-properties-file>
    com.company.plugins.iManagerCodeExamplesResources
  </resource-properties-file>
  <object-type-name>User</object-type-name>
  <role-assignment>iManager Code Examples</role-assignment>
  <page-assignment>sdk.PPExampleTask1</page-assignment>
  <page-assignment>sdk.PPExampleTask2</page-assignment>
</book>
```

For complete descriptions of supported XML elements for the property book registration file, see [“XML Schema for Installation and Registration Files” on page 111](#).

4.2 Creating a Property Book Page

Property book pages are helpful for viewing and modifying attributes of an object. Unless you specify otherwise, pages automatically appear in the Modify Object task when the object selected is of the type that the page is designed to modify. Pages can also be accessed in a role-based way similar to tasks.

To create a property book page, you need to:

- ♦ [Section 4.2.1, “Create the Java Class,” on page 50](#)
- ♦ [Section 4.2.2, “Create the JSP,” on page 51](#)
- ♦ [Section 4.2.3, “Create the XML Registration File,” on page 52](#)

Once you have created a property book page, test it by running the Modify Object task (requires the Object Management plug-in) and select an object of the type your page can modify. Confirm that your page is available.

4.2.1 Create the Java Class

Create a Java class that handles the business logic of the page. The class should extend `com.novell.emframe.dev.PropertyBookPage`. Typically, you don’t need to create a custom Java class unless your page must deal with nonstandard syntaxes, such as octet strings.

NOTE: For pages that deal with eDirectory object attributes, use `com.novell.emframe.dev.DirPropertyBookPage` instead of creating your own class. This automatically uses the eDirectory Access Service (EDAS) for reading and writing attributes to eDirectory.

To create a custom Java class:

- 1 Create a class that extends `DirPropertyBookPage`.
- 2 Override the `show(TaskContext context, java.util.Properties resultStrings)` method to get data from the target.
- 3 Override `cache(PropertyPageContext context)` to cache data posted from the client.
- 4 If you are not using `NSObject` in the cache method, override the `commit(PropertyPageContext context)` method to properly update back links and write changes to the data source through an API other than the `NDSNamespace`.

IMPORTANT: To get a reference to an `NSObject` (for eDirectory) from any of the above-mentioned methods (`show`, `cache`, `commit`), call one of the following, then read or write to the resulting `NSObject`:

- ♦ `this.m_nsobj`
- ♦ `NSObject target = this.getDirPropertyBook().getNSObject()`

When the user clicks *Apply*, it calls the `NSObject`'s update method.

The iManager SDK provides an example of extending `DirPropertyBookPage` in `PropertyPageExampleTask1.java` and `PropertyPageExampleTask2.java`, which are available in the SDK at `SDK_HOME/samples/src/com/company/plugins/`.

4.2.2 Create the JSP

To create the UI for a page, you create a JSP. The process is nearly the same as creating the UI for a task, except that for a page, you always create only one JSP. In a property book, related attributes are usually grouped together on a page. Create a JSP that displays the values of the attributes. Use form controls for attributes that you want to allow users to modify. Disable the controls when users don't have rights to modify the attributes.

To create a property book page JSP:

- 1 Copy the `PageTemplate.jsp` file and save it as your new JSP (For example, `myPage.jsp`)

`PageTemplate.jsp` is in
`SDK_HOME\tomcat\webapps\nps\portal\modules\dev\dkins\default\devices\browser.`
- 2 Insert the necessary HTML controls after the comment `<!-- REMIND: CONTENT GOES HERE -->`.
- 3 In the JavaScript `isPageValid` method, insert client-side validation that tests the values in the controls and returns `true` if the values are valid.

For more information about creating JSPs, see [“Creating the UI” on page 26](#) and [“Tips for Creating JSPs” on page 32](#).

4.2.3 Create the XML Registration File

Property book pages must be defined in an XML registration file. Store the registration file in a \plugins directory inside your module. For example, Identity Manager stores its property book registration files in tomcat\webapps\nps\portal\modules\dirxml\plugins.

The registration file uses <page> or <private-page> elements to organize tasks into pages. The <page> and <private-page> elements have identical structures. Use the <private-page> element to prevent the page from appearing as part of the Modify Object task; otherwise, use <page>.

For example:

```
<page>
  <id>core.UserId</id>
  <version>1.0</version>
  <required-version>2.7.0</required-version>
  <class-name>com.novell.emframe.dev.DirPropertyBookPage</class-name>
  <merge-template>ModifyUser_Info</merge-template>
  <help-file>base/TestHelp.html</help-file>

  <default-display-name>Information</default-display-name>
  <display-name-key></display-name-key>
  <resource-properties-file>com.novell.CoreTasksResourceBundle</ resource-
properties-file>

  <chapter>
    <id>core.general</id>
    <display-name-key>GeneralChapterName</display-name-key>
    <resource-properties-file >com.novell.ResourceBundle</resource-properties-
file>
  </chapter>

  <object-type-name>User</object-type-name>
  <description></description>

  <rights-assignment>
    <attribute-name>Given Name</attribute-name>
    <privilege>Supervisor</privilege>
  </rights-assignment>
  <rights-assignment>
    <attribute-name>Surname</attribute-name>
    <privilege>Supervisor</privilege>
  </rights-assignment>
</page>
```

For complete descriptions of supported XML elements for the property book page registration file, see [“XML Schema for Installation and Registration Files” on page 111](#).

Using the Plug-In Studio

5

iManager includes the Plug-In Studio, which allows administrators to easily create iManager plug-ins from within iManager, usually without writing any code. You can use the Plug-In Studio to create the following types of eDirectory management plug-ins:

- ♦ Property book pages
- ♦ Tasks that create eDirectory objects
- ♦ Tasks that delete eDirectory objects
- ♦ Tasks that modify existing eDirectory objects

If you need to create a plug-in with capabilities that the Plug-In Studio cannot create, you can use the Plug-In Studio to quickly create a basic plug-in that you can customize for your needs. For more information, see [“Customizing Plug-Ins Created with the Plug-In Studio” on page 57](#).

This section contains the following topics:

- ♦ [Section 5.1, “The Plug-In Studio User Interface,” on page 53](#)
- ♦ [Section 5.2, “Creating a Custom Plug-In,” on page 54](#)
- ♦ [Section 5.3, “Control Parameters,” on page 56](#)
- ♦ [Section 5.4, “What the Plug-In Studio Creates,” on page 57](#)
- ♦ [Section 5.5, “Customizing Plug-Ins Created with the Plug-In Studio,” on page 57](#)

5.1 The Plug-In Studio User Interface

The Plug-In Studio contains the following UI elements:

- ♦ **Menu Bar.** From the menu bar, you can preview the plug-in. You can also install the task. This generates the files necessary and installs the current plug-in into eDirectory. It is then available for immediate use. Exit closes the studio without saving the current plug-in.

WARNING: Changes you make to an object using the preview of your plug-in are saved to the directory. If you need to test the plug-in beyond verifying the layout and testing the operation of the controls, you should use the Plug-In Studio in a non-production environment.

- ♦ **Plug-In Fields.** These are the fields you have added to your plug-in. If you are building a Create task, then the mandatory attributes are already added for you. To the right of each field are three buttons. The first button allows you to change the parameters of the field. The second button allows you to change the type of control the field will be displayed with. The final button allows you to remove the field. Mandatory attributes cannot be removed from Create tasks.
- ♦ **Plug-In Properties.** This section allows you to provide general information about the plug-in, such as its name.

- ♦ **Attributes.** The attributes section shows all of the fields that you can add to the selected fields section. Double-click a field to add it using the default control or click once and select a control from the controls section.
- ♦ **Controls.** For the attribute selected in the attributes section, the Plug-in Studio discovers all of the controls that can be used to manage the attribute based on its syntax and attribute type. In many cases you have multiple options. For example, the Allow Unlimited Credit attribute is a Boolean syntax. The Plug-in Studio provides three controls for managing attributes of Boolean syntax: check box, radio button, and select box.

5.2 Creating a Custom Plug-In

To create a plug-in using the Plug-In Studio:

- 1 Launch iManager and open the Configure view.
- 2 Select *Role Based Services > Plug-In Studio*.
- 3 In the Installed Custom Plug-ins page, click *New*.

To create a task or property book page based on an existing one, select the existing task or page, then click *Actions > Copy*.

- 4 In the Choose Object Type and Platform page, provide the required information, then click *Next*.

Available Classes: Select the type of object that you want to manage with your plug-in. The list shows the classes that are available in the tree to which you are currently connected.

Target Device: Select the device type that your plug-in will target. *Default* is appropriate for most situations where users access the plug-in with a Web browser. Select *Browser* to target Web browsers that require a simplified layout.

Plug-in Type: Select the type of plug-in you want to create. You can select a property book page or a task that either creates, modifies, or deletes objects of the selected class.

Add Aux Classes: Select to make auxiliary classes available in the Plug-In Studio. When selected, you can specify the auxiliary classes that you want to use. To select multiple classes, <Ctrl>-click.




- 5 In the Plug-in Studio page, complete the custom plug-in configuration.

The Plug-In Studio consists of four main areas:

Attributes: Provides a list of attributes the plug-in can modify. Double-click an attribute to add it as a field in your plug-in. Each attribute has If you are creating a Create task, the Plug-In Studio automatically adds fields for mandatory attributes.

Controls: Displays the controls that are available for the selected attribute.

Plug-in Fields: Displays the currently selected attribute controls for your plug-in. Click-and-drag a control to change its position within the plug-in. Plug-in Studio provides three buttons with each control so you can customize your plug-in as needed:

- ♦  *View or Modify Parameters:* Lets you configure the control's properties to specify if it is read-only, mandatory, etc.
- ♦  *Change Control:* Lets you change an attribute's control type from the attribute's list of available controls.
- ♦  *Delete:* Removes the attribute control from the plug-in.

Plug-in Properties: Specifies the properties of your custom plug-in, including the following:

- ♦ *Plug-in ID:* Specifies a unique name for the plug-in. By default, this is the plug-ins Display Name as well (in the Advanced Properties dialog box.) You can make the Display Name of a task or page identical to that of an existing task or page by modifying the XML registration file to use a display name from a resource bundle. For example: `<display-name-key>MyTaskDisplayName</display-name-key>`.
- ♦ *RBS Collection:* Specifies the RBS Collection into which the custom plug-in is installed.
- ♦ *Role:* Specifies the RBS role with which the custom plug-in is associated.
- ♦ *Advanced:* Lets you specify additional plug-in settings, including control labels and multi-object editing.
- ♦ *Chapter: (Conditional)* Specifies the chapter for a property book page.
- ♦ *Object Type: (Conditional)* Specifies the types of objects a property book page can modify.

6 Select *Preview* to test your plug-in.

When you have added fields and set the properties for your plug-in, you can test it by clicking Preview. iManager displays a new browser window showing your plug-in as it will appear to the user. Click Cancel to close the preview window.

WARNING: Changes you make to an object using the preview of your plug-in are saved to the directory. If you need to test the plug-in beyond verifying the layout and testing the operation of the controls, you should use the Plug-In Studio in a non-production environment.

7 Select *Install* to install the plug-in.

If you have specified all required properties and parameters, your custom plug-in is installed into the directory and saved in `/tomcat/webapps/nps/packages/custom.npm`.

5.2.1 Adding Drop Down List Values Dynamically

Custom plug-ins, including those developed with Plug-in Studio, support dynamic creation of drop down list options. The following example shows one way to accomplish this.

In your task, include Java code such as the following:

```
properties.put("Sch_ClassList.count" , size);

for (int i=0; i<size; i++)
{
    properties.put("Sch_ClassList.name." + i, str[0]);
}
```

Then, in plug-in's JSP file, display the dynamic dropdown list as follows:

```
<td>

<select name="AvailableClasses" size=10 style="WIDTH: 340px" >

    <% c.set("i", "0");
```

```

while (c.lt(c.var("i"), c.var("Sch_ClassList.count")))
{
    %>

    <option value="<%= c.var("Sch_ClassList.name."+c.var("i")) %>

    <% c.inc("i");

    } %>

</select>

</td>

```

Dynamic drop down lists are also supported by MVSelectBox. In HTML, the MVSelectBox control is rendered as a drop-down list using the select tag. Options are specified using the option tag. To dynamically add options, modify the JSP created by the Plug-In Studio and use JSP scriptlets or tag library actions to create options at runtime. The following example shows how an MVSelectBox control designed to modify the employeeType attribute can be modified to add an additional option if the user has a certain given name:

```

<SELECT name="_employeeType" style="width:<iman:string key="UI.textboxPixel"/>"
onChange="onMVSelectChange('employeeType', this)">

    <OPTION value="Individual Contributor" <x:if select="count($edasXml/edas/
employeeType/value[text()='Individual Contributor'])">selected</x:if>>Individual
Contributor

    <OPTION value="Manager" <x:if select="count($edasXml/edas/employeeType/
value[text()='Manager'])">selected</x:if>>Manager

    <x:if select="count($edasXml/edas/Given_Name/value[text()='Jack'])">    <OPTION
value="CEO" <x:if select="count($edasXml/edas/employeeType/
value[text()='CEO'])">selected</x:if>>CEO

    </x:if>

</SELECT>

```

The Plug-In Studio creates JSPs in webapps/nps/portal/modules/custom/skins/default/devices/default.

5.3 Control Parameters

Most controls allow you to specify parameters to customize the control. Currently, most controls support the read-only parameter, which is very useful when you want to build a task for displaying information, without allowing the user to modify it. Choosing to make a field read-only both disables the user interface and gives the users read access to the attributes (if you are using Role-Based Services).

The values parameter is useful for limiting the possible values that can be entered. If your company has three departments (research, marketing, and IS) you can build a plug-in for creating users that allows user creation only in those containers.

5.4 What the Plug-In Studio Creates

The Plug-In Studio creates:

- ♦ A JSP file for the UI

The JSP file contains both HTML and Java tag library calls that are run on the server. This file is created in `SDK_HOME/tomcat/webapps/nps/portal/modules/custom/skins/default/devices/default`. A copy of the file is also added to the `Custom.npm` file.

This file can be modified to make custom enhancements that are not possible through the Plug-In Studio. For example, you could add an image that displays your company's logo. However, if you later use the Modify Plug-In feature of the Plug-In Studio, you lose any manual modification you have made to the JSP file.

- ♦ A registration file

For property pages, an XML file is created and written to the `SDK_HOME/tomcat/webapps/nps/portal/modules/custom/plugin-ins/plugin-ins.xml` file. This contains information such as the page name and the object class it is used to manage. For tasks, the registration file is installed into eDirectory, and the information is stored in an `RBSTask` object. For both pages and tasks, a copy of the registration file is added to `Custom.npm`.

- ♦ An updated `Custom.npm` file

The `Custom.npm` file is ZIP-compressed file that contains all of the custom plug-ins created by the Plug-In Studio. This file is used for two purposes. First, it is used to automatically synchronize plug-ins with other iManager servers in the same tree. Second, it moves the custom plug-ins to another tree. For example, you can take this one file from a test environment to a production environment and simply deploy the NPM and then run the Configure iManager task. The Deploy and Configure tasks are found in the iManager Configure view.

5.5 Customizing Plug-Ins Created with the Plug-In Studio

Sometimes you need a plug-in with more functionality than the Plug-In Studio can provide. In this situation you can use the Plug-In Studio to build most of the user interface needed, then add additional features by writing custom Java code.

The following code shows how to customize a task:

```
public class MyCustomTask extends eDirAccessService
{
    protected boolean preRead(TaskContext context, Properties resultStrings)
    {
        // do work here
        return true;
    }

    protected boolean postRead(TaskContext context, Properties resultStrings)
    {
        // do work here
        return true;
    }
}
```

```

protected boolean preWrite(TaskContext context, Properties resultStrings)
{
    // do work here
    return true;
}

protected boolean postWrite(TaskContext context, Properties resultStrings)
{
    // do work here
    return true;
}
}

```

To customize a page, use the following example:

```

public MyCustomPage extends DirPropertyBookPage
{
    public void cache(PropertyPageContext context) throws PageException
    {
        // work here
        save(context, context.getResultStrings());
        // or work here
    }

    public void show(PropertyPageContext context) throws PageException
    {
        show(context, context.getResultStrings());
    }
}

```

After your Java code is compiled and placed in the classpath (in the classes directory or in the lib directory in a JAR file), you must update the plug-in to point to your Java class instead of the default one that the Plug-in Studio assigns. For a task, this is done by modifying the `rbsTask` object in `eDirectory`. Change the entry point attribute to the fully qualified name of your class. For a property page, edit the plug-in XML file in the `custom/plug-ins` directory. Change the entry-point element to the fully qualified class name.

5.5.1 Dynamically Updating Drop-down Lists

It is possible to dynamically fill a drop-down list for plug-ins developed with Plug-in Studio, and other custom plug-ins. However, it is not possible to do this for base content plug-ins. The following example shows one way this can be done:

In your task, include Java code such as the following:

```

properties.put("Sch_ClassList.count" , size);

for (int i=0; i<size; i++)
{
    properties.put("Sch_ClassList.name." + i, str[0]);
}

```

Then, in the JSP file you could display the dynamic dropdown list as follows:

