



Adobe

Overview

Adobe® LiveCycle™ Form Manager

July 2006

Version 7.2

© 2006 Adobe Systems Incorporated. All rights reserved.

Adobe® LiveCycle™ Form Manager 7.2 Overview for Microsoft® Windows®, Linux, and UNIX®
Edition 2.0, July 2006

If this guide is distributed with software that includes an end user agreement, this guide, as well as the software described in it, is furnished under license and may be used or copied only in accordance with the terms of such license. Except as permitted by any such license, no part of this guide may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, recording, or otherwise, without the prior written permission of Adobe Systems Incorporated. Please note that the content in this guide is protected under copyright law even if it is not distributed with software that includes an end user license agreement.

The content of this guide is furnished for informational use only, is subject to change without notice, and should not be construed as a commitment by Adobe Systems Incorporated. Adobe Systems Incorporated assumes no responsibility or liability for any errors or inaccuracies that may appear in the informational content contained in this guide.

Please remember that existing artwork or images that you may want to include in your project may be protected under copyright law. The unauthorized incorporation of such material into your new work could be a violation of the rights of the copyright owner. Please be sure to obtain any permission required from the copyright owner.

Any references to company names and company logos in sample material or in the sample forms included in this software are for demonstration purposes only and are not intended to refer to any actual organization.

Adobe, the Adobe logo, Acrobat, LiveCycle, and Reader are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States and/or other countries.

Java is a trademark or registered trademark of Sun Microsystems, Inc. in the United States and other countries.

JBoss is a registered trademark (in the United States) and a service mark, and the JBoss graphic is a trademark and service mark, of Marc Fleury. They are licensed exclusively to JBoss, Inc.

Linux is the registered trademark of Linus Torvalds in the U.S. and other countries.

Microsoft and Windows are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

UNIX is a registered trademark of The Open Group in the US and other countries.

All other trademarks are the property of their respective owners.

This product includes software developed by Brian M. Clapper: bmc@clapper.org.

This product contains either BISAFE and/or TIPEM software by RSA Data Security, Inc.

The TWAIN Toolkit is distributed as is. The developer and distributors of the TWAIN Toolkit expressly disclaim all implied, express or statutory warranties including, without limitation, the implied warranties of merchantability, noninfringement of third party rights and fitness for a particular purpose. Neither the developers nor the distributors will be liable for damages, whether direct, indirect, special, incidental, or consequential, as a result of the reproduction, modification, distribution or other use of the TWAIN Toolkit.

This product includes software developed by the Apache Software Foundation (<http://www.apache.org/>)

This Program was written with MacApp®: ©1985-1988 Apple Computer, Inc. APPLE COMPUTER, INC. MAKES NO WARRANTIES WHATSOEVER, EITHER EXPRESS OR IMPLIED, REGARDING THE PRODUCT, INCLUDING WARRANTIES WITH RESPECT TO ITS MERCHANTABILITY OR ITS FITNESS FOR ANY PARTICULAR PURPOSE. The MacApp software is proprietary to Apple Computer, Inc. and is licensed to Adobe for distribution only for use in combination with Adobe LiveCycle Form Manager 7.0.

Portions of this code are licensed from Apple Computer, Inc. under the terms of the Apple Public Source License, Version 2. The source code version of these portions and the license are available at <http://www.opensource.apple.com/apsl/>.

NOTICE REGARDING SABLOTRON

March 27, 2003

Portions of this product are based on Modifications created from the Original Code known as the "Sablotron XSLT Processor". The Sablotron XSLT Processor is subject to the Mozilla Public License Version 1.1 (the "License"). You may obtain a copy of the License at <http://www.mozilla.org/MPL/Software> distributed under the License is distributed on an "AS IS" basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the License for the specific language governing rights and limitations under the License.

The Original Code is the Sablotron XSLT Processor. The Initial Developer of the Original Code is Ginger Alliance Ltd. Portions created by Ginger Alliance are Copyright (C) 2000 Ginger Alliance Ltd. All Rights Reserved.

Pursuant to sections 3.2 and 3.6 of the License, the Modifications created by Adobe Systems Incorporated are available as Source Code. The Modifications may be downloaded via the Internet from: <http://partners.adobe.com/asn/tech/xml/sablotron/index.jsp>

The Original Code may be downloaded via the Internet from: <http://www.gingerall.org>

The contents of this file are subject to the Mozilla Public License Version 1.1 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at <http://www.mozilla.org/MPL/>

Software distributed under the License is distributed on an "AS IS" basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the License for the specific language governing rights and limitations under the License.

Portions copyright 1992, 1993 Simmule Turner and Rich Salz. All rights reserved.

This software is based in part on the work of the Independent JPEG Group

RSA Data Security, Inc. MD5 Message-Digest Algorithm.

Copyright (C) 1991-2, RSA Data Security, Inc. Created 1991. All rights reserved.

Portions based in part on the work of the FreeType team.

Adobe Systems Incorporated, 345 Park Avenue, San Jose, California 95110, USA.

Notice to U.S. Government End Users. The Software and Documentation are "Commercial Items," as that term is defined at 48 C.F.R. §2.101, consisting of "Commercial Computer Software" and "Commercial Computer Software Documentation," as such terms are used in 48 C.F.R. §12.212 or 48 C.F.R. §227.7202, as applicable. Consistent with 48 C.F.R. §12.212 or 48 C.F.R. §§227.7202-1 through 227.7202-4, as applicable, the Commercial Computer Software and Commercial Computer Software Documentation are being licensed to U.S. Government end users (a) only as Commercial Items and (b) with only those rights as are granted to all other end users pursuant to the terms and conditions herein. Unpublished-rights reserved under the copyright laws of the United States. Adobe Systems Incorporated, 345 Park Avenue, San Jose, CA 95110-2704, USA. For U.S. Government End Users, Adobe agrees to comply with all applicable equal opportunity laws including, if appropriate, the provisions of Executive Order 11246, as amended, Section 402 of the Vietnam Era Veterans Readjustment Assistance Act of 1974 (38 USC 4212), and Section 503 of the Rehabilitation Act of 1973, as amended, and the regulations at 41 CFR Parts 60-1 through 60-60, 60-250, and 60-741. The affirmative action clause and regulations contained in the preceding sentence shall be incorporated by reference.

Contents

Preface	5
What's in this guide?	5
Who should read this guide?	5
Related documentation	5
1 About LiveCycle Form Manager	6
How LiveCycle Form Manager works	6
2 LiveCycle Form Manager Integration	8
Integrating with other Adobe LiveCycle products	8
Process management	9
LiveCycle Designer	9
LiveCycle Forms	9
LiveCycle Reader Extensions	9
LiveCycle Workflow	9
Document control and security	9
LiveCycle Policy Server	9
LiveCycle Document Security	10
Glossary	11

Preface

Adobe® LiveCycle™ Form Manager is a web-based application that provides an integrated framework for storing, accessing, managing, and distributing forms to users and to other applications, such as Adobe LiveCycle Forms, within organizations.

What's in this guide?

This guide provides an overview of how LiveCycle Form Manager works, describes the components included in LiveCycle Form Manager, and describes how LiveCycle Form Manager integrates with other LiveCycle products.

Who should read this guide?

You should read this guide before using any of the product tools or reading the related documentation.