```

<td>

<select name="AvailableClasses" size=10 style="WIDTH: 340px" >

  <% c.set("i", "0");

  while (c.lt(c.var("i"), c.var("Sch_ClassList.count")))

  { %>

    <option value="<%= c.var("Sch_ClassList.name."+c.var("i")) %>

    <% c.inc("i");

  } %>

</select>

</td>

```

The Plug-In Studio creates JSPs in webapps/nps/portal/modules/custom/skins/default/devices/default.

Using the iManager Widgets

6

iManager provides many reusable controls and other UI elements that you can use to create the UI for your plug-in. The UI elements, called widgets, include buttons, icons, and controls for selecting objects and modifying attribute values. The code defining these items is found in `*_inc.jsp` files, which are stored in subdirectories of the `SDK_HOME/tomcat/webapps/nps/portal/modules/dev/skins/default/devices` directory, by device.

In addition, the `SDK_HOME/tomcat/webapps/nps/portal/modules/dev/javascrpts` directory contains libraries of useful JavaScript functions that you can utilize in JSPs.

The widgets and JavaScript libraries enable you to quickly create a UI that looks and works like the iManager base content.

We recommend that you study the iManager samples for ideas about how to use widgets and JavaScript functions.

This section covers the following topics:

- ♦ [Section 6.1, “The Object Selector Widget,” on page 61](#)
- ♦ [Section 6.2, “The Advanced Selection Widget,” on page 69](#)
- ♦ [Section 6.3, “The MVStringEditor Widget,” on page 75](#)
- ♦ [Section 6.4, “The Date/Time Widget,” on page 79](#)

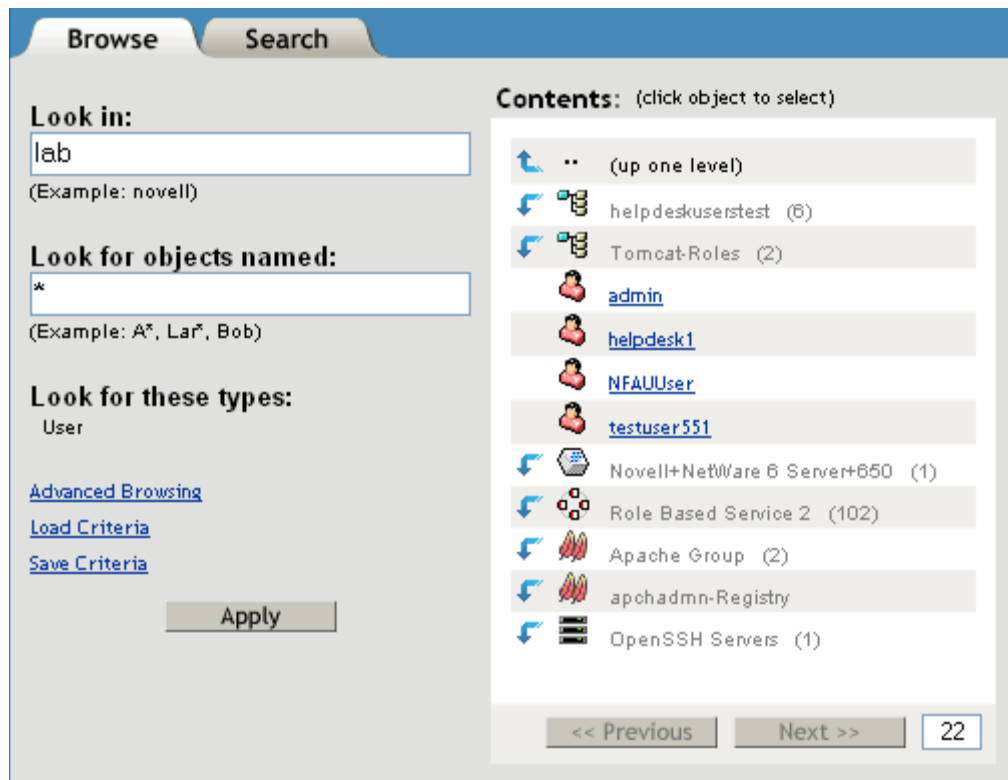
6.1 The Object Selector Widget

The Object Selector widget lets a user search or browse for an object in eDirectory, select the object, and return the name of the object to the form field of the task user interface. It supports single or multiple selections. Selected objects are returned as DN strings which are, by default, placed in the control specified using the `OS.Control` variable. However, the task developer can have the result strings returned to a JavaScript callback specified with the `OS.CallBack` variable, where they can be manipulated as desired.

Implement the Object Selector widget using include files and several parameter variables.

The following figure shows the iManager Object Selector in simple browse mode.

Figure 6-1 iManager Object Selector in simple browse mode.



This section describes the following topics:

- ◆ [Section 6.1.1, “Include Files,” on page 62](#)
- ◆ [Section 6.1.2, “Parameter Variables,” on page 63](#)
- ◆ [Section 6.1.3, “Advanced Selection XML Syntax,” on page 65](#)
- ◆ [Section 6.1.4, “Dynamically Enabling and Disabling the Object Selector,” on page 65](#)
- ◆ [Section 6.1.5, “Pre-Processing and Post-Processing Routines \(preOS and postOS\),” on page 65](#)
- ◆ [Section 6.1.6, “Making the Root Selectable,” on page 66](#)
- ◆ [Section 6.1.7, “Making Public and This Selectable,” on page 66](#)
- ◆ [Section 6.1.8, “Filtering on All Container Types,” on page 66](#)
- ◆ [Section 6.1.9, “Filtering on Containers that Are Partitions,” on page 66](#)
- ◆ [Section 6.1.10, “Object Selector Support in the iManager Tag Library,” on page 66](#)
- ◆ [Section 6.1.11, “A JSP Tag Library Example: Delete User,” on page 68](#)
- ◆ [Section 6.1.12, “Troubleshooting,” on page 68](#)

6.1.1 Include Files

The following table describes the include files used to implement the Object Selector:

Table 6-1 *Object selector include files*

File	Description
OSScripts_inc.jsp	Contains the JavaScript used for implementing the OS widget. With a browser, these scripts are stored in cached .js script files to minimize data transfer over the wire.
OSFooter_inc.jsp	Contains hidden variables used for implementing the selector functionality. Must be included after the last </form> tag on the JSP page.
OS_inc.jsp	Inserts an image that can be clicked to display the OS widget.
OSBtn_inc.jsp	Inserts an HTML button that can be clicked to display the OS widget.
OSLink_inc.jsp	Inserts an HTML hypertext link that can be clicked to display the OS widget.

To use these files, include them in your task JSPs using the jsp:include action.

6.1.2 Parameter Variables

Use either `OS_inc.jsp`, `OSBtn_inc.jsp`, or `OSLink_inc.jsp` to insert a control that opens the Object Selector when clicked. Each of these controls use the following parameters to determine how the Object Selector operates:

Table 6-2 *Object selector parameter variables*

Parameter	Description
OS.Control	(Required) The name of the field where the selected objects are returned.
OS.Callback	(Required) The name of a JavaScript routine where the returned values and control name are returned for processing. The callback routine must have the following parameter list format: <code><callbackName>(<controlName>,<resultsArray>)</code> . The names of the selected objects are returned in <code>resultsArray</code> .
OS.AdvancedSelection	Whether the Object Selector appears with the advanced search visible. False displays simple type-only filters. The default is false.
OS.AdvSelCriteria	A string representing a comma-separated list of XML representations of advanced selection filter criteria for ds-classes. (See Section 6.1.3, "Advanced Selection XML Syntax," on page 65 for the required syntax of the XML criteria). If this parameter is present, the OS.AdvancedSelection parameter is assumed to be true.
OS.AdvSelCriteriaCallback	The name of a JavaScript routine that returns a string representing a comma-separated list of XML representations of advanced selection filter criteria for ds-classes. (See Section 6.1.3, "Advanced Selection XML Syntax," on page 65 for the required syntax of the XML criteria). This routine must have the following parameter list format: <code><callbackName>(<controlName>)</code> . If this parameter is present, the OS.AdvancedSelection parameter is assumed to be true.
OS.AltText	The hint or accessibility text displayed when the user moves the mouse over the Object Selector icon. If not supplied, "Object Selector" is displayed. Used with OS.inc.

Parameter	Description
OS.IconName	The name of an icon to use for the selector button instead of the standard icon. Used with OS.inc.
OS.InitialContext	The name of a container where the search or browse begins The default is the container from the last search, or [root].
OS.IsOSAllowed	The name of a JavaScript routine that returns a Boolean true if the object selector is to proceed or false if the object selector operation is to be prevented. This routine is called before the object selector widget is displayed and can be used to prevent its launch unless conditions (which are checked in the routine) are met. For more information, see “Dynamically Enabling and Disabling the Object Selector” on page 65.
OS.Mode	The Object Selector mode. Set to Search to allow the user to search and find objects in the tree with a search query. Set to Browse to allow the user to walk the tree and manually select objects. The default is Browse.
OS.MultiSelect	Whether the user can select more than one object. The default is false (single select).
OS.NameFilter	A string representing a search filter to be used to limit the number of items returned to those objects whose name matches the filter requirements. The * wildcard can be used in the name filter. The default is *.
OS.ResultsPerPage	The number of results to show per page. The default depends on the selection mode and device.
OS.SearchSubContainers	Whether to allow the search operation to search for objects starting in the specified container and all of its subcontainers. False limits the search to the specified container only. The default is true.
OS.SearchOnStartup	Whether to force an initial search operation to begin as soon as the Object Selector appears, if it appears in search mode. OS.Mode="search" must also be set. This automatic search behavior occurs only when the object selector starts up, never while switching between the search and browse tabs. False prevents an automatic search, requiring the user to click the Search button to start the search. The default is false.
OS.ShowSubClasses	Whether the search returns objects of types derived from types that match the type filter in addition to objects that match the base types. False causes the search to not return objects derived from matching types. The default is true.
OS.Text	The text displayed on the HTML button or HTML link used to bring up the object selector. Used with OSBtn.inc and OSLink.inc.
OS.TypeFilter	A comma-separated list of object types to be returned. Specifying * returns all object types.
OS.TypeFilterCallback	The name of a JavaScript routine that returns a string containing a comma-separated list of object types to be returned. Specifying * returns all object types. The callback routine must have the following parameter list format: <code><callbackName>(<controlName>)</code>
OS.Windowed	Whether the simple Object Selector appears in a separate window. If false, the Object Selector appears in the same window as the task, which requires that the task state be saved before the Object Selector runs and be restored afterwards. This is problematic for complex tasks. Setting this to true might solve these problems. The default is false.

6.1.3 Advanced Selection XML Syntax

The XML syntax definition of an Advanced Selection criterion is:

```
<selection-criterion>
  <types>
    <id> ds-type1 </id>
    <id> ds-type2 </id>

  </types>
  <aux-types>
    <id> ds-auxtype1 </id>
    <id> ds-auxtype2 </id>

  </aux-types>
  <group>
    <row>
      <attribute> name </attribute>
      <operation> op </operation>
      <value> string </value>
      <row-op> and/or </row-op>
    </row>
    ...
    <group-op> and/or </group-op>
  </group>
  ...
</selection-criterion>
```

6.1.4 Dynamically Enabling and Disabling the Object Selector

The user activates the Object Selector by clicking on an icon, link, or button, depending on whether you have used the OS.inc, OSLink.inc or OSBtn.inc files in your task. However, at times you might want to disable the Object Selector until certain steps have been performed in the task. This is accomplished using the `IsOSAllowed()` function (see [OS.IsOSAllowed](#)). You provide a JavaScript function that determines, at runtime, when the Object Selector is allowed to run. Each time the user clicks the Object Selector button, this function is called. You provide the logic to determine whether the Object Selector operation is allowed at that time. If so, you return Boolean true, otherwise false.

6.1.5 Pre-Processing and Post-Processing Routines (preOS and postOS)

It is often desirable for a task to perform some processing before the Object Selector executes or after it returns the selected objects to the task. You can do this by implementing JavaScript routines, `preOS(controlName)` and `postOS(controlName)` for your tasks.

If the Object Selector is enabled (see [OS.IsOSAllowed](#)), an attempt is made to call a JavaScript routine called `preOS(controlName)`, if it exists. This enables you to do any processing you need before the Object Selector runs.

When the Object Selector finishes, the selected results are returned to either the specified control or callback routine and then the `postOS(controlName)` function is called, if it exists. At this point, you can do anything you want to clean up after the Object Selector in your task.

controlName is a string representing the HTML control that is associated with the Object Selector using the **OS.Control** parameter.

6.1.6 Making the Root Selectable

It is often necessary to allow the user to select the root of the tree as a place to perform operations (like create, delete, move, and modify). However, the root of the tree is not an object in DS, so you cannot search or browse to it directly. If you want your users to be able to select the root of the tree as a location to perform operations, simply specify [root] as the value of the **OS.TypeFilter** parameter.

6.1.7 Making Public and This Selectable

eDirectory has two other pseudo-objects that can be made selectable: This and Public. To make these selectable, specify [this] or [public] as the value of the **OS.TypeFilter** parameter.

6.1.8 Filtering on All Container Types

Sometimes you need to allow the user to select any container-type object in the directory, but it is not always possible to know what all the container types in the directory are at the time you create the task. The system administrator can extend the directory schema and create new container objects.

To enable users to select any container object in the directory, specify [containers] as the value of the **OS.TypeFilter** parameter. The Object Selector queries the directory to get a list of all containers in the tree and dynamically populates the type filter with the available container types.

6.1.9 Filtering on Containers that Are Partitions

eDirectory represents partitions as an attribute of a container class and does not currently provide the ability to find only containers that are partitions. However, to enable users to select only partition container objects in the directory, specify [partitions] as the value of the **OS.TypeFilter** parameter. All container objects are displayed, but only those that are partitions are selectable. If you want all containers that are not partitions, specify [non-partitions].

6.1.10 Object Selector Support in the iManager Tag Library

iManager 2.7 provides a custom tag library to developers who prefer using tag libraries in their JSP templates. Use these tags along with, or in place of, the auto-conversion tools the Conduit object creates during the port from .http files to .jsp files.

IMPORTANT: An iManager Tag Library reference is available in iManager. To view it, open the iManager Developer view, then select *Developer Reference > iMan Tag Library Reference*.

The following tags support the Object Selector:

- ♦ `<iman:eMFrameScripts/>`
- ♦ `<iman:osScripts/>`

- ♦ `<iman:os attrib1 . . . attribn/>`
- ♦ `<iman:osFooter/>`

The `<iman:eMFrameScripts/>` and `<iman:osScripts/>` tags define general data and routines used by most iManager tasks and those required specifically for the Object Selector. They must be included in the `<head>` section of the HTML task document.

The `<iman:os attrib1 . . . attribn/>` tag displays an Object Selector button in the task. You can provide values for all of the defined Object Selector values through the attributes listed in the following table:

Table 6-3 *Object selector attributes*

Property Object Name	JSP Tag Attribute Name
OS.Control	control
OS.Callback	callBack
OS.AltText	altText
OS.AdvancedSelection	advancedSelection
OS.AdvSelCriteria	advSelCriteria
OS.AdvSelCriteriaCallback	advSelCriteriaCallback
OS.CustomProcessing	customProcessing
OS.IconName	iconName
OS.InitialContext	initialContext
OS.IsOSAllowed	isOSAllowed
OS.Mode	mode
OS.MultiSelect	multiSelect
OS.NameFilter	nameFilter
OS.ResultsPerPage	resultsPerPage
OS.SearchSubContainers	searchSubContainers
OS.SearchOnStartup	searchOnStartup
OS.ShowSubClasses	showSubClasses
OS.Text	text
OS.TypeFilter	typeFilter
OS.TypeFilterCallback	typeFilterCallback
OS.Windowed	windowed

The `<iman:osFooter/>` tag creates the form through which all data is posted between the Object Selector and the task that is using it. It must be placed at the bottom of the HTML before the `</body>` tag.

6.1.11 A JSP Tag Library Example: Delete User

```
<%@ page pageEncoding="utf 8" contentType="text/html; charset=utf 8" %>
<%@ taglib uri="/WEB-INF/iman.tld" prefix="iman" %>
<HEAD><TITLE>MyTask</TITLE>
<iman:eMFrameScripts/>
<iman:osScripts/>
<script>
    function capitalize(controlName, results)
    {
        var strResults = "";
        for (i=0; i<results.length(); i++)
        {
            if (i>0) { strResults += "+" }
            strResults += results[i].toUpperCase();
        }
        document.CreateUserForm.elements[controlName].value=strResults;
    }
    function readyToSearch() { return true }
</script>
</HEAD>
<BODY TEXT="#000000" bgcolor="#FFFFFF" onLoad="returnFromOS()">
<FORM name="DeleteUserForm" method=post action="webacc">
    <input type=hidden name="taskId" value="{VAR taskId}">
    <input type=hidden name="merge" value="dev,GenConf">
    <input type=hidden name="error" value="dev.GenErr">
    <input type=hidden name="nextState" value="doDeleteUser">
    <BR><FONT size=+1 color=#000080><B>Delete User</B></FONT><BR><BR>
    <FONT face="Helvetica,Arial" size= 1><NOBR>User name:</NOBR></FONT><BR>
    <input type=text name="DeleteUserName" value="" size=35>
    <iman:os control="DeleteUserName"
        callBack="capitalize"
        nameFilter="a*"
        typeFilter="User"
        initialContext="Novell"
        mode="browse"
        multiSelect="true"
        altText="User to delete"
        iconName="dev/add.gif"
        searchSubContainers="false"
        searchOnStartup="true"
        isOSAllowed="readyToSearch" />
    <br><br><input type=image name="Delete" alt="Delete" src="<path>/delete.gif"
border=0
</form>
<iman:osFooter/>
</BODY>
```

6.1.12 Troubleshooting

- I select an object, but it is not returned to the edit box. What is wrong?
 - ♦ Make sure you have included the **OSFooter_inc.jsp** as the last line of the template before the </body> tag. You can put it anywhere in the template where a form can be defined, but it is easy to simply put it before the </body> tag. Never put it inside of <form></form> tags.

- ♦ Make sure you entered the name of the **OS.Control** correctly. The value of this attribute must match the corresponding HTML control name exactly. This check is case sensitive.
- ♦ If you are using a callback routine, debug to see if the callback is being called and if the two required parameters, `controlName` and `resultArray`, are being returned correctly.
- ♦ I get a developer alert when running my task. Why?
You are probably using the obsolete `OSHeader_inc.jsp` file in your template. Remove it and use the `OSFooter_inc.jsp` file as the last line in your template before the `</body>` tag.
- ♦ My password and file `<input>` fields are not being saved in single-window mode. Why?
These fields are intentionally not saved for security reasons.
- ♦ Returning a selected object on Simple pages causes unexpected errors or processing. Why?
The Simple implementation of the Object Selector assumes it can restore the state of your task by reissuing the URL that brought up the page in the first place. This is almost always a safe assumption. However, if the URL causes some preprocessing to occur that cannot safely be repeated when the Object Selector is returning to your task, there is a problem.
There is nothing the Object Selector can do to prevent this problem. You must know that this problem can occur and prevent it. When a result is being returned from Object Selector, the following parameter is added to the original URL: `OS.ReturningToAnchor="true"`. Your code must detect this condition and prevent preprocessing that might cause an error to occur.

6.2 The Advanced Selection Widget

The Advanced Selection (AS) widget lets a user specify a set of advanced selection criteria for a specified object type (class.) The result (output) is a string representing the XML description of the criteria the user has entered. This criteria can be used by a namespace's `getChildren` process call to specify which objects in the directory to return to the developer's task.

The input to the Advanced Selection widget is a string representing with the name of a directory type (class). Optionally, a string representing a previous XML selection criteria can be supplied and the widget will be initialized accordingly. (Useful when editing selection criteria).

The widget displays attributes for the specified directory type (class). When the user selects an attribute, operators appropriate for the selected attribute are presented in a pull-down list. Further, a list of possible criteria values might be displayed. The user selects an attribute, chooses a comparison operation and supplies their desired criterion (or possibly selects the criterion from a list of allowed values). These criteria can be grouped together into a logic group and multiple logic groups can be defined.

You can supply a Java method to populate the widget's attribute list, operators/attribute, and possible values/attribute. If you don't supply a method, `iManager` assumes that `eDirectory` is being accessed and it uses a default method (`NDSTypeInfoCallback.java`) to retrieve this information.

This design implies that this widget is completely namespace independent. It can be used with `eDirectory`, any LDAP namespace, or any future namespace.

This section includes the following topics:

- ♦ [Section 6.2.1, "How to Call the AS Widget," on page 70](#)
- ♦ [Section 6.2.2, "Include Files," on page 70](#)
- ♦ [Section 6.2.3, "Parameter Variables," on page 70](#)

- ♦ [Section 6.2.4, “A JSP Example: Delete Users,” on page 71](#)
- ♦ [Section 6.2.5, “Setting the Options at Runtime Using JavaScript,” on page 73](#)
- ♦ [Section 6.2.6, “Implementing AdvSelTypeInfoCallback,” on page 74](#)
- ♦ [Section 6.2.7, “The Resulting XML Selection Criteria,” on page 75](#)

6.2.1 How to Call the AS Widget

The Advanced Selection widget is statically implemented using include files and several parameter variables. Static implementation means that the parameters cannot be changed at runtime because they are merged into the HTML. However, there is still significant flexibility since you can statically specify callback routines that execute at runtime to retrieve required input parameters and process the XML output.

6.2.2 Include Files

The following table describes the include files used to implement the AS widget:

Table 6-4 *AS widget include files*

File	Description
ASScripts_inc.jsp	Contains the JavaScript used for implementing the AS widget. With a browser, these scripts are stored in cached .js script files to minimize data transfer over the wire.
AS_inc.jsp	Inserts an image that can be clicked to display the AS widget.
ASBtn_inc.jsp	Inserts an HTML button that can be clicked to display the AS widget.
ASLink_inc.jsp	Inserts an HTML hypertext link that can be clicked to display the AS widget.

To use these files, include them in your task JSPs using the `jsp:include` action.

6.2.3 Parameter Variables

Use either `AS.jsp`, `ASBtn.jsp`, or `ASLink.jsp` to insert a control that displays the Advanced Selection feature when clicked. Each of these controls use the following parameters to determine how the feature operates:

Table 6-5 *Advanced Selection control parameters*

Parameter	Description
AS.Types	(Required) The name of the directory type (class) that advanced selection criteria will be entered for. Specify either <code>AS.Types</code> or <code>AS.TypesCallBack</code> .
AS.TypesCallBack	(Required) The name of a JavaScript routine that returns the name of the directory type (class) that advanced selection will be entered for. This routines has no parameters. Specify either <code>AS.Types</code> or <code>AS.TypesCallBack</code> .

Parameter	Description
AS.Control	(Required) The name of the HTML component where the selection criteria is returned. Specify either AS.Control or AS.ControlCallBack.
AS.ControlCallBack	(Required) The name of a JavaScript routine that processes the returned XML criteria. The callback routine must have the following parameter list format: <callbackName>(<controlName>,<typeName>,<XMLCriteria>). The <controlName> can be used to know where to direct the process XML criteria. The typeName identifies which directory type (class) the criteria is for. The XML Criteria is a string representing the advanced selection criteria. Specify either AS.Control or AS.ControlCallBack.
AS.InitialCriteria	An XML criteria string representing an set of initial selection criteria that is used to start the widget in a desired state. This is useful if you want to edit an existing criteria or help the user by pre-entering known selection information. Specify either AS.InitialCriteria or AS.InitialCriteriaCallBack.
AS.InitialCriteriaCallBack	The name of a JavaScript routine that locates and returns an XML criteria string representing an set of initial selection criteria that is used to start the widget in a desired state. This routine must have the following parameter list format: <callbackName>(<typeName>). The <typeName> is used to identify which XML criteria is requested.
AS.TypeInfoCallBack	A string representing the name of a Java implementation of the AdvSelTypeInfoCallBack abstract class that can used to retrieve the attribute/comparison operations/allowed values information in the form of a AdvSelTypeInfo class. If this parameter is undefined or the empty string, it is assumed that the eDirectory is being used and that all attributes of type string, integer or boolean are to be displayed to the user in the widget. The NDSTypeInfoCallBack.getInfo() method is called to retrieve this default information.
AS.IsASAllowed	The name of a JavaScript routine that returns a Boolean true if the advanced selection widget is to proceed, or false if its execution is to be prevented (disabled). This routine is called before the advanced selection widget is displayed and can be used to prevent its launch until desired conditions (which are checked in the routine) have been met.
AS.IconName	The name of an icon to use for the Advanced Selection widget button instead of the standard one. Used with AS.inc.
AS.AltText	The hint or accessibility text to be displayed when the user moves the mouse over the advanced selection icon/link/button. If the text is not defined, "Advanced Selection" is displayed. Used with AS.inc, ASBtn.inc and ASLink.inc.
AS.Text	The text to be displayed on the HTML button or HTML link, which is used to bring up the Advanced Selection widget. Used with ASBtn.inc and ASLink.inc.

6.2.4 A JSP Example: Delete Users

```

<%@ page pageEncoding="utf-8" contentType="text/html; charset=utf-8" %>
<%      <%@ taglib uri="/WEB-INF/iman.tld" prefix="iman" %>

<HEAD><TITLE>Delete Users</TITLE>
<iman:eMFrameScripts/>
<iman:asScripts/>

```

```

<script>
  var deleteCriteria=null;
  function saveXML(controlName, type, xmlCriteria)
  {
    deleteCriteria[type] = xmlCriteria;
    document.DeleteUserForm.DeleteUsersCriteria.value = xmlCriteria;
  }
  function getXML(type)
  {
    var xml = deleteCriteria[type];
    if (xml!=null) return xml;
    else return "";
  }
  function readToSearch()
  {
    if (something) return true;
    else return false;
  }
</script>
</HEAD>

<BODY TEXT="#000000" bgcolor="#FFFFFF">
<FORM name="DeleteUserForm" method=post action="webacc">
  <input type=hidden name="taskId" value="{VAR taskId}">
  <input type=hidden name="merge" value="dev,GenConf">
  <input type=hidden name="error" value="dev.GenErr">
  <input type=hidden name="User.context" value="{VAR User.context}">
  <input type=hidden name="nextState" value="doDeleteUsers">

  <BR><FONT size=+1 color=#000080><B>Delete Users</B></FONT><BR><BR>
  <NOBR>Users to be deleted:</NOBR><BR>
  <input type=text name="DeleteUsersCriteria" value="" size=35>

  <iman:as control="DeleteUsersCriteria"
    controlCallBack="saveXML"
    types="User"
    initialCriteriaCallBack="getXML('User') "
    typeInfoCallBack="com.novell.emframe.zen.ZenDeleteUsersTypeInfo"
    altText="Describe users to be deleted"
    iconName="dev/delete.gif"
    isASAllowed="readyToSearch"/>

  <BR><BR>
  <input type=image name="Delete" alt="Delete" src="<path>/delete.gif" border=0
</FORM>
</BODY>

```

The preceding example assumes:

- ♦ saveXML is a JavaScript routine that processes the XML criteria.
- ♦ getXML is a JavaScript routine that retrieves initial XML criteria.
- ♦ ZenDeleteUsersTypeInfo is a Java class that implements the AdvSelTypeInfoCallback abstract class and returns an AdvSelTypeInfo object containing the attribute/comparison operations/selection values to be displayed.

6.2.5 Setting the Options at Runtime Using JavaScript

You can call the `doDynamicAS` function from JavaScript to set the Advanced Selection widget parameters at runtime. However, we recommend that you use the static include files when possible to standardize the user experience. The following example shows how to use the `doDynamicAS` function:

```
function doDynamicAS(userContext, AS)
{
    //make sure we have an entry for the required parameters
    if ( (AS.types!=null || AS.typesCallBack) && //do we have a valid type or type
        callback
        (AS.control!=null || AS.controlCallBack)) / //do we have a control or
        control callback
    {
        //make sure we at least have default values for all parameters
        if (AS.types==null) AS.types="";
        if (AS.typesCallBack==null) AS.typesCallBack="";
        if (AS.control==null) AS.control="";
        if (AS.controlCallBack==null) AS.controlCallBack="";
        if (AS.initialCriteria==null) AS.initialCriteria="";
        if (AS.initialCriteriaCallBack==null) AS.initialCriteriaCallBack="";
        if (AS.isASAllowed==null) AS.isASAllowed="";
        if (AS.typeInfoCallBack==null) AS.typeInfoCallBack="";
        if (AS.validateSyntax==null) AS.validateSyntax="";

        //is window being sized dynamically?
        if (AS.windowWidth!=null) window.asWidth=AS.windowWidth;
        if (AS.windowHeight!=null) window.asHeight=AS.windowHeight;

        //are the widths of the input well being sized dynamically?
        if (AS.attributeWidth!=null) window.asAPercent=AS.attributeWidth;
        if (AS.operationWidth!=null) window.asOPercent=AS.operationWidth;
        if (AS.valueWidth!=null) window.asVPercent=AS.valueWidth;

        //call the private doAS routine
        doAS(userContext,
            AS.types,
            AS.typesCallBack,
            AS.control,
            AS.controlCallBack,
            AS.initialCriteria,
            AS.initialCriteriaCallBack,
            AS.isASAllowed,
            AS.typeInfoCallBack,
            AS.validateSyntax);
    }
}
```

NOTE: `AS.validateSyntax` is reserved for future use.

As you can see from this code, the following parameters can be changed dynamically at runtime using this function. You must create a JavaScript object with these values. Note that the first character following the “AS.” is lowercase to distinguish it from the corresponding static {SET} variables.

- ◆ `AS.types`

- ♦ AS.typesCallBack
- ♦ AS.control
- ♦ AS.controlCallBack
- ♦ AS.initialCriteria
- ♦ AS.initialCriteriaCallBack
- ♦ AS.isASAllowed
- ♦ AS.typeInfoCallBack

You can change the window size dynamically by setting the following variables:

- ♦ AS.windowWidth
- ♦ AS.windowHeight

You can change the relative size percentages of the three primary entry wells as follows:

- ♦ AS.attributeWidth
- ♦ AS.operationWidth
- ♦ AS.valueWidth

Implementing the user interface to bring up the widget is up to the task developer. We recommend that you preserve the look and feel provided with the static include files to standardize the user's GUI experience.

6.2.6 Implementing AdvSelTypeInfoCallback

The Advanced Selection widget is independent of namespace (eDirectory, ActiveDirectory, iPlanet, LDAP, etc). It simply displays a list of strings representing attributes for a given type, a list of strings representing comparison operators, and possibly a list of strings representing allowed values for a particular attribute. It understands no syntax or semantics pertaining to these values.

Because the Advanced Selection widget knows nothing about any directory (their types, operations, values, etc), this information must come from somewhere. You provide the information through the AdvSelTypeInfoCallback Java class. If you are either not using eDirectory or do not want all the eDirectory attributes for a particular type displayed, you must create a new class that extends the AdvSelTypeInfoCallback abstract class and implement its getInfo() method. When called, this method must return an AdvSelTypeInfo object containing this information. The AdvSelTypeInfo class has the following two constructors. The first takes an array of strings representing the comparison operations that are assumed if none are specified for a particular type (class). The second is used if comparison operations are specifically defined for each type:

- ♦ public AdvSelTypeInfo(String[] defaultOps)
- ♦ public AdvSelTypeInfo()

After you have created an `AdvSelTypeInfo` object, you must call one of the following two `addAttribute()` methods for each type that you want to appear in the widget's [attribute] list. The first is used to create an attribute with allowed values; the second is used if you want the user to be able to type anything into the [value] entry well. If the `ops` parameter is null, the default operations are assumed, if defined:

- ♦ `public void addAttribute(String name, String[] ops, String[] values)`
- ♦ `public void addAttribute(String name, String[] ops)`

After writing the `AdvSelTypeInfoCallback` class, compile it and placed it in a JAR that appears on the Java classpath. Then by setting the `AS.TypeInfoCallback {SET}` parameter (or the `AS.typeInfoCallback` dynamic parameter), it will be used to supply the necessary information at runtime.

TIP: See the default eDirectory (NDS[®]) implementation in `NDSTypeInfoCallback.java` for an example of how to write an `AdvSetTypeInfoCallback` class. This class is assumed if no callback class is specified or if the specified class cannot be found.

6.2.7 The Resulting XML Selection Criteria

The Advanced Selection widget returns the selection criteria that the user enters in the following XML format:

```
<selection-criteria>
  <types> ds-type </types>
  <group>
    <row>
      <attribute> name </attribute>
      <operation> op </operation>
      <value> string </value>
      <row-op> and/or </row-op>
    </row>
    . . .
    <group-op> and/or </group-op>
  </group>
  . . .
</selection-criteria>
```

The string contains no whitespace (spaces, tabs, newlines, etc.) unless they are part of the attribute names, comparison operations, or match criterion that the user enters.

6.3 The MVStringEditor Widget

The `MVStringEditor` (`mved`) widget provides an editor control that can store multiple values. It provides the following benefits:

- ♦ Allows single source for a multivalue string edit (`mved`) control
- ♦ Customizes the control for the browser used
- ♦ Automatically detects the browser and sets the appropriate mode
- ♦ Allows the mode to be overridden

The mved supports the following Web browsers:

- ♦ Firefox* 1.5.x and later
- ♦ Internet Explorer 6 and later

6.3.1 How to Use the MVStringEditor Widget

- 1 Include the required mved JavaScript file.
- 2 Add the mved on the page.
- 3 Load values into the control.
- 4 Get values from the control.

For example:

```
01:<%@ page pageEncoding="utf-8" contentType="text/html; charset=utf-8" %>
02:<%@ taglib uri="/WEB-INF/iman.tld" prefix="iman" %>
03:<%@ taglib uri="/WEB-INF/c.tld" prefix="c" %>
04:
05:<HTML>
06:<HEAD>
07:    <LINK rel="stylesheet" href="<c:out value="\${ContextPath}" />/portal/
modules/dev/css/hf_style.css">
08:    <iman:mvedScripts/>
09:    <iman:eMFrameScripts/>
10:
11:    <SCRIPT>
12:        function onInit()
13:        {
14:            mvLoadFromPack("mved1", "<c:out value="\${myData}" />");
15:        }
16:
17:        function onExit()
18:        {
19:            document.form.mvedData.value = mvGetValuesAsPack("mved1");
20:        }
21:    </SCRIPT>
22:</HEAD>
23:
24:<BODY onLoad="onInit();">
25:<FORM name="form" method="post" action="webacc" onSubmit="return onExit();">
26:    <iman:mved name="mved1" width="150"/>
27:    <input type="hidden" name="mvedData"/>
28:</FORM>
29:</BODY>
30:</HTML>
```

Of particular interest in the above example are the following:

- 02: Includes iManager Tag Library
- 08: Includes javascripts methods required by MVStringEditor
- 14: Loads data into control
- 19: Retreives data from control

- 26: Including control on page
27: Hidden input for posting data to server

To pass values to the control with Java:

```
String values = ...;  
  
request.setAttribute("myItems", values);
```

To get values from the control with Java:

```
String pack = request.getParameter("mved1_packValues");  
  
String[] items = eMFrameUtils.unpack(pack, taskContext);
```

6.3.2 Modes

When including the `mved`, you must decide on a mode. If you do not set a mode, it chooses a mode for you based on the current device.

Table 6-6 *MVStringEditor modes*

Mode	Description
mvie	Fancy mved for holding and editing multiple values; normal mved for Internet Explorer. This is the default mode for the Browser device type.
mvsel	In this mode, the mved is based on the HTML select control. It has two modes: <ul style="list-style-type: none">♦ One line. This mode uses a smaller area. It is the default mode for the Simple device type. The user sees only the first value until clicking the drop-down arrow.♦ Combobox. This mode uses a larger area. It is composed of a text entry area and an expanded select box. It can be used with Browser or Simple device types when it is the only control on the page.

6.3.3 Parameters

Table 6-7 *MVStringEditor parameters*

Parameter	Description
name	Specifies a unique name identifier for the <code>mved</code> .
mode	Specifies one of three <code>mved</code> modes (<code>mvsel</code> , <code>mvie</code> , or <code>mvtxt</code>). If you do not specify a mode, the <code>mved</code> picks the mode best suited for the current device. This helps when single-sourcing JSP files across multiple devices.
width	Specifies the width of the control in pixels.
useRootedName	Specifies complete object name including the tree name.
bgColor	Specifies the color of the page around the control. The default is white. Use standard hex HTML color codes to specify the background color.
readonly	Boolean value that, when <code>True</code> , disables the control and dims the control text.

Parameter	Description
objectTypeName	Specifies an Object Selector typeFilter ("User", "*", etc.)
enforceUnique	Boolean value that, when <code>True</code> , forces all values in the list to be unique.
ignoreCase	Boolean value that, when <code>True</code> , instructs the control to ignore case when determining if a value is unique. For example, when <code>ignoreCase</code> is true, "ABC" and "abc" cannot both be added to the list. This attribute is used only when <code>enforceUnique</code> is <code>True</code> .
size	Specifies the number of rows to display in the select box. This attribute is used only when mode is <code>mvsel</code> .
numbersOnly	Boolean value that, when <code>True</code> , instructs the control to only support numeric values.
upperBound	Instructs the control to support only numeric values less than the specified value.
lowerBound	Instructs the control to support only numeric values greater than the specified value.
minLength	Instructs the control to support only string longer than the specified number of characters.
maxLength	Instructs the control to support only strings shorter than the specified number of characters.
xml	Specifies XML as a string to display to the user. The root node can be anything, but it must have child nodes of name value.
history	Displays a history button next to the OS value. This attribute is used only when <code>objectTypeName</code> is <code>True</code> .
disableEdit	Boolean value that, when <code>True</code> , disables the editing of control.
biggerBox	Boolean value that, when <code>True</code> , gives bigger text box. Used in conjunction with <code>Size</code> attribute which should be greater than 0.

6.3.4 JavaScript API

The following JavaScript methods allow you to interact with the MVStringEditor:

mvLoadFromPack (String name, String packedString) The `name` parameter must be the same as the `MVStringEditor_name` used for `MVStringEditor_inc`.

mvGetValuesAsPack (String name) : String (packed) The `name` parameter must be the same name as the `MVStringEditor_name` used for `MVStringEditor_inc`. The return value is a packed string, safe to be posted.

mvGetValuesAsXml (String name) : String (xml content) The `name` parameter must be the same as the `MVStringEditor_name` used for `MVStringEditor_inc`. The return value is XML in string format.

mvLoadFromXml (String name, String xml) The `name` parameter must be the same as the `MVStringEditor_name` used for `MVStringEditor_inc`.

mvEnable (String name)

mvDisable (String name)

The JavaScript methods `mvLoadFromPack` and `mvGetValuesAsPack` interact with packed Strings. An empty string is not a valid packed string. If you want an empty packed string, call `pack(new Array())`. There are methods in `eMFrameScripts` (JavaScript) and `eMFrameUtils` (Java) to pack and unpack these strings.

6.3.5 Examples

Following are some `MVStringEditor` examples::

Packed Strings

In JavaScript header:

```
oninit : mvLoadFromPack ("myControl", "<c:out value=\"${data}\"/>");
onexit: document.form.data.value = mvGetValuesAsPack ("myControl")
```

In HTML body:

```
<iman:mved name="myControl" ignoreCase="true" enforceUnique="true"/>
```

XML

In JavaScript header:

```
oninit: mvLoadFromXml("mycontrol", "<root><value>...</value></root>");
onexit: hiddenField = mvGetValuesAsXml("mycontrol");
```

In XML:

```
<root>
  <value>test1</value>
  <value>test2</value>
  <value>test3</value>
</root>
```

6.4 The Date/Time Widget

`iManager` includes a date/time widget that is implemented as a JSP tag library. To use the date/time widget:

- 1 Include the following tag in the `<head>` section of your JSP:

```
<iman:calendarScripts/>
```

- 2 Include the following tag in the `<body>` section of your JSP where you want the widget to appear:

```
<iman:calendar name="MyCal" onLoadCallback="onLoadCalendar"
returnCallback="returnFromCalendar"/>
```

To easily create a template that uses the date/time widget, use the Plug-in Studio to create a template that modifies an attribute of time syntax.

Creating a Plug-In

7

This section covers the following topics related to creating a plug-in:

- ♦ [Section 7.1, “Overview,” on page 81](#)
- ♦ [Section 7.2, “Creating a Manifest File,” on page 83](#)
- ♦ [Section 7.3, “Plug-In Information Through XML,” on page 85](#)
- ♦ [Section 7.4, “Creating a Plug-In Installer,” on page 87](#)
- ♦ [Section 7.5, “Installing Plug-ins to an Existing iManager,” on page 88](#)
- ♦ [Section 7.6, “Precompiling JSPs for Tomcat 5,” on page 90](#)
- ♦ [Section 7.7, “Testing a Plug-In,” on page 92](#)

7.1 Overview

A plug-in typically provides all the management functionality that a particular product, or feature set within a product, requires. It is assembled as a single file so you can quickly and easily add extend iManager to support the required management functionality.

However, sometimes it makes sense to bundle multiple plug-ins together, as Novell does for Novell® eDirectory™ and Novell Identity Manager™. Plug-in bundling might make sense in the following situations:

- ♦ Make it easier to distribute all the plug-ins for a given product
- ♦ Plug-ins have dependencies on other plug-ins or shared code

You can create plug-in files using the `Jar` program included in the JDK. Access and manipulate the resulting `.jar` files using any tool that supports the ZIP archive format.

The following sections explain concepts related to plug-ins:

- ♦ [Section 7.1.1, “Plug-In File Structure,” on page 81](#)
- ♦ [Section 7.1.2, “Plug-In Directory Objects and Attributes,” on page 83](#)
- ♦ [Section 7.1.3, “Plug-in Update and Uninstallation,” on page 83](#)
- ♦ [Section 7.1.4, “Plug-In Installers,” on page 83](#)

7.1.1 Plug-In File Structure

A plug-in is a Java archive (JAR) file named with an `.npm` extension. Like all JAR files, a plug-in combines multiple files into one file and stores the relative path of each included file. The combination of the file names and the path information form a directory hierarchy. iManager imposes a directory hierarchy by requiring that some categories of files be placed within specific directories in the plug-in.

Most of the files that you would typically include in a plug-in, such as Java class files, JSPs, registration files, resources, and a manifest file, must be organized into special directories according to the file type or function. These directories are described in the following table:

Table 7-1 Plug-in directories

File or Directory	Description
currentwebapp	Contains files that are installed as part of the overall iManager Web application. These files are copied to the webapps directory of the servlet container. For example, tomcat/webapps/nps.
META-INF	(Required) Contains the required manifest file (MANIFEST.MF) with configuration information about the plug-in.

IMPORTANT: The case of the directory names and files is significant.

The structure of a plug-in that includes all of the special directories is represented in the following figure.

Figure 7-1 Plug-in structure showing all special directories.



Any files that do not need to be placed in one of the special directories, such as files for background services, can be included in a plug-in and organized in any way that meets your needs.

iManager 2.7.2 provides a framework for the plug-ins to extend the eDirectory schema during the plug-in installation. So, if a plug-in wants to extend the eDirectory schema during its installation, it has to follow a set of guidelines in its structure which are required by the schema extension framework.

- ♦ Plug-in should provide the definition of the schema it wants to extend in either a .LDIF (single file) or a .SCH (single file) format.
- ♦ The schema definition file should be placed under `ModuleName.npm/currentwebapp/schema/` folder of the npm.

After the successful installation of the plug-in, if there is a schema file in the expected location, imanager extends the schema defined in the directory to which iManager is logged in during the plug-in installation.

NOTE: The above description is for [Section 7.5.2, “Installing a Plug-In Manually,” on page 89](#). For instructions on extending the schema during programmatic plug-in installation, see [Section 7.5.1, “Installing a Plug-In Programmatically,” on page 89](#).

Deleting files after Plug-in installation by using command-line files

A few iManager plug-ins bundle certain command-line files such as `symboliclinks.txt` and `permissions.txt` in the `currentwebcommand` directory of their `.npm` file. Another type of command-line file `deletetfiles.txt` can now be bundled in the same directory which has a list

of files to be deleted. Any file that is in the `<iManager_tomcat>/webapps/nps` folder can be deleted. So, the name of the files should be relative to the `/nps` folder. The files are deleted at the end of plug-in installation.

Limitations:

- ♦ The names of only files can be specified. The file names should not contain any wild card characters such as `*`, `?`, and so on. If a directory name is specified, it is not deleted because the directory would not be empty.
- ♦ Sometimes on Windows, a few files are not deleted after the plug-in is installed. This is because the files are locked by iManager (tomcat). These files should be manually deleted.

7.1.2 Plug-In Directory Objects and Attributes

iManager creates a `bhModule` object in the directory to represent a plug-in. This object is registered with the portal object in the directory so that when a plug-in is installed, updated, or deleted, iManager can replicate the NPM file changes to all iManager servers servicing the portal object. The `bhModule` object can also store plug-in-specific configuration information and links to other plug-ins that are bundled with the plug-in.

In addition, other objects have attributes that link them to a plug-in.

7.1.3 Plug-in Update and Uninstallation

When a plug-in is updated, plug-ins contained in the NPM file are also updated. iManager uses the `bhGadgets` and `bhModules` attributes to identify the `.npm` and `.npg` files that correspond to each plug-in.

When a plug-in is uninstalled, all tasks within the plug-in are also uninstalled. As an option, the administrator can uninstall individual plug-ins within a plug-in bundle.

The uninstallation of a plug-in causes plug-ins in the file system to be deleted. File system dependency lists can be created during the uninstallation so that files that appear in other NPM archives are not deleted.

7.1.4 Plug-In Installers

A plug-in installer enables plug-ins to perform installation and configuration tasks during the installation of the portal. To learn how to create a plug-in installer, see [Section 7.4, “Creating a Plug-In Installer,” on page 87](#).

7.2 Creating a Manifest File

A manifest file contains plug-in configuration information such as the location of the plug-in install class. Every plug-in must contain a manifest file, named `MANIFEST.MF`, in a directory named `META-INF`. The format of the manifest file follows the [JAR File Specification \(http://java.sun.com/j2se/1.4.1/docs/guide/jar/jar.html\)](http://java.sun.com/j2se/1.4.1/docs/guide/jar/jar.html). A manifest file looks similar to the following:

```

Manifest-Version: 2.7
Implementation-Vendor: Novell, Inc.
Module-ID: iPrintX
Created-By: iPrint team
Implementation-Title: iPrint Linux Management
Implementation-Description: Allows management of iPrint services on Linux.
Main-Class: com.novell.admin.iPrint.iPrintLinuxInstaller
Min-iManager-Version: 2.7.0
Max-iManager-Version: 2.8.0
Implementation-Version: 2.7.0.20070625
Supported-OS:
NetWareServer;LinuxServer;WindowsServer;WindowsWorkstation;LinuxWorkstation

```


iManager reads the following entries in the `MANIFEST.MF`:

Table 7-2 *MANIFEST.MF entries*

Entry	Description
Module-ID	<p>(Required) Specifies an identifier for the plug-in. All versions and releases of the plug-in should specify the same <code>Module-ID</code>, which iManager uses to determine if the plug-in replaces an existing plug-in. If <code>Module-IDs</code> differ, iManager does not replace the previously installed plug-in.</p> <p>Use the following guidelines when creating a <code>Module-ID</code>:</p> <ul style="list-style-type: none"> ♦ Should be lower-case with no spaces. ♦ Should identify the product to which the plug-in is related ♦ Should not include version numbers
Implementation-Title	<p>(Required) Specifies a brief (48 characters or less) plug-in description that displays in iManager's plug-in list.</p> <p>Use the following guidelines when creating an <code>Implementation-Title</code>:</p> <ul style="list-style-type: none"> ♦ Should be easily readable. Spaces are OK. ♦ Should be descriptive enough so users know the product to which this plug-in belongs ♦ Indicate if the plug-in is platform-specific ♦ Do not include product version numbers
Implementation-Description	<p>(Required) Provides a description of the plug-in. Use this entry to list any plug-in dependencies.</p>

Entry	Description
Implementation-Version	<p>(Required) Specifies the plug-in version. The format is <i>major.minor.revision.build</i>, where <i>major</i>, <i>minor</i>, <i>revision</i>, and <i>build</i> are integers (up to 8 digits). For example, 4.1.0.20070926.</p> <p>Higher version numbers should indicate more recent versions. The relative age of two plug-ins is determined by comparing the Implementation-Version entry of the manifest file using the following algorithm:</p> <ol style="list-style-type: none"> 1. If the integer value of <i>major</i> in one plug-in is greater than the integer value of <i>major</i> in the other, then the one with the greater <i>major</i> value is more recent. If they are equal, proceed to Step 2. 2. If the integer value of <i>minor</i> in one plug-in is greater than the integer value of <i>minor</i> in the other, then the one with the greater <i>minor</i> value is more recent. If they are equal, proceed to Step 3. 3. If the integer value of <i>revision</i> in one plug-in is greater than the integer value of <i>revision</i> in another, then the one with the greater <i>revision</i> value is more recent. If they are equal, the two plug-ins are the same version. 4. If the integer value of <i>build</i> in one plug-in is greater than the integer value of <i>build</i> in another, then the one with the greater <i>revision</i> value is more recent. If they are equal, the two plug-ins are the same version. <p>When a user attempts to install a plug-in, iManager checks the version number of the currently installed plug-in. If the version of the plug-in being installed is equal to or greater than the version of the current plug-in, the plug-in is installed. If the plug-in version is smaller, the plug-in is not installed.</p>
Min-iManager-Version	(Required) Specifies the minimum version of iManager required to run this plug-in.
Max-iManager-Version	(Optional) Specifies the maximum version of iManager required to run this plug-in.
Supported-OS	(Optional) Specifies a semi-colon-delimited list of operating systems on which this plug-in can install and run. Leave this list blank to install and run on all operating systems.
Main-Class	<p>(Optional) A standard entry, used by Java, that specifies a class to run if the plug-in is run as a <code>.jar</code> file (executed in the following manner: <code>java -jar module_name.npm</code>). If the main class implements <code>PortalModuleInstaller</code> (and, optionally, <code>PortalModuleUI</code>), it is instantiated by the plug-in installation code and called at key points during installation so you can perform plug-in-specific operations.</p> <p>For more information about plug-in installers, see Section 7.1.4, “Plug-In Installers,” on page 83 and Section 7.4, “Creating a Plug-In Installer,” on page 87. Required for all plug-ins with installer classes.</p>
RBS-DisplayName	(Optional) Specifies a display name to show in the Configuration Wizard and iManager’s list of installed plug-ins. If multiple plug-ins have the same RBS-DisplayName, iManager installs them as a package.

7.3 Plug-In Information Through XML

In the Available Novell Plug-in Modules page, under Novell Plug-in Modules, click the  icon next to the plug-in to display the plug-in information such as, supported platforms, recommended versions, change logs, build number, and other miscellaneous information.

To get the icon for a plug-in:

- 1 Create an XML file similar to the following Sample XML file:

Sample XML:

```
<info>
<miscellaneous>blah blah</miscellaneous>
<recommended_versions>2.6 imanager</recommended_versions>
<supported_platforms>linux</supported_platforms>
<change_log>
<build_number>20061025</build_number>
<entry>
Fixed an issue where HP printers don't show up correctly.
</entry>
<entry>
Fixed a problem where the iPrint plugin would not uninstall.
</entry>
<entry>
Added the ability to import drivers in Atari 2600 format.
</entry>
</change_log>
<change_log>
<build_number>20060516</build_number>
<entry>Fixed the plugin so that it actually works</entry>
<entry>
The plugin now interprets the :) as an actual smily face.
</entry>
</change_log>
<special_instruction>
This plugin is the same version that ships with OES SP9.56.
</special_instruction>
<special_instruction>
Before installing this plugin, you must stand on your head and
clap your hands 3 times while reciting the words to your
favorite nursery rhyme.
</special_instruction>
<dependencies> lots and lots of them!!</dependencies>
</info>
```

- 2 Specify the location (<http://{serverip}/sample.xml>) of the XML file that you have created, in the `iman_mod_desc.xml` file which contains information of all the iManager plug-ins.

The sample iman_mod_desc.xml file looks like:

```
<modules>
<module>
<moduleID>eDirectoryRepairAndLogfile</moduleID>
<title>eDirectory Repair and Logfile</title>
<filename>dsrepair.npm</filename>
<depends-on>FTPAdmin</depends-on>
<depends-on>NFSAdmin</depends-on>
<depends-on>FakeModule</depends-on>
<version>2.7.20070831</version>
<min-required-version>2.7.0</min-required-version>
<url type="zip">ftp://ftp.novell.com/outgoing/2.zip</url>
<info_url>http://164.99.166.23/artifacts/about.xml</info_url>
<description>
Contains repair and logfile eDirectory functionality
</description>
</module>
<module>
<moduleID>FTPAdmin</moduleID>
<title>FTP</title>
<filename>FTPAdmin.npm</filename>
<version>1.1.20070828</version>
<depends-on>eDirectoryRepairAndLogfile</depends-on>
<min-required-version>2.7.0</min-required-version>
<url type="zip">ftp://ftp.novell.com/outgoing/2.zip</url>
<info_url>http://164.99.166.23/artifacts/about.xml</info_url>
<description>Allows management of FTP</description>
</module>
</modules>
```

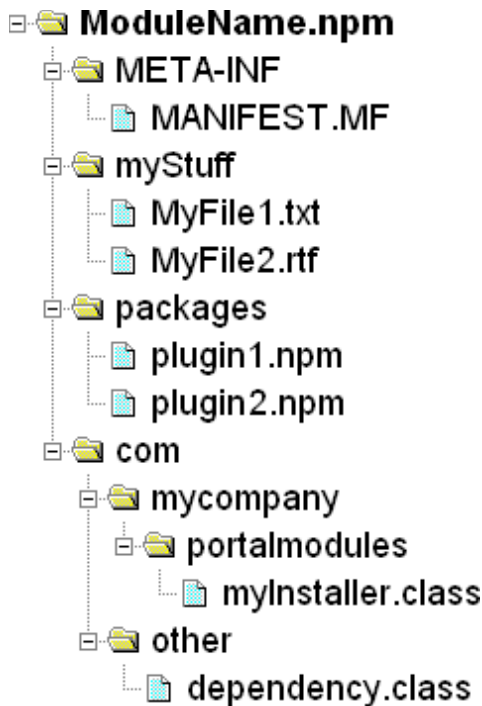
7.4 Creating a Plug-In Installer

A plug-in installer enables plug-ins to perform installation and configuration tasks during part of the installation of iManager.

An installer is a Java class that implements the `com.novell.nps.serviceProviders.PortalModuleInstaller` interface. During installation of the plug-in, the `PortalModuleInstallManager` class calls the methods in your installer class that are prescribed by the `PortalModuleInstaller` interface.

The installer class must be added to the plug-in. The following figure shows the file structure of a plug-in that contains an installer class.

Figure 7-2 File structure of a plug-in with an installer class



The following example shows a manifest file for a plug-in with an installer class:

```
Module-ID: AddrAdvertise
Implementation-Title: NCP Server Address Advertising
Implementation-Description: NCP Server Address Advertising Plugin
Implementation-Version: 1.0.0
Main-Class: com.mycompany.portalmodules.myInstaller
Depends-on: plugin1,plugin2 (Optional)
```

7.5 Installing Plug-ins to an Existing iManager

You can install an NPM into an existing iManager environment. However, the NPM you plan on installing must be compatible with the version of iManager currently installed. You can determine the currently installed version of iManager by calling `eMFrameUtils.getVersion(TaskContext context)`.

Alternatively, you can retrieve the current version of iManager from `version.properties`, as shown in the following example:

```
public static Version getVersion(TaskContext context)

{
    String versionStr =
eMFrameUtils.getLocalizedString("com.novell.emframe.version", "version", context);
    return new Version(versionStr);
}
```

You can install plug-in programmatically or manually. If you install plug-in manually, the process varies depending on whether or not you have RBS configured. For more information about RBS, see [“Role-Based Services” on page 20](#).

To install a plug-in programmatically, see “Installing a Plug-In Programmatically” on page 89. To install a plug-in manually, see “Installing a Plug-In Manually” on page 89.

7.5.1 Installing a Plug-In Programmatically

- 1 To programmatically install a plug-in into iManager, use one of the following calls:
 - ♦ `ModuleManager.externalInstall(File sWebAppContextPath, File npmFile, File fJspPath)`
 - ♦ `ModuleManager.externalInstall(File sWebAppContextPath, File npmFile, File fJspPath, boolean silentInstall)`
 - ♦ `ModuleManager.externalInstall(File sWebAppContextPath, File npmFile, File fJspPath, String LOGIN_DN, String HOST, String PASSWORD)`
 - ♦ `ModuleManager.externalInstall(File sWebAppContextPath, File npmFile, File fJspPath, boolean silentInstall, String LOGIN_DN, String HOST, String PASSWORD)`
- 2 Make sure that all the plug-ins are in the local directory because the command line installation checks only the local directory for plug-ins. This completely resolves the dependencies.
Dependencies are identified through Depends-on element of the MANIFEST file from the npm.
If there are any unavailable dependent plug-ins, you are prompted.
- 3 After getting the set of dependent plug-ins, you are prompted to continue the installation of main plug-in by installing all dependent plug-ins, install only the main plug-in, or terminate the installation process. Select the desired option.

NOTE: To programmatically uninstall a plug-in from iManager, use the following call:

```
ModuleManager.externalUninstall(File sWebAppContextPath, String  
npmFileName)
```

7.5.2 Installing a Plug-In Manually

- 1 Log in to iManager.
- 2 Click *Configure > Plug-in Installation > Available Novell Plug-in Modules*.
The Available Novell Plug-in Modules page is displayed.
- 3 Under *Novell Plug-in Modules*, click *Add*, browse for the appropriate NPM file, then click *OK*.
- 4 In the Available Novell Plug-in Modules page, select the NPM and click *Install*.
- 5 Restart Tomcat.

Platform	Restart Command
NetWare® 6.5 or later	Enter <code>tomcat5 stop</code> . Wait at least one minute, then enter <code>tomcat5 start</code> to start the service again.
Windows*	Stop and start the Tomcat service.
Linux	Enter <code>/etc/init.d/novell-tomcat5 restart</code> .

Tomcat takes some time to fully initialize. Wait a couple minutes before trying to log in to iManager.

- 6 (Conditional) If RBS is configured, do the following:
 - 6a Log in to iManager, then click the Configure button.
 - 6b Select *Role-Based Services > RBS Configuration*.
 - 6c To update an RBS module, select the number in the Out-of-Date column for the Collection you want to update.

The list of outdated plug-ins is displayed.
 - 6d Select the plug-in you want to update and then click *Update* at the top of the table.
 - 6e Repeat steps 8c and 8d for the Not-Installed column of the RBS Configuration page.
- 7 Verify that the new Role appears in the Roles and Tasks page.

To add members to the new Role, select *Role Based Services > Edit Member Association* in the navigation frame.

7.6 Precompiling JSPs for Tomcat 5

This section discusses the Tomcat 5 compiled JSP class file path changes and using Ant to precompile JSP files.

To avoid the delay Tomcat takes to compile and load a JSP file in iManager you can precompile the .jsp file into a Java class file. Deliver the precompiled Java class files with your plug-in and install them to the appropriate Tomcat 5 directory.

7.6.1 Tomcat 5 Compiled Java Class File Location

The Tomcat 5 path for iManager plug-ins is `TOMCAT_HOME/work/Catalina/localhost/nps/org/apache`. Previously, Tomcat 4 stored iManager plug-ins in `TOMCAT_HOME/work/Standalone/localhost/nps`. For example:

- ♦ Compiled DeleteClass.jsp path using Tomcat 4: `TOMCAT_HOME/work/Standalone/localhost/nps/portal/modules/base/skins/default/devices/default/DeleteClass_jsp.class`
- ♦ Compiled DeleteClass.jsp path using Tomcat 5: `TOMCAT_HOME/work/Catalina/localhost/nps/org/apache/jsp/portal/modules/base/skins/default_/devices/default_/DeleteClass_jsp.class`

To determine the path for a specific JSP class file:

- 1 Log into iManager.
- 2 Select the task or page that includes the JSP file.

The first time you access page, Tomcat compiles the JSP into a .java file, then a .class file.
- 3 Browse `TOMCAT_HOME/work/Catalina/localhost/nps/org/apache/` for the specific class file.

This is the path you would use when precompiling your JSP.

7.6.2 Precompiling JSP Pages with Ant

The iManager 2.7 build process uses Ant to precompile the JSP files. For information about using Ant to precompile JSPs, see [Web Application Compilation \(http://tomcat.apache.org/tomcat-5.0-doc/jasper-howto.html#Web%20Application%20Compilation\)](http://tomcat.apache.org/tomcat-5.0-doc/jasper-howto.html#Web%20Application%20Compilation) in the *Tomcat 5 Jasper 2 JSP Engine How To*.

Use the sample script found there to pass in the appropriate parameters for *TOMCAT_HOME* and *WEBAPP_PATH* and precompile your JSP files. You can modify this script and include in to your build process.

The following examples show an iManager Ant build script and build target. The compiled JSP class files are copied to a working directory and then, later in the build process, zipped up in the .npm file.

IMPORTANT: Because iManager 2.7 plug-ins need to run on NetWare and the 1.4 JDK, you should compile your JSP files with the source and target set to version “1.4” as shown in the above example. This will ensure that the precompiled files will work on all of the iManager 2.7 supported platforms.

iManager Ant build script named `jspc`:

```
<target name="jspc" depends=".setproperties">
  <taskdef classname="org.apache.jasper.JspC" name="jasper2" >
    <classpath id="jspc.classpath">
      <pathelement location="${java.home}/../lib/tools.jar"/>
      <fileset dir="${tomcat.home}/bin">
        <include name="*.jar"/>
      </fileset>
      <fileset dir="${tomcat.home}/server/lib">
        <include name="*.jar"/>
      </fileset>
      <fileset dir="${tomcat.home}/common/lib">
        <include name="*.jar"/>
      </fileset>
    </classpath>
  </taskdef>

  <jasper2
    webXml="${webapp.path}/WEB-INF/web.xml"
    validateXml="false"
    uriroot="${webapp.path}"
    webXmlFragment="${webapp.path}/WEB-INF/generated_web.xml"
    outputDir="${webapp.path}/WEB-INF/work/Catalina/localhost/${Product.Name}"/>
</target>
```

iManager Ant build target named `compilejsps`.

```
<target name="compilejsps" depends="jspc">
  <javac destdir="${webapp.path}/WEB-INF/classes"
    optimize="off"
    source="1.4"
    target="1.4"
    debug="on"
    failonerror="false"
    srcdir="${webapp.path}/WEB-INF/work/"
```

```

excludes="**/*.smap">
<classpath>
  <pathelement location="${webapp.path}/WEB-INF/classes/org"/>
  <fileset dir="${webapp.path}/WEB-INF/lib">
    <include name="*.jar"/>
  </fileset>
  <pathelement location="${tomcat.home}/common/classes"/>
  <fileset dir="${tomcat.home}/common/lib">
    <include name="*.jar"/>
  </fileset>
  <pathelement location="${tomcat.home}/shared/classes"/>
  <fileset dir="${tomcat.home}/shared/lib">
    <include name="*.jar"/>
  </fileset>
  <fileset dir="${tomcat.home}/bin">
    <include name="*.jar"/>
  </fileset>
</classpath>
  <include name="*" />
  <exclude name="tags/*" />
</javac>
  <copy preservelastmodified="yes" todir="${webapp.path}/WEB-INF/work/Catalina/
localhost/nps/org" >
    <fileset dir="${webapp.path}/WEB-INF/classes/org" />
  </copy>
</target>

```

7.7 Testing a Plug-In

Once you have created a new iManager plug-in, you should test the plug-in against the various iManager features to make sure it functions properly. Often, plug-in developers focus only on making the plug-in available through iManager's navigation frame in the Roles and Tasks view when there are other integration points that should be considered as well.

Consider testing your plug-in in the following ways to make sure it performs properly:

- ♦ [Section 7.7.1, “Installation Testing,” on page 92](#)
- ♦ [Section 7.7.2, “Plug-In Studio Testing,” on page 95](#)
- ♦ [Section 7.7.3, “Individual Task Testing,” on page 96](#)
- ♦ [Section 7.7.4, “Object View Testing,” on page 96](#)
- ♦ [Section 7.7.5, “Generic Operation Testing,” on page 96](#)

7.7.1 Installation Testing

You should test your plug-in installation and upgrade capabilities both during the iManager installation process, and after iManager is installed. Additionally, you should test installing your plug-in by downloading it and by installing it from a local drive.

Furthermore, if you bundle multiple plug-ins together into a single `.npm` for delivery, make sure to test the plug-in both as part of the parent installation (installing all plug-ins as a group), and as an individual (installing a single plug-in from the group) installation if that option is available.

NOTE: Linux is a good testing platform because of its file name case-sensitivity. Windows and NetWare do not use case-sensitive file names.

This section includes the following topics:

- ♦ “Testing During the iManager Installation” on page 93
- ♦ “Testing After the iManager Installation” on page 93
- ♦ “Creating a Test Download.xml File” on page 94

Testing During the iManager Installation

To test your plug-in installation or upgrade as part of the iManager install, you should both install a new iManager server, and upgrade an existing iManager server. As part of the install or upgrade process, users can install plug-ins from Novell’s download site or from the local file system.

To ensure your plug-in installs properly during a new iManager installation (not an upgrade), do the following:

- ♦ Test downloading and installing your plug-in. For more information, see “Creating a Test Download.xml File” on page 94.
- ♦ Install your plug-in from a local directory specified during the iManager installation routine.

To ensure your plug-in installs properly during an iManager upgrade (from iManager 2.6 to 2.7, for example), do the following:

- ♦ Test downloading and installing your plug-in. For more information, see “Creating a Test Download.xml File” on page 94.
- ♦ Test installing your plug-in from a local directory specified during the iManager installation routine.
- ♦ Test upgrading iManager with a previous version of your plug-in installed. If a new plug-in is available on Novell’s download site during the iManager upgrade, it should be checked automatically and notify the user that the plug-in update is available.
- ♦ Test upgrading iManager without a previous version of your plug-in installed. Unless the `download.xml` file specifies `selected=true`, your plug-in is not checked by default. For example, `<moduleID selected="true">pwdpolicy</moduleID>`.

Testing After the iManager Installation

To test your plug-in installation or upgrade after installing iManager, use iManager Workstation. This lets you shutdown and restart iManager as needed during the testing process. You can also easily re-install iManager Workstation, if necessary.

Once you install iManager, you can install a plug-in three different ways. You should test your plug-in using each of these installation scenarios:

- ♦ Manually copy the plug-in to `<TOMCAT_HOME>/webapps/nps/packages`. This makes the plug-in available from *Plug-in Installation > Available Novell Plug-in Modules* in the Configure view. Make sure you refresh the page to see the new plug-in listing.

- ♦ Select Add from the *Plug-in Installation > Available Novell Plug-in Modules* in the Configure view. This uploads the plug-in to <TOMCAT_HOME>/webapps/nps/packages.
- ♦ Download the plug-in (see the steps below on how to set this up for testing). For more information, see “[Creating a Test Download.xml File](#)” on page 94.

Creating a Test Download.xml File

During the iManager installation, the installation routine displays a URL to an XML file on the plug-in download page. Once you install iManager, you can modify the `config.xml` file to point to a custom XML file for downloading plug-ins.

Once you create a custom `download.xml`, instruct iManager, or the iManager installation routine, to use this file for downloading plug-ins by modifying the following setting in iManager’s <TOMCAT_HOME>\webapps\nps\WEB-INF\config.xml. Restart Tomcat after making the change.