Related documentation

The resources in this table can help you learn about LiveCycle Form Manager.

For information about	See
The new features in this product release	<i>What's New</i>
How to install, configure, and deploy LiveCycle Form Manager	<i>Installing and Configuring LiveCycle for JBoss</i> <i>Installing and Configuring LiveCycle for WebSphere</i> <i>Installing and Configuring LiveCycle for WebLogic</i>
How to use Adobe Administrator to administer LiveCycle Form Manager	<i>LiveCycle Form Manager Administrator Help</i>
How to use LiveCycle Form Manager end-user components	<i>LiveCycle Form Manager Help</i>
How to use Adobe User Management	<i>Adobe User Management Help</i>
Changes to the product that occurred late in the development cycle	<i>LiveCycle Form Manager Readme</i>
Other services and products that integrate with LiveCycle Form Manager	www.adobe.com
Patch updates, technical notes, and additional information on this product version.	www.adobe.com/support/products/enterprise/index.html

1

About LiveCycle Form Manager

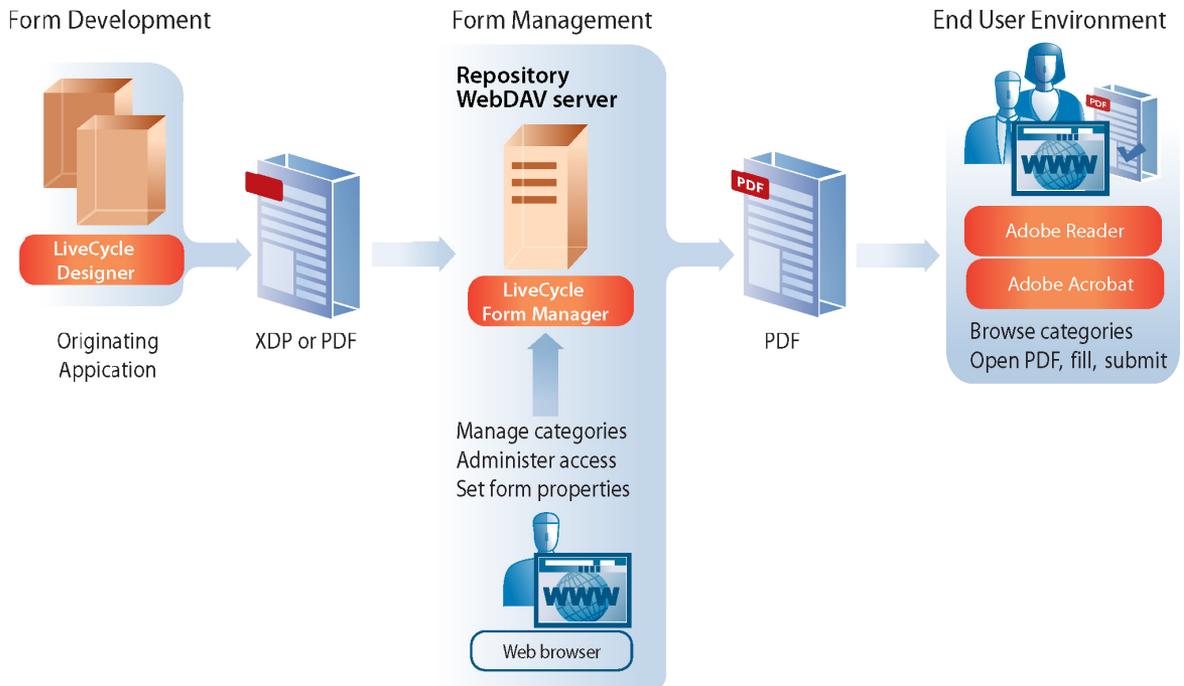
LiveCycle Form Manager is a form management tool that provides a central location for publishing forms and a secure, single point of access for end users. With LiveCycle Form Manager, administrators can better manage form publication, version control, and user access. Administrators can perform the following tasks:

- Publish forms to a central repository
- Organize forms and configure access to make the forms available to end users and to other applications in your organization
- Maintain version and access control

How LiveCycle Form Manager works

LiveCycle Form Manager is comprised of several components:

- The LiveCycle Form Manager repository includes a registry and a third-party database. LiveCycle Form Manager uses the repository to store and retrieve files.
- The Web-based Distributed Authoring and Versioning (WebDAV) server and URL handler are the mechanisms that enable form developers and other applications to store and access files in the LiveCycle Form Manager repository.
- Web applications provide form administrators and end users with access to forms.



LiveCycle Form Manager provides the following features:

Centralized repository for forms access

The repository is a registry and database that lets administrators and other users store and access forms in a centralized, integrated environment.

- Form authors who use Adobe LiveCycle Designer can publish forms directly from LiveCycle Designer to the repository.
- Form authors and other authorized users can use web folders to access the repository.
- The administrator uses the LiveCycle Form Manager File Manager page to access the repository for the purpose of organizing and categorizing forms for end users.
- Applications such as LiveCycle Forms, whose roles are defined by the application server, use URL handlers to access files in the repository.
- End users who save and archive forms using the LiveCycle Form Manager user application are saving those forms in the repository.

Web application for forms administration

LiveCycle Form Manager provides a web-based administration interface that enables administrators to organize and make forms available to end users. The administrator does several tasks to make forms available:

- Creates and links forms to categories to make the forms accessible to users of the LiveCycle Form Manager web application.
- Specifies form access rights for individual users and groups of users who access forms.
- Sets form display and processing options and usage rights for users and applications that access forms.

Web application for user access

LiveCycle Form Manager provides an easy-to-use web application interface for users who are using forms. The LiveCycle Form Manager graphical user interface (GUI) enables users to find, use, organize, and archive forms that they need to complete their work.