```
<setting>
  <name><![CDATA[ModuleDownloadDescriptorURL]]></name>
  <value><![CDATA[http://<IP_Address>/iManager_plugins/test.xml]]></value>
  <!--<value><![CDATA[file:///D:\builds\iManager\test\test.xml]]></value>-->
</setting>
```

The <value> tag can specify either an HTTP or file location. The default URL for downloading iManager plug-ins is:

`http://www.novell.com/products/containers/imanager/iman_mod_desc.xml`

NOTE: Once you create a custom `download.xml` file, test that it is syntactically correct by opening it in a Web browser.

The `download.xml` looks similar to the following:

```
<modules>

<module>
  <moduleID>CaseSensitivePassword</moduleID>
  <filename>CSP.npm</filename>
  <version>2.5.20050908</version>
  <min-required-version>2.6</min-required-version>
  <url type="zip"><![CDATA[file:///D:\builds\iManager\test\Dir_88_iMan26_Plugins.npm]]></url>
  <description>Case Sensitive Password Plugin</description>
</module>

<module>
  <moduleID>novell_imanager_plugins</moduleID>
  <filename>identity_manager_plugins.npm</filename>
  <version>10.0.20060302</version>
  <min-required-version>2.5</min-required-version>
  <url><![CDATA[file:///D:\builds\iManager\test\identity_manager_plugins.npm]]></url>
  <description>Novell Identity Manager 3 Plugins</description>
</module>

<module>
  <moduleID selected="true">pwdpolicy</moduleID>
```

```

    <filename>pwdpolicy.npm</filename>
    <version>10.0.20060302</version>
    <min-required-version>2.5</min-required-version>
    <url type="zip"><![CDATA[file:///
D:\builds\iManager\test\identity_manager_plugins.npm]]></url>
    <description>Universal Password Management Plugin (requires IDM - Common
Utilities Plugin)</description>
</module>

<module>
    <moduleID selected="true">DirXMLCommon</moduleID>
    <filename>DirXMLCommon.npm</filename>
    <version>10.0.20060302</version>
    <min-required-version>2.5</min-required-version>
    <url type="zip"><![CDATA[file:///
D:\builds\iManager\test\identity_manager_plugins.npm]]></url>
    <description>IDM - Common Utilities Plugin (needed for Universal Password
Plugin)</description>
</module>

<module>
    <moduleID>SecretStore</moduleID>
    <filename>secretstore.npm</filename>
    <version>10.0.20051209</version>
    <min-required-version>2.5</min-required-version>
    <url>http://137.65.135.100/secretstore.npm</url>
    <description>Secret Store Administration Plugin</description>
</module>

```

NOTE: The download.xml file includes a `<url>` tag with a conditional `type` definition. The `type` definition is required for any plug-in that is part of a plug-in bundle, but that you want to be able to install separately. It specifies the type of compression (zip or targz) used with the individual plug-in files.

7.7.2 Plug-In Studio Testing

iManager's Plug-in Studio lets you create additional tasks or property book pages (see [Chapter 5, "Using the Plug-In Studio," on page 53](#).) Plug-in Studio is often used to create custom tasks or property book pages that simplify or limit the set of task available in an existing plug-in, or combine certain tasks from multiple plug-ins into a single interface. Because of this, newly created tasks or property book pages can contain attributes from your plug-in or product.

To test your plug-in functionality with Plug-In Studio:

- 1 Launch iManager and make sure that RBS is installed (see ["Setting Up Role-Based Services" on page 14](#).)
- 2 From the Configure view, select *Role Based Services > Plug-in Studio*.
- 3 Click *New* to open the Create iManager Task Wizard.
- 4 Create a custom task that includes attributes from your plug-in.
For information about using Plug-in Studio, see ["Creating a Custom Plug-In" on page 54](#).
- 5 Test the new Plug-in Studio task to make sure it can properly access attributes from your plug-in.

7.7.3 Individual Task Testing

A plug-in can contain multiple roles with multiple tasks. Each task in a plug-in can set specific RBS rights that allows the assigned user to perform the task. If you are assigned a role that contains more than one task, your RBS rights for that role are the combined rights for all of the tasks associated with the role.

Testing your plug-in only with default role and task assignments can mask mask rights issues with individual tasks.

To test individual tasks within your plug-in:

- 1 Create a custom RBS role.
- 2 Assign one of your plug-ins individual tasks to the role.
- 3 Assign the custom RBS role to a user that does not have any rights granted via trustee assignments or through any other roles assignments.
- 4 Login as the user and verify that you can use the task without any rights issues.

You can also check the rights RBS assigns to a task by opening the Configure view and selecting *RBS Reporting > Role Rights Assignments*.

7.7.4 Object View Testing

The Object view lets you drill down to a specific eDirectory object first, then select the task you want to perform. You should test your plug-in to make sure it functions properly when users access plug-in tasks through the Object view.

In most cases, any task available from the Roles and Tasks view should be accessible from the Object view. The name of the selected object should be passed to the task's object name field (if this is part of the tasks functionality.) Additionally, Object view supports a multi-select mode that you should test with your plug-in. Some tasks don't make sense in a multi-select scenario, so there might be a smaller list of available tasks for your plug-in in that case.

7.7.5 Generic Operation Testing

The Directory Administration role contains the tasks used with any object type: Create Object, Copy Object, Modify Object, Rename Object, Move Object, and Delete Object. Because of this, it is possible for these generic tasks to interact with objects improperly. For example, a generic copy operation copies all attributes of an existing object to a new object, even if some of those attributes are instance-specific and not appropriate to copy.

Test the interaction of all object types specific to your plug-in against iManager's generic task operations to make sure the generic tasks don't corrupt the objects or cause any other unwanted issues.

Installing iManager with Your Application

8

If you have developed an application that leverages iManager as its management platform, you might want to include the installation of iManager as part of your application install process. This section introduces the `installer.properties` file, and describes how to launch the iManager install from within your application's installation routine.

This section includes the following topics:

- ♦ [Section 8.1, “Windows,” on page 97](#)
- ♦ [Section 8.2, “Linux \(SUSE and Red Hat\),” on page 98](#)
- ♦ [Section 8.3, “Installer.properties File,” on page 98](#)

8.1 Windows

1. Obtain the latest platform-specific archive files for iManager install. These files are available from [the Novell Developer Support Web site \(http://developer.novell.com/support/\)](http://developer.novell.com/support/).
2. Include the iManager install archive as part of your product install. This can be done in multiple ways. For example, as an expanded directory hierarchy on your CD; bundled with your application installer for later extraction; or downloaded during the application install, then extracted.
3. Expand the iManager install archive. If your application installer uses InstallAnywhere, the archive can be expanded using the Expand Archive action, or from custom code using the `java.util.zip` package. Other installers have similar APIs for expanding compressed archive files.
4. Launch the iManager installer. The installer is located in `SDK_HOME\installs\PLATFORM` directory, where `PLATFORM` can be either win or unix.

Operating System	Install Executable
------------------	--------------------

Windows	iManagerInstall.exe, located in the \win folder
---------	---

If you are using InstallAnywhere, the appropriate executable is launched using the Execute Command action or from custom code using `Runtime.exec()` and `Process` class methods. Add `-i silent` to the command line to run a silent install. You can also pass parameters (such as the UI mode) to the iManager installer through an `installer.properties` file. The `installer.properties` file must reside in the same directory as the iManager installer.

Set the `$LAUNCH_BROWSER$` property to `FALSE`, either in `installer.properties` or from the command line, if iManager should not launch a Web browser and display `gettingstarted.html` at the end of the install process. For example:

```
-DLAUNCH_BROWSER=false
```

5. To include a plug-in for automatic installation after the iManager installation is complete, simply place the necessary NPMs in the plug-in directory. By default, the plug-in directory is `SDK_HOME\installs\plugins` directory. However, you can also specify a different plug-in directory by setting the `PLUGIN_DIRECTORY` property, either in the `installer.properties` file or from the command line.
6. Once you have completed the installation, edit or create the following configuration file, which ensures that administrative rights are properly configured for use with eDirectory:

```
C:\Program Files\Novell\tomcat\webapps\nps\WEB-INF\configiman.properties
```

The `configiman.properties` file should specify the list of authorized users, in the following form: `user.context.treename=eDirectory`. For example:

```
admin.lab.LAB_TREE=eDirectory
```

8.2 Linux (SUSE and Red Hat)

1. Obtain the latest platform-specific archive files for iManager install. These files are available from [the Novell Developer Support Web site](http://developer.novell.com/support/) (<http://developer.novell.com/support/>).
2. Include the iManager install archive as part of your product install. This can be done in multiple ways, including as an expanded directory hierarchy on your CD; bundled with your application installer for later extraction; or downloaded during the application install, then extracted.

Alternatively, you can extract only those iManager RPMs needed by your application and include those as part of your application installation. All iManager RPMs are in the `packages` folder of the archive. The iManager RPM includes a dependency list for the required RPMs that can help you determine whether an RPM is necessary for your particular situation. All plug-in RPMs in the `packages` folder are installed.

NOTE: Using the RPMs to install iManager does not configure Tomcat. The Tomcat `server.xml` file (located in `/var/opt/novell/tomcat5/conf`) must be modified. If needed, you can also change port numbers.

3. Once you have completed the RPM installation, edit or create the following configuration file, which ensures that administrative rights are properly configured for use with eDirectory:

```
/var/opt/novell/iManager/nps/WEB-INF/configiman.properties
```

The `configiman.properties` file should specify the list of authorized users, in the following form: `user.context.treename=eDir`. For example:

```
admin.lab.LAB_TREE=eDir
```

8.3 Installer.properties File

After a successful installation, the iManager installer generates an `installer.properties` configuration file with values based upon the questions asked during the install. You can use this file as a template to create a custom `installer.properties` files for use when installing iManager together with another application.

An `installer.properties` file looks something like the following. A comment describes the file entries directly above it.

```

CONFIGIMAN_DIRECTORY=none

#specifies the default directory tree (all lowercase)

CONFIGIMAN_USER_CONTEXT=none

#Specifies the authorized user of format, admin.novell

TOMCAT_HTTP=8080

#Specifies the HTTP tomcat port

TOMCAT_SSL=8443

#Specifies the tomcat SSL port

TOMCAT_JK=9009

#Specifies the Tomcat jk connector port
PLUGIN_INSTALL_MODE=BOTH

#Defines the plug-in installation method. Options are DISK, NET, SKIP, or BOTH.

PLUGIN_DIRECTORY=/home/user1/plugins

#Specifies the plug-in install directory. Only used when PLUGIN_INSTALL_MODE
equals DISK or BOTH.

PLUGIN_INSTALL_URL=http://www.novell.com/products/containers/imanager/
iman_mod_desc_beta.xml

#Specifies the plug-in download URL. Leave blank to use the default URL.

PLUGIN_MODULE_ID_1=ark

PLUGIN_VERSION_1=3.1.1.20070315

#Specifies a plug-in ID and version that you want to install as part of the
iManager installation. These values come from the plug-in manifest file. Only used
when PLUGIN_INSTALL_MODE equals NET or BOTH.

PLUGIN_MODULE_ID_2=eDirectoryBackupAndRestore

PLUGIN_VERSION_2=2.7.20070410

#Specifies a plug-in ID and version that you want to install as part of the
iManager installation.

LAUNCH_BROWSER=false

#Enables/disables launching the gettingstarted.html when the iManager installation
completes

```

Once you have the `installer.properties` file configured to your satisfaction, use it to direct the silent installation of iManager as part of your application installation routine.

Logging Debug Messages

The `com.novell.emframe.dev.D` (Debug) class provides a simple mechanism for plug-ins to log debug messages to the screen, a file, or both. Writing the log occurs asynchronously. The log methods simply queue the event and return immediately—the actual writing is done later by a separate thread. Three levels of logging are supported: *info*, *warning*, and *error*. Only messages with a priority level greater than or equal to the current level are logged. Those with a lower priority are discarded.

The following table describes the properties used to control logging.

Table 9-1 *com/novell.emframe.dev.D members*

Property	Description
<code>void log(String)</code>	Overloaded. Logs a message.
<code>String getLogFileName()</code>	Returns the name of the log file
<code>boolean isDebugEnabled()</code>	Returns true if logging is enabled for INFO level
<code>int getLoggingLevel()</code>	Returns the logging priority level
<code>boolean getLoggingToFile()</code>	Returns true if logging to file is enabled
<code>boolean getLoggingToErr()</code>	Returns true if logging to <code>System.err</code> is enabled
<code>boolean getLoggingToOut()</code>	Returns true if logging to <code>System.out</code> is enabled
<code>void setLoggingLevel(int)</code>	Sets the logging priority level
<code>void setLoggingToFile(boolean)</code>	Enables/disables logging to file
<code>void setLoggingToErr(boolean)</code>	Enables/disables logging to <code>System.err</code>
<code>void setLoggingToOut(boolean)</code>	Enables/disables logging to <code>System.out</code>

If messages are logged to a file, they are logged to a standard text file `<iManager Home>/bin/iManager.log` as well as to an HTML file (`webapps/nps/WEB-INF/logs/debug.html`). The HTML log is formatted as a single HTML `<table>` element with each log message formatted as a single row in the table. The table remains unclosed (no closing `</table>` tag) so that additional log messages can be added. Although the table and document remain unclosed, the file can still be view with any web browser or text editor.

To view the HTML log from iManager, browse to the iManager Server > Configure iManager, and select the Logging Events tab.

Customizing iManager for Schema Extensions

10

If you are using iManager to manage a directory whose schema has been extended or you have created a plug-in that extends the schema, there are a number of things you can do to improve the way iManager works with the new objects and attributes. These things are covered in the following sections:

- ♦ [Section 10.1, “Providing an Image for a New Object Class,” on page 103](#)
- ♦ [Section 10.2, “Providing Translated Names for New Object Classes and Attributes,” on page 103](#)
- ♦ [Section 10.3, “Providing a Creator for a New Object Class,” on page 104](#)
- ♦ [Section 10.4, “Handling Deletion, Moving, and Renaming of a New Object Class,” on page 104](#)
- ♦ [Section 10.5, “Providing Pages for the Modify Object Property Book,” on page 104](#)
- ♦ [Section 10.6, “Providing Tasks to Interact with a New Object Class,” on page 104](#)

10.1 Providing an Image for a New Object Class

To provide an image that shows up in the object selector and object view, place a GIF image file in the `portal/modules/dev/images/dir` directory. The image should have the same name as the object class with all non-alphanumeric characters converted to underscores. For example, the GIF for the NDPS:Server Domain would be `NDPS_Server_Domain.gif`.

10.2 Providing Translated Names for New Object Classes and Attributes

- 1 Create a Java resource bundle with the translations of the object class and attributes. Prefix object class keys with `ObjectType` and attributes with `Attribute`. Replace non-alphanumeric characters in the keys to underscores. For example:

```
ObjectType.AFP_Server=AFP Server
ObjectType.Alias=Alias
ObjectType.NLS_Product_Container=License Product Container
ObjectType.NLS_License_Certificate=License Certificate
ObjectType.NLS_License_Server=License Service Provider
Attribute.L=Location
Attribute.CN=Common name
Attribute.Full_Name=Full name
Attribute.Surname=Surname
```

- 2 Copy the resource bundle to `SDK_HOME/tomcat/webapps/nps/WEB-INF/classes` or include the resource bundle in a JAR file and copy the JAR file to `SDK_HOME/tomcat/webapps/nps/WEB-INF/lib`.
- 3 Create a plug-in XML file in the `portal/modules/modulename/plugins` directory and add a `<dir-translator>` section. The following example shows the `<dir-translator>` section that registers several object classes and attributes:

```

<dir-translator>
  <resource-properties-file>
    com.novell.emframe.fw.FwResources
  </resource-properties-file>
  <object-type-name>AFP Server</object-type-name>
  <object-type-name>Alias</object-type-name>
  <object-type-name>NLS:Product Container</object-type-name>
  <object-type-name>NLS:License Certificate</object-type-name>
  <object-type-name>NLS:License Server</object-type-name>
  <attribute-name>L</attribute-name>
  <attribute-name>CN</attribute-name>
  <attribute-name>Full Name</attribute-name>
  <attribute-name>Surname</attribute-name>
</dir-translator>

```

10.3 Providing a Creator for a New Object Class

A user who is assigned to the eDirectory™ Management role has access to the Create Object task. The Create Object task only creates objects of types that are registered with iManager for the creator task. If you want to enable creation of a new object class using the Create Object task, you can either use the generic creator plug-in provided by iManager or create your own creator plug-in. For more information, see [Section 3.9, “Extending the Object Management Tasks,” on page 43](#).

10.4 Handling Deletion, Moving, and Renaming of a New Object Class

A user who is assigned to the eDirectory Management role has access to the Delete Object, Move Object, and Rename Object tasks. These tasks work on nearly all object classes, and they operate on each object in the same way. If you want to handle the delete, move, or rename operation in a different way for your new object class, you need to extend the corresponding generic deleter, move, or rename plug-in provided by iManager as explained in [Section 3.9, “Extending the Object Management Tasks,” on page 43](#).

10.5 Providing Pages for the Modify Object Property Book

A user who is assigned to the eDirectory Management role has access to the Modify Object task, which displays a property book that is used to modify attributes of an object. You can provide custom property book pages for modifying the attributes of your new object class. To do this, you need to create Java code for your page, create a JSP for the user interface of the page, and create a plugin XML file that describes the property book page. For more information, see [Chapter 4, “Creating Property Books and Pages,” on page 49](#).

10.6 Providing Tasks to Interact with a New Object Class

If you have extended the schema with a new object class and attributes, you probably need to create tasks to manage the object. For example, if you have created an object class that represents a vehicle in a company's fleet, you might create a task for assigning the vehicle object to a user. Another task might allow users to record the vehicle mileage. Another task might allow fleet management department employees to record maintenance information.

For information about creating a task, see [Chapter 3, “Creating Tasks,”](#) on page 23.

Creating an iPrint Gateway Plug-In

11

This section discusses creating an iManager plug-in to manage an iPrint gateway, and includes the following topics:

- ♦ [Section 11.1, “Introduction to iPrint,” on page 107](#)
- ♦ [Section 11.2, “Getting Started,” on page 108](#)
- ♦ [Section 11.3, “Using the Sample Gateway Plug-In,” on page 109](#)
- ♦ [Section 11.4, “Creating Gateway Plug-In Java Class Files,” on page 109](#)
- ♦ [Section 11.5, “Creating Gateway Plug-In JSP Files,” on page 110](#)
- ♦ [Section 11.6, “Creating Gateway Plug-In Registration Files,” on page 110](#)

11.1 Introduction to iPrint

iPrint is an innovative NetWare® feature that gives users access to remote printers over the Internet using the Internet Printing Protocol (IPP). When iPrint is installed, printer and other related objects are created in eDirectory™. These objects are managed using iManager and the iPrint module, which also ship with NetWare.

The iPrint module defines an iPrint Management role, which has several associated tasks, including Create Printer. Administrators use the Create Printer task to set up printers on the network. To complete this task, administrators must select a gateway type from a list of available gateways. Novell® provides the Novell LPR gateway, which always appears in this list.

Figure 11-1 The Create Printer task.

The screenshot shows the Novell iManager interface. The top bar includes the Novell logo, 'ADMIN NW65_SP7_44_TREE', and a 'View Example' link. The left-hand pane is titled 'Roles and Tasks' and lists various categories: eDirectory Encryption, eDirectory Maintenance, File Protocols, Groups, Help Desk, Identity Manager, Identity Manager Utilities, iPrint (Linux), Create Driver Store, **Create Printer**, Create Print Manager, Delete Print Object, and iPrint Client Management. The main area is titled 'Create Printer' and contains the following fields:

- Printer name: *
- Container name: * (value: lab)
- Print Manager name: *
- DNS name or IP address: *
- Location:
- Description:
- LPR Printer name: (value: PASSTHROUGH)
- RAW Port number:

At the bottom of the main area are 'Next >>' and 'Cancel' buttons. A status bar at the very bottom is empty.

Gateways allow iPrint clients to send jobs to printers that are not IPP-aware. Gateways translate iPrint queries or commands to printer-specific language that the physical printer can use. This is possible because gateways are configured to know the specific type (make and model) of printer being used. Third-party gateways are developed by printer manufacturers to support printers attached to the network. These gateways are developed to interact with specific proprietary printers and can provide a wider array of information and offer options that are not available for the generic Novell gateway.

This section explains how to create a task to provide for the configuration of third-party gateways. You should be familiar with the concepts discussed elsewhere in this document concerning the creation of iManager tasks before you proceed.

Gateway configuration plug-ins are not installed or registered with iManager as tasks are—they “plug in” to the iPrint task. This is accomplished by creating plug-in registration files that use the `<gateway-config>` tag and placing the files in special directories that are recognized by the iPrint plug-in. Except for these differences, creating a gateway configuration plug-in is just like creating an iManager task.

The following table lists the types of files used in gateway configuration plug-ins:

Table 11-1 *Gateway configuration plug-in files*

File type	Location
Java class files	tomcat\webapps\nps\WEB-INF\lib\jarfilename.jar
JSP files	tomcat\webapps\nps\portal\modules\iPrint\skins\default\devices\default\gatecfg
Registration files	tomcat\webapps\nps\portal\modules\iPrint\plugins\gatecfg
Resource files	tomcat\webapps\nps\WEB-INF\lib\jarfilename.jar

The gateway configuration plug-in file locations are the same as file locations for iManager tasks with the exception that the JSP and registration files are in subdirectories named “gatecfg” within the iPrint module directory hierarchy.

When you create a gateway configuration plug-in and copy the files to the appropriate directories, the name of the gateway appears in the Gateway Type drop-down list of gateways in the Create Printer task. Also, when the administrator selects the gateway and clicks the Next button, the plug-in provides a user interface that queries for configuration information required by the gateway. This user interface can gather information on a single screen or on several successive screens, if necessary. Your plug-in then writes that information to eDirectory and completes other processing, if needed.

11.2 Getting Started

The iManager 2.7 SDK contains the latest iPrint source and runtime components that are part of iManager, along with a sample third-party gateway plug-in.

Follow these steps to prepare for creating iPrint gateway plug-ins:

1. If you haven't done so already, read [Chapter 1, “Getting Started,” on page 11](#) for instructions on how to get started using the iManager 2.7 SDK.

2. Start iManager and log in.
3. If necessary, install the iPrint plug-in and assign your user object to the iPrint Management role.

11.3 Using the Sample Gateway Plug-In

The SDK contains a sample gateway plug-in named CompanyXYZ. We recommend that you use this sample as a template when you begin to write your own gateway plug-in. It is found in the iPrintGateway directory under the SDK home directory. The plug-in consists of the following files:

- ♦ GatewayTemplate.jsp. This is the JSP file that presents the user interface for gathering gateway configuration information.
- ♦ GatewayTemplate.java. This file contains the plug-in Java code that handles the logic and attribute management of the gateway.
- ♦ CompanyXYZ.xml. This is the plug-in registration file.
- ♦ CompanyXYZResources.properties. This resource file contains strings used in the plug-in template file.

Install the CompanyXYZ gateway plug-in by running

`SDK_HOME\iPrintGateway\makebat.bat`. This batch file compiles GatewayTemplate.java, stores the resulting GatewayTemplate.class file in WEB-INF\lib\CompanyXYZ.jar, and copies the remaining plug-in files to the appropriate directories under the nps servlet document root directory.

To test the plug-in:

- 1 Start iManager and log in.
- 2 Click iPrint Management > Create Printer.
- 3 Fill in the Printer name, Container name, and Manager name fields.
- 4 In the Gateway type field, select CompanyXYZ.
- 5 Click Next.
- 6 Type a URL in the Printer URL field.
- 7 Click Next.

11.4 Creating Gateway Plug-In Java Class Files

Java classes for gateway plug-ins retrieve the parameters that were set in the user interface, as displayed by the JSP file, then they save these parameters in eDirectory as printer agent object attribute values. The GatewayTemplate.java file included with the CompanyXYZ plug-in provides an excellent example of how to do this, and we recommend that you use this file as a starting point for creating your own gateway plug-ins. This file illustrates the following points concerning gateway plug-ins:

- ♦ They must extend the `com.novell.emframe.dev.Task` class and implement the `execute` method defined in that class.
- ♦ They must import the `com.novell.service.ndps` package, as shown in the following example:

```
import com.novell.service.ndps.*;
```

- They should contain the following code, which retrieves the printer agent object that was stored in the session:

```
Hashtable sessionCache = context.getSessionCache();
PrinterAgent newPa = (PrinterAgent)sessionCache.get("NDPS.newPrinterAgent");
```

- The `PrinterAgent.modifyAttrs` method is used to set the attributes in the Managed Object Database (MOD):

```
BasicAttributes attrSet;
attrSet = new BasicAttributes();
attrSet.put(Oid.NDPS_ATT_PDS_EXEC_AND_PARAMS.getString(),
            "NDPSGW IPP URL=" + printerURL);
newPa.modifyAttrs(DirContext.REPLACE_ATTRIBUTE, attrSet);
```

For further information about plug-in Java classes, see [Section 3.2, “Creating a Task: An Example,” on page 23](#) and [Section 3.3, “Tips for Creating Task Java Classes,” on page 30](#).

11.5 Creating Gateway Plug-In JSP Files

The process for creating gateway plug-in JSP files is identical to the process used to create JSP files for iManager tasks. This process is described in [Section 3.2.2, “Creating the UI,” on page 26](#) and [Section 3.4, “Tips for Creating JSPs,” on page 32](#).

11.6 Creating Gateway Plug-In Registration Files

When you execute the Create Printer task, the Gateway type drop-down list contains a dynamic list of gateway types. This list is created from the registration files that reside in the directory `WEB-INF\plugins\ndps\gatecfg`. To make your gateway plug-in available in this list you need to create a registration file for your plug-in and copy it to this directory.

Registration files for gateway plug-ins are similar to registration files for regular iManager plug-ins. They consist of XML tags that define the plug-in. The following example shows the `CompanyXYZ.xml` file that is included with the iManager 2.7 SDK:

```
<plugins>
  <gateway-config>
    <id>iPrint.gatecfg.CompanyXYZ</id>
    <version>1.0</version>
    <required-version>2.7.0</required-version>
    <register-gadget>true</register-gadget>
    <display-name-key>GateConfigNameKey</display-name-key>
    <resource-properties-file>CompanyXYZResources</resource-properties-file>
    <class-name>com.novell.imatege.iPrint.plugins.GatewayTemplate</class-name>
    <merge-template>iPrint.gatecfg.GatewayTemplate</merge-template>
  </gateway-config>
</plugins>
```

The outermost tag, `<plugins>`, is identical to that used for iManager plug-ins. The main difference is that registration files for gateway plug-ins use the `<gateway-config>` tag to define a gateway rather than `<role>`, `<task>`, or `<book>`. However, the remaining tags are identical to those used to define tasks, and they serve the same purposes. For detailed information about these tags, see [Section 3.5, “Creating Registration Files,” on page 35](#).

This section describes the template functions and their syntax, parameters, and return values, and includes the following reference information:

- ♦ [Section 12.1, “iManager API Documentation,” on page 111](#)
- ♦ [Section 12.2, “XML Schema for Installation and Registration Files,” on page 111](#)
- ♦ [Section 12.3, “Role-Based Services Directory Objects,” on page 111](#)
- ♦ [Section 12.4, “eDirectory Access Service XML Formats for eDirectory Attribute Syntax Definitions,” on page 113](#)

12.1 iManager API Documentation

Documentation for the iManager API is available as Javadoc-generated documentation:

- ♦ [iManager API \(../api/index.html\)](#)

12.2 XML Schema for Installation and Registration Files

The iManager installation and registration files are formatted in XML and must conform to the [iManager XML Schema \(../schema/iMgrXMLSchema.html\)](#).

12.3 Role-Based Services Directory Objects

This section describes the following RBS objects:

- ♦ [Section 12.3.1, “rbsCollection,” on page 111](#)
- ♦ [Section 12.3.2, “rbsRole,” on page 112](#)
- ♦ [Section 12.3.3, “rbsModule,” on page 112](#)
- ♦ [Section 12.3.4, “rbsTask,” on page 112](#)
- ♦ [Section 12.3.5, “rbsBook,” on page 112](#)
- ♦ [Section 12.3.6, “rbsScope,” on page 113](#)

12.3.1 rbsCollection

rbsCollection objects are the topmost containers for all RBS objects. A tree can have any number of rbsCollection objects. These objects have “owners,” which are users who have management rights over the collection.

Containment:

- ♦ Country
- ♦ domain
- ♦ Locality

- ♦ Organization
- ♦ Organizational Unit

12.3.2 rbsRole

rbsRole objects are container objects that represent a role in an organization. Role “members” can be Users, Groups, Organizations, or Organizational Units, and they are associated to a role in a specific scope of the tree. The rbsTask and rbsBook objects are assigned to rbsRole objects.

Containment:

- ♦ rbsCollection

12.3.3 rbsModule

rbsModule objects are container objects that hold rbsTask and rbsBook objects. They have a module name attribute that should represent the name of the product that defines the tasks or books.

Containment:

- ♦ rbsCollection

12.3.4 rbsTask

rbsTask objects are leaf objects that describe the behavior of a task. They have the following characteristics:

1. An entry point for invoking the task
2. A parameters string for miscellaneous data needed to perform the task
3. A list of rights that are assigned to perform the task
4. A back link to all roles to which the task is assigned

Containment:

- ♦ rbsModule

12.3.5 rbsBook

rbsBook objects are leaf objects that describe a book. They have the following characteristics:

1. An entry point for launching the book
2. A parameters string for miscellaneous data needed to display the book
3. A list of page attributes that are assigned rights for the book
4. A back link to all roles to which the book is assigned
5. A list of pages assigned to the book
6. A list of object class types that the book supports

Containment:

- ♦ rbsModule

12.3.6 rbsScope

rbsScope objects are leaf objects used for ACL assignments (instead of making assignments for each User object). They inherit from the Group class. User objects are assigned to an rbsScope object. These objects have a reference to the scope of the tree that they are associated with. rbsScope objects are dynamic, meaning that they are created, modified, and deleted on the fly. Do not modify or delete these objects manually.

Containment:

- ♦ rbsRole

12.4 eDirectory Access Service XML Formats for eDirectory Attribute Syntax Definitions

The eDirectory Access Service (eDAS) reads and writes attribute values as strings in XML format. This section lists the formats used for standard eDirectory syntax definitions. The first column lists the eDirectory syntax definitions and the second column lists the corresponding eDAS XML format.

SYN_BACK_LINK	<pre><attribute_name> <value> <remote-id>{Long}</remote-id> <object-name>{String}</object-name> </value> </attribute_name></pre>
SYN_BOOLEAN	<pre><attribute_name> <value>{true false}</value> </attribute_name></pre>
SYN_CE_STRING	<pre><attribute_name> <value>{String}</value> . . . </attribute_name></pre>
SYN_CI_LIST	<pre><attribute_name> <value> <item>{String}</item> . . . </value> </attribute_name></pre>
SYN_CI_STRING	<pre><attribute_name> <value>{String}</value> . . . </attribute_name></pre>

SYN_CLASS_NAME	<attribute_name> <value>{String}</value> . . . </attribute_name>
SYN_COUNTER	<attribute_name> <value>{String}</value> . . . </attribute_name>
SYN_DIST_NAME	<attribute_name> <value>{String}</value> . . . </attribute_name>
SYN_EMAIL_ADDRESSES	<attribute_name> <value> <type>{Long}</type> <address>{String}</address> </value> . . . </attribute_name>
SYN_FAX_NUMBER	<attribute_name> <value> <bits>{Long}</bits> <telephone-number>{String}</telephone-number> </value> . . . </attribute_name>
SYN_HOLD	<attribute_name> <value> <amount>{Int}</amount> <subject>{String}</subject> </value> . . . </attribute_name>
SYN_INTEGER	<attribute_name> <value>{String}</value> . . . </attribute_name>

SYN_INTERVAL	<attribute_name> <value>{String}</value> . . . </attribute_name>
SYN_NET_ADDRESS	<attribute_name> <value> <address-type>{Long}</address-type> <address>{String}</address> <address-string>{String}</address-string> </value> . . . </attribute_name>
SYN_NU_STRING	<attribute_name> <value>{String}</value> . . . </attribute_name>
SYN_OBJECT_ACL	<attribute_name> <value> <privileges>{Long}</privileges> <protected-attr-name>{String}</protected-attr-name> <subject>{String}</subject> </value> . . . </attribute_name>
SYN_OCTET_LIST	<attribute_name> <value> <item>{String}</item> . . . </value> </attribute_name>
SYN_OCTET_STRIN G	<attribute_name> <value>{String}</value> . . . </attribute_name>

SYN_PATH	<attribute_name> <value> <type>{Long}</type> <volume>{String}</volume> <path>{String}</path> </value> . . . </attribute_name>
SYN_PO_ADDRESS	<attribute_name> <value> <item>{String}</item> . . . </value> . . . </attribute_name>
SYN_PR_STRING	<attribute_name> <value>{String}</value> . . . </attribute_name>
SYN_REPLICA_POINTER	<attribute_name> <value> <replica-type>{Long}</replica-type> <replica-count>{Long}</replica-count> <replica-number>{Long}</replica-number> <server-name>{String}</server-name> <net-address-hint> <address-type>{Long}</address-type> <address>{String}</address> <address-string>{String}</address-string> </net-address-hint> </value> . . . </attribute_name>
SYN_STREAM	<attribute_name> <value>{String}</value> </attribute_name>
SYN_TEL_NUMBER	<attribute_name> <value>{String}</value> . . . </attribute_name>

SYN_TIME	<attribute_name> <value>{Time}</value> . . . </attribute_name>
SYN_TIMESTAMP	<attribute_name> <value> <seconds>{Int}</seconds> <replica>{Int}</replica> <event>{Int}</event> </value> . . . </attribute_name>
SYN_TYPED_NAME	<attribute_name> <value> <distinguished-name>{String}</distinguished-name> <level>{Long}</level> <interval>{Long}</interval> </value> . . . </attribute_name>
SYN_UNKNOWN	<attribute_name> <value> <attribute-name>{String}</attribute-name> <syntax-id>{Int}</syntax-id> <bytes>{String}</bytes> </value> . . . </attribute_name>

iManager Security Issues

A

The iManager SDK is a development tool, and not intended for use in production environments. However, as an iManager plug-in developer, you should be aware of iManager's security considerations when developing and testing iManager plug-ins to make sure you don't introduce any security vulnerabilities into your network environment.

This section provides information about potential security issues related to iManager, and includes information about the following topics:

- ♦ [Section A.1, "Secure LDAP Certificates," on page 119](#)
- ♦ [Section A.2, "Self-Signed Certificates," on page 120](#)
- ♦ [Section A.3, "iManager Authorized Users and Groups," on page 121](#)
- ♦ [Section A.4, "Preventing Username Discovery," on page 121](#)
- ♦ [Section A.5, "Tomcat Settings," on page 122](#)
- ♦ [Section A.6, "Encrypted Attributes," on page 122](#)
- ♦ [Section A.7, "Secure Connections," on page 122](#)

A.1 Secure LDAP Certificates

iManager can create secure LDAP connections behind the scenes without any user intervention. If the LDAP server's SSL certificate is updated for any reason (for example, new Organizational CA), iManager should automatically retrieve the new certificate using the authenticated connection and import it into its own keystore database.

If this does not happen correctly, you must delete the private key store that iManager uses, in order to force iManager and Tomcat to re-create the database and reacquire the certificate:

- 1 Shut down Tomcat.
- 2 Delete the `TOMCAT_HOME\webapps\nps\WEB-INF\iMKS` file.
- 3 Restart Tomcat.

Platform	Restart Command
NetWare® 6.5 or later	Enter <code>TC5STOP</code> . Wait at least one minute, then enter <code>TOMCAT5</code> to start the service again.
Windows*	Stop and start the Tomcat service.
Linux	Enter <code>/etc/init.d/novell-tomcat5 restart</code> .

- 4 Open iManager in a browser and log back in to the tree, to automatically reacquire the new certificate and re-create the database store.

Alternately, you can also manually import the required certificate into Tomcat's JVM default keystore using the `keytool` certificate management utility available in the JDK*. When creating secure SSL connections, iManager first tries the JVM default keystore, then uses the iManager specific keystore database.

After you have an eDirectory™ certificate saved in DER format, you must import the trusted root certificate into the iManager keystore. To do this, you need a JDK to use keytool. If a JRE was installed with iManager, you must download a JDK to use the keytool.

NOTE: For information about creating a .der certificate file, see [Exporting a Trusted Root or Public Key Certificate \(http://www.novell.com/documentation/crt32/crtadmin/data/a2ebopb.html\)](http://www.novell.com/documentation/crt32/crtadmin/data/a2ebopb.html) in the *Novell Certificate Server Admin Guide*. You will want to export the trusted root certificate.

1 Open a command window.

2 Change to the \bin directory where you have installed the JDK.

For example, on a Windows system, you would enter the following command:

```
cd j2sdk1.5.0_11\bin
```

3 Import the certificate into the keystore with the keytool, executing the following keytool commands (platform specific):

- ♦ NetWare

```
keytool -import -alias [alias_name] -file [full_path]\trustedrootcert.der  
-keystore sys:java\lib\security\cacerts
```

- ♦ Windows

```
keytool -import -alias [alias_name] -file [full_path]\trustedrootcert.der  
-keystore [full_path]\jre\lib\security\cacerts
```

- ♦ Linux

```
keytool -import -alias [alias_name] -file [full_path]/trustedrootcert.der  
-keystore [full_path]/jre/lib/security/cacerts
```

Replace *alias_name* with a unique name for this certificate and make sure you include the full path to trustedrootcert.der and cacerts.

The last path in the command specifies the keystore location. This varies from system to system because it is based on where iManager is installed. The default location for iManager on a Windows server, for example, is C:\Program Files\Novell\jre\lib\security\cacerts.

4 Enter `changeit` for the keystore password.

5 Click *Yes* to trust this certificate.

NOTE: This process must be repeated for each eDirectory tree you will be accessing with iManager. If LDAP has been configured to use a certificate not signed by the tree's Organizational CA, you must import that certificate's Trusted Root. This is necessary, for example, if LDAP is configured to use a VeriSign*-signed certificate.

A.2 Self-Signed Certificates

iManager includes a temporary, self-signed certificate that you use when installing iManager on any platform except NetWare. It has an expiration date of one year. For more information, see “[Self-Signed Certificates \(http://www.novell.com/documentation/imanager27/imanager_install_27/data/bu3uiv1.html\)](http://www.novell.com/documentation/imanager27/imanager_install_27/data/bu3uiv1.html)” in the *iManager 2.7 Installation Guide*.

A.3 iManager Authorized Users and Groups

Authorized Users and Groups are those that iManager permits to perform its various administrative tasks. For more information about specifying and configuring Authorized Users and Groups, see [Authorized Users and Groups \(http://www.novell.com/documentation/imanager27/imanager_admin_272/index.html?page=/documentation/imanager27/imanager_admin_272/data/b8qrh89.html\)](http://www.novell.com/documentation/imanager27/imanager_admin_272/index.html?page=/documentation/imanager27/imanager_admin_272/data/b8qrh89.html) in the *iManager 2.7.2 Administration Guide*.

Authorized User and Group data is stored in the `configiman.properties` file which must be secured to prevent unauthorized modification. To do this, modify the access controls for `configiman.properties` to restrict those users authorized to manually edit the file.

NOTE: Not specifying an Authorized User and Group, which prevents the `configiman.properties` file from being created, or specifying an Authorized User of `AllUsers`, allows any user to install iManager plug-ins and modify iManager server settings. This is a security risk for server-based iManager environments.

A.4 Preventing Username Discovery

In some installations, the eDirectory server is protected behind a firewall, but the iManager server is open to the outside world to allow management from home or on the road. Access to iManager is controlled with Username, Password, and Treename fields on the login screen. In such installations, it is often desirable to tighten security to avoid revealing any information about the system.

Standard iManager configurations pass through eDirectory messages related to invalid usernames and passwords during iManager authentication. These messages can inadvertently provide too much information to potential crackers. To avoid this, iManager 2.7 includes a configuration option to hide the specific reason for login failure. When enabled, the following error messages are replaced with a generic error message that reads: `Login Failure. Invalid Username or Password`.

- ♦ Invalid Username (-601)
- ♦ Incorrect password (-669)
- ♦ Expired password or disabled account (-220)

To enable this setting, open the *Configure* view and select *iManager Server > Configure iManager*. On the *Authentication* tab, select *Hide specific reason for login failure*. This sets `Authenticate.Form.HideLoginFailReason=true` in iManager's `config.xml` file.

Additionally, iManager 2.7 does not support the asterisk (*) character as a wildcard in the Username field. This prevents unauthorized users from discovering valid usernames. It also prevents possible denial-of-service attacks that attempt to overload the eDirectory server by continually attempting a login using only the wildcard (*), which forces eDirectory to search for and return all matching usernames.

A.5 Tomcat Settings

Because iManager makes use of Tomcat Servlet Container, iManager administrators should be aware of the encryption-related configuration options of those resources as part of their overall security strategy. Of particular interest are cipher suites and trusted certificates, which directly impact the quality of your wire-level encryption. Consider the following rules when configuring your Tomcat environment:

- ♦ Do not use SSL 2.0 cipher suites, which are outdated and not guaranteed to be secure.
- ♦ Do not use the NULL cipher suite in a production environment.
- ♦ Do not use any cipher suite classified as LOW or EXPORT quality, because these are less secure.
- ♦ Regularly review the list of trusted certificates, and limit the list of accepted Certificate Authorities to only those you are actually using

More information for Tomcat is available at the [Tomcat Documentation Web site \(http://tomcat.apache.org/tomcat-5.5-doc/index.html\)](http://tomcat.apache.org/tomcat-5.5-doc/index.html).

NOTE: Because of the way that iManager interprets and uses data, there are no known risks of HTML-based attacks such as cross-site scripting.

A.6 Encrypted Attributes

iManager is able to securely read eDirectory 8.8 encrypted attributes. However, because of the way it determines if an attribute is encrypted, iManager does not securely modify or delete these encrypted attributes. The impact of this, which can result in some wire-level data exposure, can be mitigated through normal network security practices such as the following:

- ♦ Locating all iManager servers behind the firewall
- ♦ Locating iManager servers physically near their associated eDirectory servers
- ♦ Physically securing iManager and eDirectory servers
- ♦ Requiring remote administrators to use a VPN to access iManager and eDirectory servers

A.7 Secure Connections

Although iManager leverages secure HTTP (SSL) for client communications, and secure LDAP connections between iManager and eDirectory servers, iManager does not, with the exception of reading encrypted attributes, utilize secure NCP connections for communications between iManager servers and eDirectory servers.

This is also true for the NCP connection used by Mobile iManager. The impact of this, which can result in some wire-level data exposure, can be mitigated through normal network security practices such as the following:

- ♦ Locating all iManager servers behind the firewall
- ♦ Locating iManager servers physically near their associated eDirectory servers
- ♦ Physically securing iManager and eDirectory servers
- ♦ Requiring remote administrators to use a VPN to access iManager and eDirectory servers

NOTE: Regardless of the wire-level encryption being used, passwords are always encrypted and protected as part of the iManager authentication process.

Revision History

B

This section outlines all the changes that have been made to the iManager 2.7 Developer Kit documentation (in reverse chronological order).

Date	Description
January 13, 2009	Made minor edits and technical updates. Updated the content in Section 1.1 to be consistent with iManager 2.7.2. Updated the content in Section 3.3 to be consistent with iManager 2.7.2. Updated the content in Section 6.3.3 to be consistent with iManager 2.7.2. Updated the content in Section 7.1.1 to be consistent with iManager 2.7.2. Added Section 7.3, "Plug-In Information Through XML," on page 85 to be consistent with iManager 2.7.1. Updated the content in Section 7.5.1 to be consistent with iManager 2.7.2. Updated the content in Appendix A.3 to be consistent with iManager 2.7.2.
October 17, 2007	Updated Chapter 3 content to be consistent with iManager 2.7. Removed Section 3.6, "Specifying Supported Device Types". Updated Chapter 4 content to be consistent with iManager 2.7. Updated Chapter 5 content to be consistent with iManager 2.7. Updated Chapter 6 content to be consistent with iManager 2.7. Removed Section 6.1.13, "A JSP Conduit Example: Delete User". Removed Section 6.2.2, "Include Files". Removed Section 6.2.3, "Parameter Variables". Updated Chapter 7 content to be consistent with iManager 2.7. Added Section 7.6, "Precompiling JSPs for Tomcat 5," on page 90 . Removed Chapter 11, "Migrating iManager 1.5 Plug-Ins to Version 2.x". Removed Chapter 12, "Migrating XSL to JSP Tasks". Minor edits and technical updates throughout book.
October 11, 2006	Updated Chapter 9, "Logging Debug Messages," on page 101 .
March 1, 2006	Made minor technical edits.
October 5, 2005	Transitioned to revised Novell documentation standards.
July 15, 2005	Updated Section 6.1.3, "Advanced Selection XML Syntax," on page 65 .

Date	Description
June 1, 2005	<p>Made minor technical edits.</p> <p>Added Section 5.5.1, "Dynamically Updating Drop-down Lists," on page 58.</p> <p>Removed outdated task development information, concerning mandatory methods, from Chapter 11, "Migrating iManager 1.5 Plug-Ins to Version 2.x".</p>
March 21, 2005	<p>Added Chapter 8, "Installing iManager with Your Application," on page 97.</p> <p>Added Section 7.5, "Installing Plug-ins to an Existing iManager," on page 88</p>
March 2, 2005	Made minor technical edits.
October 6, 2004	<p>Added Section 3.10, "Enabling a Task for the Object View," on page 45.</p> <p>Renamed and added more detail and examples to Section 3.9, "Extending the Object Management Tasks," on page 43.</p> <p>Added Chapter 10, "Customizing iManager for Schema Extensions," on page 103.</p> <p>Added Chapter 11, "Migrating XSL Tasks to JSP Tasks".</p> <p>Updated Section 6.1, "The Object Selector Widget," on page 61.</p> <p>Added Section 6.2, "The Advanced Selection Widget," on page 69.</p> <p>Added Section 6.3, "The MVStringEditor Widget," on page 75.</p> <p>Added Section 3.3.1, "Encoding Data," on page 32 and Section 3.4.1, "Encoding Data Using Tag Libraries," on page 33.</p> <p>Added Chapter 7, "Creating a Plug-In," on page 81.</p> <p>Removed the NDS Namespace Reference section from Chapter 12, "Reference," on page 111. This is superseded by the AdminNamespace, and the API documentation is now integrated into the iManager Framework API documentation.</p>
June 29, 2004	<p>Added a warning about configuring iManager to a tree that hosts a previous version of iManager to "Setting Up Role-Based Services" on page 14.</p> <p>Made minor technical edits.</p>

Date	Description
June 9, 2004	<p>Updated Section 1.1, "Requirements," on page 11.</p> <p>Renamed and reorganized "Prerequisites" on page 12.</p> <p>Replaced references to the Gadget Runner with iManager or Tomcat.</p> <p>Renamed the Starting iManger section to Accessing iManager.</p> <p>Renamed the Setting Up iManager section to "Setting Up Role-Based Services" on page 14 and created a single procedure.</p> <p>Removed the Setting Up An IDE section in Getting Started.</p> <p>Updated and reorganized Section 2.1, "How iManager Works," on page 17.</p> <p>Improved Section 2.3, "Role-Based Services," on page 20 and Section 2.4, "Module ID," on page 21.</p> <p>Removed the Tasks section.</p> <p>Added the following: Chapter 3, "Creating Tasks," on page 23, Chapter 4, "Creating Property Books and Pages," on page 49, Chapter 5, "Using the Plug-In Studio," on page 53, Chapter 6, "Using the iManager Widgets," on page 61, Chapter 9, "Logging Debug Messages," on page 101, and Chapter 11, "Migrating iManager 1.5 Plug-Ins to Version 2.x".</p> <p>Added Section 3.8, "Task Chaining," on page 41.</p> <p>Added reference information to Section 3.7, "Launching Tasks and Delegating to Tasks," on page 38.</p>

Date	Description
February 18, 2004	<p>Updated and fixed typos in code sample in “A JSP Conduit Example: Delete User” on page 69.</p> <p>Updated Section 1.1, “Requirements,” on page 11. Changed the JDK, browser, and NICI requirements.</p> <p>Updated Section 1.2, “Installing the SDK,” on page 11.</p> <p>Updated Section 1.3, “Using the iManager SDK,” on page 12, and included instructions for using the SDK on Linux.</p> <p>Moved the Enabling Connections in Assigned and Collection Owner Modes section from Section 2.5, “Connection Modes,” on page 22.</p> <p>Added Section 3.6, “Specifying Supported Device Types,” on page 37.</p> <p>Added Section 3.6, “Specifying Conditions for Task Execution,” on page 37.</p> <p>Added “Using the Plug-In Studio” on page 53 and removed the outdated section on using the Create iManager Template task.</p> <p>Changed the name of the Converting an iManager 1.5 Plug-In to a Task Gadget section to Migrating iManager 1.5 Plug-Ins to Version 2.0.</p> <p>Reorganized Migrating iManager 1.5 Plug-Ins to Version 2.0 by removing the Using the Automated JSP Conversion Tool section, adding Migrating Tasks and Pages Created Using the Create iManager Task, adding Migrating Other Plug-Ins, and adding other needed information.</p> <p>Added Section 6.4, “The Date/Time Widget,” on page 79.</p> <p>Renamed the XML Installation and Registration File Element Definitions section to Section 12.2, “XML Schema for Installation and Registration Files,” on page 111. Removed the DTD and its element definitions and examples, and replaced them with a link to the iManager XML Schema document.</p>
October 8, 2003	<p>Updated Section 1.1, “Requirements,” on page 11 to exclude JDK 1.4.2 because of incompatibilities.</p> <p>Updated Section 1.1, “Requirements,” on page 11 with the requirement that NICI be installed when running the SDK on Windows.</p> <p>In “Prerequisites” on page 12, changed references to gadgetRunner.bat to startSDK.bat.</p> <p>Added the XML Installation and Registration File Element Definitions section to Chapter 12, “Reference,” on page 111.</p> <p>Improved the quality of some graphics.</p>
June 2003	<p>Added Section 3.11, “Using the AdminNamespace,” on page 48 and added a link to the NDS Namespace API documentation to Section 12.1, “iManager API Documentation,” on page 111.</p> <p>Added Section 3.7, “Launching Tasks and Delegating to Tasks,” on page 38.</p> <p>Renamed the Registering a Plug-In section to Creating Registration Files (page 35) and updated it.</p> <p>Added Section 2.5, “Connection Modes,” on page 22.</p>

Date	Description
April 10, 2003	<p>Updated Section 2.2, "The iManager Directory Structure," on page 19.</p> <p>Moved the "RBS Objects" section from Section 2.3, "Role-Based Services," on page 20 to Chapter 12, "Reference," on page 111, and renamed it "Role Based Services Directory Objects."</p> <p>Added Chapter 11, "Creating an iPrint Gateway Plug-In," on page 107.</p> <p>Updated "Using the iManager Widgets" on page 61.</p>
March 2003	Added to the NDK.