2

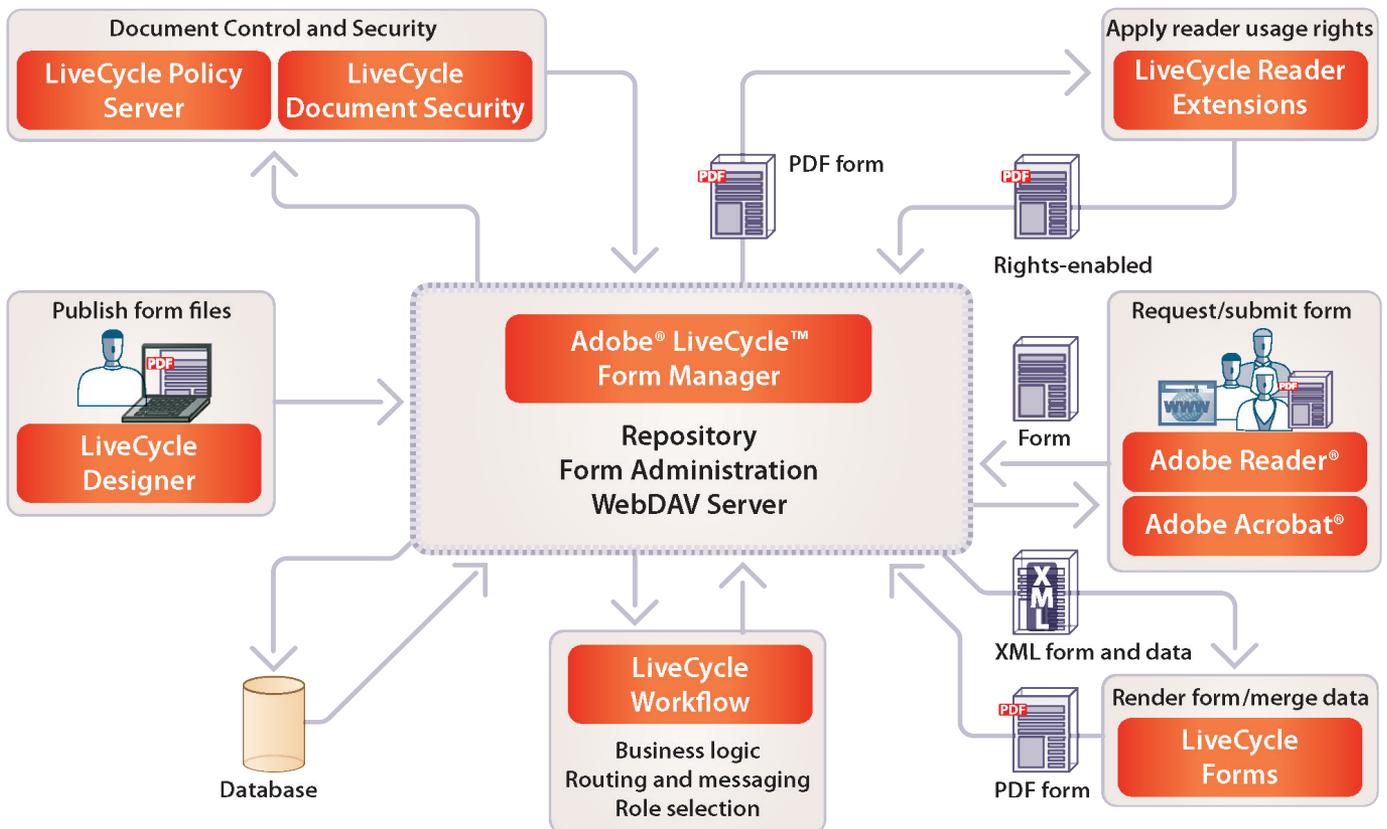
LiveCycle Form Manager Integration

LiveCycle Form Manager is part of the Adobe LiveCycle suite of products that can help your organization to efficiently and effectively deliver intelligent documents to end users and retrieve information from end users. LiveCycle products enable your organization to create and integrate intelligent documents (including the data) into your enterprise applications and business processes.

Integrating with other Adobe LiveCycle products

If you are installing and configuring LiveCycle Form Manager on a system that is currently running other Adobe products, to help ensure integration among LiveCycle products, see the *Installing and Configuring LiveCycle* guide for your application server.

LiveCycle Form Manager integrates with other Adobe document services and client software. Adobe document services are an integrated suite of products that enables organizations to create, exchange, process, organize, manage, and maintain security of documents and forms. You can use Adobe document services collectively in an integrated business process solution or selectively as an organization's business requirements dictate.



Process management

There are four LiveCycle products used for process management that integrate with LiveCycle Form Manager: LiveCycle Designer, LiveCycle Forms, Adobe LiveCycle Reader® Extensions, and Adobe LiveCycle Workflow.

LiveCycle Designer

Form authors use LiveCycle Designer to develop, create, and maintain form designs, define form business logic, and take advantage of built-in functions such as interactive objects, security settings, accessibility tools, and calculations and scripting functions. LiveCycle Form Manager provides a central repository where form authors can publish forms.

LiveCycle Forms

LiveCycle Forms automates the integration of form designs with data, and the delivery of HTML format to users and web browsers and Adobe PDF to users of Adobe Acrobat® Professional and Acrobat Standard or Adobe Reader. LiveCycle Forms renders XML forms into several different file formats, such as PDF and HTML files, on demand. LiveCycle Forms also retrieves form data from central repositories and merges it with the specified form.

LiveCycle Reader Extensions

LiveCycle Reader Extensions enables organizations to extend the functionality of Acrobat to users of Adobe Reader. Any end user can interact with multiple Adobe document services through Adobe Reader and forms, extending business processes beyond the enterprise.

LiveCycle Workflow

LiveCycle Workflow enables users to automate forms-based workflows so that forms are automatically routed from person to person. LiveCycle Workflow includes tools to design and implement the business logic necessary for determining the routing of forms. The LiveCycle Form Manager end-user web application integrates with LiveCycle Workflow, providing users with instant access to automated forms-based workflows.

Document control and security

There are two LiveCycle products used for document security that integrate with LiveCycle Form Manager: Adobe LiveCycle Document Security and Adobe LiveCycle Policy Server.

LiveCycle Policy Server

LiveCycle Policy Server is a web-based security system that enables users to dynamically apply confidentiality settings to their PDF documents, and maintain control over the documents, no matter how widely they are distributed. LiveCycle Policy Server prevents information from spreading beyond the users' reach by enabling them to maintain control over how recipients use the policy-protected PDF documents. Users can specify who can open a PDF document and how they can use it, and monitor the document after they distribute it. They can also dynamically control access to a policy-protected document, and even revoke access to the document if necessary.

LiveCycle Document Security

LiveCycle Document Security is a server-based product that enables your organization to protect the security and privacy of the PDF documents that it distributes and receives. This product uses digital signatures and encryption to ensure that documents cannot be read or altered by anyone other than the intended recipients, and are returned securely to their originator. Because security features are applied to the document itself, the document remains secure and controlled for its entire life cycle: beyond the firewall, when it is downloaded offline, and when it is submitted back to your organization.

Glossary

This glossary contains terminology definitions that are specific to documentation for the Adobe LiveCycle suite of products. These terms may have different meanings in other contexts, but they have restricted meanings in this documentation.

A

accessible forms

Forms that users with disabilities or vision impairment can view and fill using screen readers and other assistive technologies. See also *tagged Adobe PDF form*.

Acrobat form

A PDF document, created in Acrobat, that contains one or more form fields. The PDF document may also contain nonform content.

action

In a workflow, the representation of a step in a business process.

Adobe certified document

A document that is signed with a specific Adobe root certificate. An Adobe certified document provides a strong guarantee as to the authenticity and immutability of the document. See also *certificate*.

Adobe document services

Adobe document services extend the value of core enterprise systems to ensure more secure, reliable, and efficient use of business-critical information across the extended enterprise. Adobe document services include the Adobe LiveCycle suite of products and the Acrobat product line.

application

A set of generally interdependent files that make up a self-contained application that LiveCycle products can run. Applications may include files such as form designs, Java™ Server Pages, HTML pages, servlets, and images.

B

branch

A segment of a workflow that includes actions and routes. Under special circumstances, the branch configuration determines the behavior of the workflow segment that it contains.

C

certificate

An electronic file that establishes your identity, by binding your identity to your public key, when doing business or other transactions on the web. A *certificate* (or sometimes called a *digital certificate*) is issued by a certificate authority (CA). See also *Adobe certified document*.

client

The requesting program in a client/server relationship. A web browser is an example of a client application.

credential

The file that contains a private key. (The corresponding public key is contained in a certificate.) A private key is what one principal presents to another used to establish identity in decryption and signing operations. Credentials are issued by an authentication agent or a certification authority. See also *certificate*.

D

deadline

The time by which a person must complete a work item. Deadlines are properties of workflows.

dynamic form

A form that can expand or contract to reflect the amount of incoming data. See also *interactive form*.

E

encryption

The conversion of data into a format (called a ciphertext) that cannot be easily understood by unauthorized persons. The conversion is done using an encryption algorithm.

F

form

An electronic document that captures and delivers data. A person may add data to an interactive form, or a server process may merge a form design with data to produce a form.

form authors

LiveCycle Designer users who are capable of creating fillable forms to be used in Acrobat or Adobe Reader, and simple non-interactive forms for deployment to LiveCycle Forms. See also *form developers*.

FormCalc

A calculation language similar to that used in common spreadsheet software that facilitates form design without requiring a knowledge of traditional scripting techniques or languages.

form design

The design-time version of a form that an author or developer creates in LiveCycle Designer.

form developers

LiveCycle Designer users who are capable of creating complex form-based applications for use in different environments. See also *form author*.

form object

A form element, such as a button or text field, that you can place on a form. An object has its own set of properties and events.

I

interactive form

A form that a person can interact with and complete electronically.

N

non-interactive form

A form that a person can view or print but cannot fill electronically. Noninteractive forms can be merged or prepopulated with data, but the data cannot be changed by a user. Noninteractive forms are designed for output.

P

PDF document

Portable Document Format. A file conforming to the PDF specification as published by Adobe Systems or a file conforming to the XDP specification, containing exactly one PDF packet and no more than one each XFA-Template, XFA-Configuration, XFA-SourceSet, and Annotations packets.

PDF form

A form that users can access in Acrobat and Adobe Reader. PDF forms are either interactive or noninteractive.

permissions

Security settings applied, for example, to restrict users from opening, editing, printing, or removing encryption from a PDF file. Permissions cannot be changed unless the user has the Permissions password. Permissions can be set in LiveCycle Designer, Acrobat, LiveCycle Document Security, and other products.

policy

Defines a set of security permissions and users who can access a PDF document to which the policy is applied. Policies are created using LiveCycle Policy Server and can be applied to documents using LiveCycle Policy Server, LiveCycle Document Security, or Acrobat 7.0.

prepopulated form

A form that appears to the user with some or all fields automatically populated with data.

Q

QPAC

Quick Process Action Component. A JAR file that contains server-side code and client-side code for use with LiveCycle Workflow. In LiveCycle Workflow Designer, QPACs provide action components that can be added to workflows to represent a step in a process. LiveCycle Workflow Server interprets each action of the workflow and executes the server-side code of the corresponding QPACs. QPACs enable LiveCycle Workflow to interact with other LiveCycle products, such as LiveCycle Forms and Adobe LiveCycle Barcoded Forms.

R

reminder

A notification sent to people that reminds them to complete a work item. Reminders are properties of workflows.

render

An action whereby LiveCycle Forms merges a form design, possibly with data, to display a PDF or HTML form in a browser.

registry

A repository of shared information that provides services for the purpose of enabling business process integration between interested parties. See also *repository*.

repository

The underlying storage area within a registry. See also *registry*.

restricted document

A PDF document with password security restrictions (permissions) that prevent the document from being opened, printed, or edited.

rights-enabled document

A PDF document that includes security extensions that enable Adobe Reader users to fill forms, add comments, and sign documents.

route

The path between actions on a workflow. Routes determine the order in which LiveCycle Workflow Server executes actions at run time.

run time

For form rendering, the time when an application or server process retrieves a form design, possibly merges it with data, and presents it to a user for viewing or filling.

S

split

A segment in a workflow that contains one or more branches. The branches in a split are executed in parallel.

static form

A form that remains exactly as it was designed. The layout does not change according to the amount of incoming data.

subform

An object that can act as a container for form objects and other subforms. A subform helps to position form objects relative to each other and provide structure in dynamic form designs. A subform can also provide a reference point, when binding data to a form, by restricting the scope for a field so that it matches that of the corresponding data node.

T

tagged Adobe PDF form

Includes a logical structure and a set of defined relationships and dependencies among the various elements, plus additional information that permits reflow. See also *accessible forms*.

turnkey

An installation option that automatically installs and configures the LiveCycle product files, JBoss® application server, and MySQL database, and deploys the product files to JBoss. After you perform a turnkey installation, the LiveCycle product is ready to use.

U

usage rights

Rights that extend the functionality of Adobe Reader and enable users to save forms with data, add comments, and sign documents.

W

workflow

The electronic representation of a business process. Workflows are created using LiveCycle Workflow Designer.

X

XDP file

XML Data Package. LiveCycle Designer saves form designs as either XDP files or PDF files. LiveCycle Forms uses XDP files to render forms in PDF or HTML format.

XML Forms Architecture

Represents the underlying technology beneath the Adobe XML forms solution. It enables the construction of robust and flexible form-based applications for use on either the client or the server.

XML form

A PDF form that conforms to the Adobe PDF specification and the Adobe XML Forms Architecture. XML forms are typically created in LiveCycle Designer. XML forms can have the file name extension .xdp or .pdf.



Adobe® LiveCycle™ Form Manager

Version 7.2

- What's New
- Overview
- Installing and Configuring LiveCycle for JBoss
- Installing and Configuring LiveCycle for WebSphere
- Installing and Configuring LiveCycle for WebLogic
- LiveCycle Form Manager Help
- LiveCycle Form Manager Administration Help
- Adobe LiveCycle Form Manager Readme

July 2006

Installing and Configuring LiveCycle

The Installing and Configuring LiveCycle™ documentation provides information about installing, configuring, and deploying LiveCycle products. To ensure that customers have access to the most recent and updated information, the documentation is located at the following website:

www.adobe.com/support/documentation/en/livecycle/

Installing and Configuring guides

The Installing and Configuring LiveCycle documentation is provided in a set of six guides based on the LiveCycle product and application servers.

The *Installing and Configuring LiveCycle for JBoss*, *Installing and Configuring LiveCycle for WebLogic*, and *Installing and Configuring LiveCycle for WebSphere* guides describe how to install and configure the following LiveCycle products and deploy the products to the specific application server:

- Adobe® LiveCycle Assembler 7.2.1
- Adobe LiveCycle Forms 7.2
- Adobe LiveCycle Form Manager 7.2
- Adobe LiveCycle PDF Generator 7.2 Professional, LiveCycle PDF Generator 7.2 Elements, and LiveCycle PDF Generator 7.2 for PostScript®
- Adobe LiveCycle Print 7.2
- Adobe LiveCycle Workflow 7.2.1
- Watched Folder

The *Installing and Configuring LiveCycle Security Products for JBoss*, *Installing and Configuring LiveCycle Security Products for WebLogic*, and *Installing and Configuring LiveCycle Security Products for WebSphere* guides describe how to install and configure the following LiveCycle products and deploy the products to the specific application server:

- Adobe LiveCycle Document Security 7.2
- Adobe LiveCycle Policy Server 7.2
- Adobe LiveCycle Reader® Extensions 7.2

Updated product information

A Knowledge Center article has also been posted to communicate any updated LiveCycle product information. You can access the article from the following website:

www.adobe.com/support/products/enterprise/knowledgecenter/c4811.pdf



Developing User Management Service Providers

July 2006

Adobe® User Management

Version 1.23

© 2006 Adobe Systems Incorporated. All rights reserved.

Adobe® User Management 1.23 Developing User Management Service Providers for Microsoft® Windows®, Linux®, and UNIX®
Edition 1.2, July 2006

If this guide is distributed with software that includes an end user agreement, this guide, as well as the software described in it, is furnished under license and may be used or copied only in accordance with the terms of such license. Except as permitted by any such license, no part of this guide may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, recording, or otherwise, without the prior written permission of Adobe Systems Incorporated. Please note that the content in this guide is protected under copyright law even if it is not distributed with software that includes an end user license agreement.

The content of this guide is furnished for informational use only, is subject to change without notice, and should not be construed as a commitment by Adobe Systems Incorporated. Adobe Systems Incorporated assumes no responsibility or liability for any errors or inaccuracies that may appear in the informational content contained in this guide.

Please remember that existing artwork or images that you may want to include in your project may be protected under copyright law. The unauthorized incorporation of such material into your new work could be a violation of the rights of the copyright owner. Please be sure to obtain any permission required from the copyright owner.

Any references to company names in sample templates are for demonstration purposes only and are not intended to refer to any actual organization.

Adobe, the Adobe logo, and LiveCycle are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States and/or other countries.

IBM and WebSphere are trademarks of International Business Machines Corporation in the United States and/or other countries.

Linux is a registered trademark of Linus Torvalds.

UNIX is a registered trademark of The Open Group in the US and other countries.

Microsoft and Windows are either trademarks or registered trademarks of Microsoft Corporation in the United States and/or other countries.

All other trademarks are the property of their respective owners.

This product contains either BISAFE and/or TIPEM software by RSA Data Security, Inc.

This product includes software developed by the Apache Software Foundation (<http://www.apache.org/>).

Portions Copyright (C) 1991, 1999 Free Software Foundation, Inc. The JBOSS, OmniORB, JacORB, and SwarmCache libraries are licensed under the GNU Library General Public License, a copy of which is included with this software.

Adobe Systems Incorporated, 345 Park Avenue, San Jose, California 95110, USA.

Notice to U.S. Government End Users. The Software and Documentation are "Commercial Items," as that term is defined at 48 C.F.R. §2.101, consisting of "Commercial Computer Software" and "Commercial Computer Software Documentation," as such terms are used in 48 C.F.R. §12.212 or 48 C.F.R. §227.7202, as applicable. Consistent with 48 C.F.R. §12.212 or 48 C.F.R. §§227.7202-1 through 227.7202-4, as applicable, the Commercial Computer Software and Commercial Computer Software Documentation are being licensed to U.S. Government end users (a) only as Commercial Items and (b) with only those rights as are granted to all other end users pursuant to the terms and conditions herein. Unpublished-rights reserved under the copyright laws of the United States. Adobe Systems Incorporated, 345 Park Avenue, San Jose, CA 95110-2704, USA. For U.S. Government End Users, Adobe agrees to comply with all applicable equal opportunity laws including, if appropriate, the provisions of Executive Order 11246, as amended, Section 402 of the Vietnam Era Veterans Readjustment Assistance Act of 1974 (38 USC 4212), and Section 503 of the Rehabilitation Act of 1973, as amended, and the regulations at 41 CFR Parts 60-1 through 60-60, 60-250, and 60-741. The affirmative action clause and regulations contained in the preceding sentence shall be incorporated by reference.

Contents

Preface	5
What's in this guide?	5
Who should read this guide?	5
Related documentation	5
1 Introduction	6
Understanding the authentication process.....	6
Creating custom service providers	7
Including the User Management SPI JAR file	7
2 Creating Custom Authentication Providers	8
About custom authentication providers	8
Examining the authentication process	9
User Management SPI interfaces	10
Sample authentication provider	10
Retrieving authentication values.....	10
Retrieving configuration information.....	11
Performing the authentication operation.....	12
Sending authentication results to User Management.....	14
3 Creating Custom Directory Service Providers	15
Sample directory service provider	15
Directory service provider interfaces	16
Implementing the directory service provider interfaces	16
DirectoryPrincipalProvider interface.....	17
DirectoryUserProvider interface	21
DirectoryGroupProvider interface	21
Connecting to the directory	23
Getting the directory properties	24
GroupConfigBO interface	24
UserConfigBO interface	24
Testing the connection.....	24
Configuring User Management for custom directory service providers	25
4 Registering Custom Service Providers	26
User Management configuration settings	26
XML configuration file.....	27
XML element types in the configuration file.....	28
Defining domains for custom service providers	28
Configuring User Management to use custom authentication providers	29
Identifying authentication providers.....	29
Configuring domains for authentication providers	31
Configuring User Management to use directory service providers	33
5 Deploying Custom Service Providers	35
Packaging your custom service provider	35
Repackaging the LiveCycle EAR file	35
Deploying custom service providers	36
Index	38

List of Examples

Example 2.1	Retrieving authentication values	11
Example 2.2	Retrieving User Management configuration information	12
Example 2.3	Performing an authentication operation	13
Example 2.4	Returning authentication results in an AuthResponseImpl object	14
Example 3.1	Entry point for user services.....	17
Example 3.2	Representing the state information.....	17
Example 3.3	Determining the state	19
Example 3.4	Collecting records.....	19
Example 3.5	Setting the principal information	20
Example 3.6	Implementing the DirectoryUserProvider interface	21
Example 3.7	Group state information	21
Example 3.8	Retrieving group members.....	22
Example 4.1	XML configuration file.....	27
Example 4.2	Defining a domain.....	28
Example 4.3	Identifying an authentication provider	30
Example 4.4	Configuring a domain for an authentication provider	31
Example 4.5	Configuring a domain	32
Example 4.6	Configuring the domain to use custom directory service providers	33
Example 4.7	Configuring the custom group provider.....	34
Example 4.8	Configuring the custom user provider	34

Preface

This guide provides information about the use of Adobe® User Management SPI, which provides a means by which you can create custom service providers for User Management.

What's in this guide?

This document provides information about the programmatic interfaces and classes that are used to create custom service providers for User Management. In addition, this guide discusses how to register and deploy custom service providers. This guide is a companion guide to *User Management SPI Reference*.

Who should read this guide?

This guide is intended for Java 2 Enterprise Edition (J2EE) developers who are responsible for developing custom service providers for User Management.

Related documentation

In addition to this guide, the *User Management SPI Reference* describes the interfaces and classes that are located in the User Management SPI. You can learn more about other Adobe services and products at www.adobe.com and <http://partners.adobe.com/public/developer/main.html>.

1

Introduction

The Adobe User Management SPI is a Java API that enables you to create custom service providers for Adobe User Management. User Management enables administrators to maintain a database for all users and groups, synchronized with one or more third-party user directories. User Management provides authentication, authorization, and user management for LiveCycle products, including Adobe LiveCycle™ Workflow, Adobe LiveCycle Forms, and Adobe LiveCycle Form Manager. For more information about User Management, see *User Management Help*.

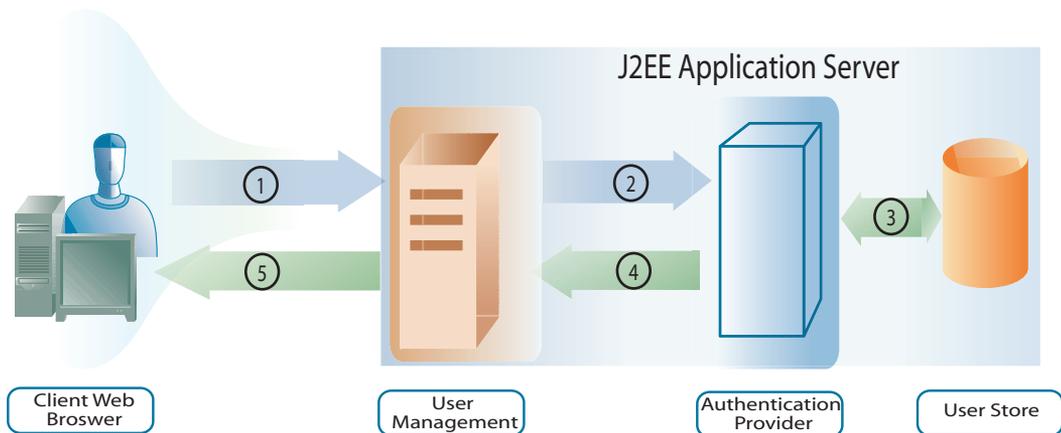
A User Management service provider consists of a custom authentication provider and a custom directory provider. A custom authentication provider authenticates users, and a directory provider stores user information. Using the User Management SPI, you can extend the User Management default functionality that is related to authenticating users and storing user information.

Understanding the authentication process

Authentication providers receive authentication requests from User Management and provide responses to it.

When User Management receives an authentication request (for example, a user attempts to log in), it passes user information to the authentication provider to authenticate. User Management receives the results from the authentication provider after it authenticates the user.

The following diagram shows the interaction among an end user attempting to log in, User Management, and an authentication provider.



The following table describes each step of the process and indicates the SPI members involved.

Step	Description
1	A user attempts to log into a LiveCycle product that invokes User Management. The user specifies a user name and password.
2	User Management sends the user name and password, as well as configuration information, to the authentication provider.

Step	Description
3	The authentication provider connects to the user store and authenticates the user.
4	The authentication provider returns the results to User Management.
5	User Management either lets the user log in or denies access to the product.

Creating custom service providers

You use the User Management SPI to create a custom service provider by performing the following tasks:

1. Create a custom authentication provider. For information, see [“Creating Custom Authentication Providers” on page 8](#).
2. Create a custom directory provider. For information, see [“Creating Custom Directory Service Providers” on page 15](#).
3. Configure the User Management XML configuration file to recognize the new service provider. For information, see [“Registering Custom Service Providers” on page 26](#).
4. Deploy the custom service provider. For information, see [“Deploying Custom Service Providers” on page 35](#).

Including the User Management SPI JAR file

User Management SPI consists of two Java packages:

- com.adobe.idp.um.spi.authentication
- com.adobe.idp.um.spi.directoryservices

You use classes and interfaces located within these packages to create custom service providers. For information about these packages and their contents, see the *API Reference*.

The User Management SPI is packaged in a JAR file named um-spi.jar. You must copy the JAR file into your application’s class path in order to use the User Management SPI in your Java project.

The um-spi.jar file is installed in the following directory when User Management is installed:

```
C:\Adobe\LiveCycle\components\um\<app_server>\lib\adobe
```

where C:\ is the drive on which User Management is installed, and app_server is the J2EE application on which User Management is deployed. For example, assume that User Management is deployed on JBoss. In this situation, the um-spi.jar file is in the following directory:

```
C:\Adobe\LiveCycle\components\um\jboss\lib\adobe
```

To access the interfaces and classes in the um-spi.jar file, add the following import statements to your Java project:

```
import com.adobe.idp.um.spi.authentication.*;  
import com.adobe.idp.um.spi.directoryservices.*;
```

2

Creating Custom Authentication Providers

This chapter explains how to use the User Management SPI to develop custom authentication providers that are based on user name and password authentication.

By default, User Management supports JAAS and LDAP authentication. However, by using the User Management SPI, you can create a custom authentication provider and then configure User Management to use the custom authentication provider to replace its default authentication provider. You can also configure User Management to use the custom authentication provider in addition to the default authentication provider.

A custom authentication provider is dependent on a custom directory service provider; therefore, you must create a custom directory service provider when you create a custom authentication provider. User information that is authenticated by a custom provider is placed in a data store that is accessed by a customer directory service provider. For information, see [“Creating Custom Directory Service Providers” on page 15](#).

This chapter provides the following information.

Topic	Description	See
About custom authentication providers	Explains how to use the User Management SPI to develop a custom authentication provider.	page 8
Retrieving authentication values	Explains how to retrieve user values that are passed from User Management to the custom authentication provider.	page 10
Retrieving configuration information	Explains how to retrieve User Management configuration information that is used in the authentication process.	page 11
Performing the authentication operation	Explains how to perform an authentication operation. This section also provides an example of an authentication operation by using a tab-delimited file.	page 12
Sending authentication results to User Management	Explains how to send authentication results to User Management.	page 14

About custom authentication providers

The User Management SPI provides a Java API for developing custom authentication providers. To develop a custom authentication provider, create a Java class that implements the `AuthProvider` interface that belongs to the `com.adobe.idp.um.spi.authentication` package. This class must contain a method named `authenticate`, which is called by User Management to authenticate users.

User Management passes user and configuration values to the `authenticate` method when a user attempts to log in. A user name and the corresponding password are passed within a `java.util.Map` object. Configuration information is passed within a `java.util.List` object. The following code fragment shows the method signature of the `authenticate` method:

```
public AuthResponse authenticate(Map credential, List authConfigs)
```

This method returns an `AuthResponse` object that specifies whether the user was authenticated. For information, see [“Sending authentication results to User Management” on page 14](#).

Examining the authentication process

This section examines in more detail the authentication process that was introduced earlier in this guide. For information, see [“Understanding the authentication process” on page 6](#).

During most authentication steps, User Management invokes methods of your custom authentication provider. For example, User Management invokes the `authenticate` method after a user attempts to log in. The following table explains the relationship between the authentication process and the User Management SPI, and specifies the SPI methods involved in each step of the authentication process.

Step	Description	SPI member used
1	A user attempts to log into a LiveCycle product that invokes User Management. The user specifies a user name and password.	No SPI methods are invoked.
2	User Management sends the user name and password, as well as configuration information, to the authentication provider.	User Management invokes the <code>authenticate</code> method that is located in an authentication provider. The <code>authenticate</code> method requires a <code>java.util.Map</code> object that contains user information and a <code>java.util.List</code> object that contains configuration information as arguments. For information, see “Retrieving authentication values” on page 10 . The configuration information contained in the <code>java.util.List</code> object is a collection of one or more <code>AuthConfigBO</code> objects. For information, see “Retrieving configuration information” on page 11 .
3	The authentication provider connects to the user store and authenticates the user.	You are required to develop Java code that performs the authentication. For example, you can authenticate a user by using a third-party API, such as the Java JDBC API, that interacts with a specified user store. This chapter authenticates a user by searching a tab-delimited file that contains user information. For information, see “Performing the authentication operation” on page 12 .
4	The authentication provider returns the results to User Management.	The authentication provider stores the results in an <code>AuthResponse</code> object. The <code>authenticate</code> method returns the <code>AuthResponse</code> object to User Management. For information, see “Sending authentication results to User Management” on page 14 .
5	User Management either lets the user log in or denies the user access to the product.	No SPI methods are invoked.

User Management SPI interfaces

The following table lists and describes the User Management SPI interfaces and classes that are used to create a custom authentication provider.

Interface	Description
<code>AuthProvider</code>	This interface is the primary interface that your custom authentication provider must implement.
<code>AuthConfigBO</code>	This interface defines a container for storing configuration information about an authentication provider.
<code>AuthResponse</code>	This interface is used to communicate authentication results to User Management. If the authentication provider successfully authenticates the user, it communicates the success to User Management along with the authenticated user name, user domain, and authentication type. Likewise, this interface is used to inform User Management that authentication was unsuccessful.
<code>AuthResponseImpl</code>	This class is used to create an object instance of the <code>AuthResponse</code> interface.

Sample authentication provider

User Management provides a sample implementation of the `AuthProvider` interface. The sample demonstrates how service providers interact with User Management. The sample Java code is provided in the file named `FBAuthenticationProviderImpl.java`. You can retrieve the User Management sample from the Adobe LiveCycle Developer Center website at http://partners.adobe.com/public/developer/livecycle/index_samples.html.

Tip: You may find it useful to print the sample code and refer to it as you read this chapter.

Retrieving authentication values

User Management passes user and configuration information to your authentication provider's `authenticate` method. User information is passed in by using a `java.util.Map` object, and configuration information is passed in by using a `java.util.List` object.

You retrieve the user information by invoking the `java.util.Map` object's `get` method. Pass the following keys to the `get` method to retrieve user information.

Key name	Description
<code>AuthProvider.USER_NAME</code>	The user name to authenticate.
<code>AuthProvider.PASSWORD</code>	The password that corresponds to the user name.
<code>AuthProvider.AUTH_TYPE</code>	The type of authentication to perform. The type must be <code>AUTHTYPE_USERNAME_PWD</code> . If no type is specified, the type <code>AUTHTYPE_USERNAME_PWD</code> is assumed. If a different type is specified, the authentication provider must respond with an indication that the parameters were not understood.

The following example retrieves authentication values from the `java.util.Map` object that is passed to the authentication provider. The `java.util.Map` object's `get` method is used to retrieve each value.

Example 2.1 Retrieving authentication values

```
public AuthResponse authenticate(Map credential, List authConfigs) {  
  
    //Declare string variables to store user information  
    String userName = null;  
    String password = null;  
    String authType = null;  
  
    // Retrieve the user values passed from User Management  
    if (credential.containsKey(AuthProvider.USER_NAME))  
        userName = (String) credential.get(AuthProvider.USER_NAME);  
    if (credential.containsKey(AuthProvider.PASSWORD))  
        password = (String) credential.get(AuthProvider.PASSWORD);  
    if (credential.containsKey(AuthProvider.AUTH_TYPE))  
        authType = (String) credential.get(AuthProvider.AUTH_TYPE);  
    //More logic
```

Note: The `java.util.Map` object may also include the keys named `AuthProvider.CONTEXT` and `AuthProvider.ENCODED_KERBEROS_TICKET`. Your authentication provider can ignore these keys.

Retrieving configuration information

A custom authentication provider can retrieve configuration information that is passed by User Management. This information represents domain information specified in the User Management configuration settings. For information about domain information, see [“Defining domains for custom service providers” on page 28](#).

User Management configuration settings can include required connection values for the data store as well as other information specific to your authentication provider. The domain information includes the name of the domain in which you are authenticating. You must retrieve the domain name for each authentication operation that is performed.

Multiple domains can exist for one authentication provider. It is the authentication provider's responsibility to try them all and select the one that succeeds. The domain name is specified in the response that is sent to User Management. For information, see [“Sending authentication results to User Management” on page 14](#).

The `java.util.List` object that was passed to the authentication provider's `authenticate` method contains an `AuthConfigBO` object for each domain that is configured for a specific authentication provider. An `AuthConfigBO` object is a container for domain configuration information that applies to a specific authentication provider.

Iterate through the `java.util.List` object to retrieve all `AuthConfigBO` objects by creating an `Iterator` object. To create an `Iterator` object, call the `java.util.List` object's `iterator` method, as shown in the following code fragment:

```
Iterator it = authConfigs.iterator();
```

After you create an `Iterator` object, reference the individual `AuthConfigBO` objects by invoking the `Iterator` object's `next` method and casting the return value, as shown in the following code fragment:

```
AuthConfigBO conf = (AuthConfigBO)it.next();
```

Call the `AuthConfigBO` object's `getCustomConfiguration` method to get a `java.util.Map` object that contains key-value pairs of string objects representing the first-level custom configuration settings. You cannot use the `java.util.Map` object to modify configuration settings.

You get the value of a specific key by invoking the `java.util.Map` object's `get` method and passing a string value that represents the key name. However, to retrieve a key value, you must know the key name that is in the User Management XML configuration file (an example of getting the value of an XML key named `FILENAME` is shown in the following code example). For information about the User Management XML configuration file, see ["User Management configuration settings" on page 26](#).

The following example retrieves configuration information from the `java.util.List` object named `authConfigs` that was passed to the authentication provider's `authenticate` method. Notice that a file name value is retrieved. This value is stored in a string variable named `filename` and is used in the implementation of this authentication provider.

Example 2.2 Retrieving User Management configuration information

```
//AuthConfigs is an authenticate parameter
//Create an Iterator object
Iterator it = authConfigs.iterator();

while(it.hasNext()){
    //Get an AuthConfigBO object
    AuthConfigBO conf = (AuthConfigBO)it.next();

    //Retrieve a java.util.Map object that stores User Management
    //configuration information
    java.util.Map configSettings = conf.getCustomConfiguration();

    //Get a value of the FILENAME key
    String filename = (String) configSettings.get("FILENAME");

    //Get the domain name
    String domainName = conf.getDomainName();

    //Do something with filename and domainName
}
```

Performing the authentication operation

After you retrieve user information and required configuration values, you can perform an authentication operation.

You can use a third-party Java API to interact with the authentication server provider you are using. For example, you can use the Java Simple Authentication and Security Layer (SASL) API to perform authentication operations.

For the purposes of this chapter, an authentication operation is performed by using a 0-based, tab-delimited file. If the specified user name, password, and domain values are located in the file, the authentication operation is successful. Otherwise, the authentication operation is unsuccessful.

The following table describes the tab-delimited file.

Column	Description	Example
0	The domain of the user	MyDomain2
1	The login identifier of the user	s_user10
2	The password of the user	password10

In the following code example, a user-defined method named `checkValues` searches a tab-delimited file for the specified user name, password, and domain values. All three values are passed to this method as arguments. This method returns `true` if the authentication operation is successful; otherwise, it returns `false`.

Example 2.3 Performing an authentication operation

```
private boolean checkValues(  
    String userid,  
    String password,  
    String domain,  
    String filename) {  
    BufferedReader br = null;  
    try {  
        //create a BufferedReader object  
        br = new BufferedReader(new FileReader(filename));  
        String line = null;  
  
        //Read each line in the file and search for the values  
        while ((line = br.readLine()) != null) {  
            String[] parts = line.split("\t");  
            if (userid.equals(parts[1]) && domain.equals(parts[0]) &&  
password.equals(parts[2]))  
                return true;  
        }  
    } catch (IOException e) {  
        System.err.println(e);  
    } finally {  
        if (br != null) {  
            try {  
                br.close();  
            } catch (IOException e) {}  
        }  
    }  
    return false;  
}
```

The file name was retrieved from the configuration settings that were passed by User Management. For information, see ["Retrieving configuration information" on page 11](#).

The user name and password values were retrieved from the `java.util.Map` object that was passed to the `authenticate` method. For information, see ["Retrieving authentication values" on page 10](#).

Sending authentication results to User Management

After the authentication provider performs the authentication, it must return the results to User Management. The authentication provider must specify the user name, the domain name, and whether the authentication was successful.

The User Management SPI provides the `AuthResponseImpl` class, which implements the `AuthResponse` interface. You create an `AuthResponse` object and populate it with the appropriate information. The following code fragment creates an `AuthResponse` object:

```
AuthResponse response = new AuthResponseImpl();
```

You can inform User Management that the authentication was successful by invoking the `AuthResponse` object's `setAuthStatus` method and passing `AuthResponse.AUTH_SUCCESS`. You can also inform User Management that the authentication was unsuccessful by invoking the `AuthResponse` object's `setAuthStatus` method and passing `AuthResponse.AUTH_FAILED`.

Use the `return` statement in the `authenticate` method to send the `AuthResponse` object to User Management. For information about the `authenticate` method, see [“About custom authentication providers” on page 8](#).

The following example populates an `AuthResponse` object after a user is successfully authenticated.

Example 2.4 *Returning authentication results in an `AuthResponseImpl` object*

```
//create the AuthResponse object
AuthResponse response = new AuthResponseImpl();
response.setAuthStatus(AuthResponse.AUTH_SUCCESS);
response.setUsername(userName);
response.setDomain(conf.getDomain());
return response;
```

Trapping errors

The `AuthProvider` interface does not throw exceptions, but errors should be included in authentication results. Your authentication provider should collect exceptions and add them to the `AuthResponse` object by using the `setExceptions` method. You can also provide an error description by using the `setErrorMessage` method.

3

Creating Custom Directory Service Providers

This chapter explains how to use the User Management SPI to develop custom directory service providers that you may integrate with User Management.

User Management is packaged with a directory service provider that supports connections to LDAP directories. If your organization uses a non-LDAP repository to store user records, you can create your own directory service provider that works with your repository.

Directory service providers retrieve records from a user store at the request of User Management. User Management regularly caches user and group records in the database to improve performance.

To implement a custom directory service provider, create a user provider and a group provider:

- User providers retrieve all required user records from the repository.
- Group providers retrieve all required user and group records within a specified group, and retrieve all required group records in the repository.

This chapter contains the following information.

Topic	Description	See
Sample directory service provider	Provides information on sample implementations of the directory service provider interfaces	page 15
Directory service provider interface	Describes the directory service provider interfaces to be implemented	page 16
Implementing the directory service provider interfaces	Explains how to develop custom directory service provider interfaces	page 16
Connecting to the directory	Describes how to retrieve directory configuration information and test the connection to User Management	page 23
Configuring User Management to use custom providers	Describes the steps required to update the XML-based configuration file to enable the usage of your custom providers within User Management	page 25

Sample directory service provider

User Management provides a sample implementation of the `DirectoryPrincipalProvider`, `DirectoryGroupProvider`, and `DirectoryUserProvider` interfaces. The sample demonstrates how directory service providers may interact with User Management. You can retrieve the User Management sample from the Adobe LiveCycle Developer Center website at http://partners.adobe.com/public/developer/livecycle/index_samples.html.

Tip: You may find it useful to print the sample code and reference it as you read this chapter.

Directory service provider interfaces

You must implement the following three interfaces when using a custom repository to store user and group information.

Interface	Purpose
<code>DirectoryPrincipalProvider</code>	The base interface for the user provider (<code>DirectoryUserProvider</code>) and group provider (<code>DirectoryGroupProvider</code>) interfaces. Declares methods for retrieving user and group records and for testing the configuration settings and constants for reporting exceptions to User Management.
<code>DirectoryUserProvider</code>	The user provider interface. You must implement the methods and exceptions needed for implementing a custom user provider that retrieves user records and for testing the configuration settings. User Management requires an implementation of this interface to synchronize its database with the user store.
<code>DirectoryGroupProvider</code>	The group provider interface. You must implement the methods and exceptions needed for implementing a custom group provider that retrieves user records that belong to a specific group in the repository and for testing the configuration settings. User Management requires an implementation of this interface when any action is performed on a group, such as adding a group to a policy.

All three interfaces require the implementation of a `getPrincipals` method and a `testConfiguration` method. In all three cases, the entry point for user services is the `getPrincipals` method.

Note: User Management calls the `getPrincipals` method only for the implementations of the `DirectoryUserProvider` and `DirectoryGroupProvider` interfaces.

Implementing the directory service provider interfaces

To retrieve user and group records, you must implement the `DirectoryUserProvider` and `DirectoryGroupProvider` interfaces described in [“Directory service provider interfaces” on page 16](#), and configure User Management for your custom repository by modifying the configuration file as described in [“Configuring User Management for custom directory service providers” on page 25](#).

In this section, implementation guidelines are provided for the interfaces and are based on these samples provided in the User Management installation: `FBDirectoryPrincipalProviderImpl.java`, `FBDirectoryUserProviderImpl.java`, and `FBDirectoryGroupProviderImpl.java`.

Note: For information about the location of these samples, see [“Sample directory service provider” on page 15](#).

DirectoryPrincipalProvider interface

The `DirectoryPrincipalProvider` interface is the base interface for `DirectoryUserProvider` and `DirectoryGroupProvider`. This interface requires that you implement methods that provide the entry point for user services (`getPrincipals`) and a test of the configuration for User Management (`testConfiguration`).

The `getPrincipals` method receives two parameters:

- `config`—A `DirectoryProviderConfig` object that contains information about the user records to be retrieved and User Management configuration information.
- `state`—An implementor-defined object that indicates whether this is the first time the method has been called or the state of the previous call. This object typically contains fields originating from the User Management configuration settings; however, it can also be used to store any information required between `getPrincipals` calls.

The `getPrincipals` method is repeatedly called by User Management until all records (both user and group) are retrieved. The record collection is returned within a `DSPrincipalCollection` object, which also contains the state information to be used in the next call.

The following example shows the `getPrincipals` method, which is the entry point for user services.

Example 3.1 *Entry point for user services*

```
public DSPrincipalCollection getPrincipals (
    DirectoryProviderConfig config,
    Object state) throws IDPEException {
    return grabBatchPrincipals (config, state);
}
```

In the example, processing is handled in the `grabBatchPrincipals` method. A description of the method's algorithm follows.

Declare an object of the `DSPrincipalCollection` class, which will store the user or group records to be retrieved from the custom repository. This object will be returned when the `getPrincipals` method terminates and will contain both the records and the state information used in the next call.

The state information should include the following data:

- Principal type
- Batch size
- Principal's file handle
- Configuration information
- Domain

The following example shows implementor-defined classes used to represent the state information.

Example 3.2 *Representing the state information*

```
protected class FBConfig {
    private String principalType;
    private int batchSize;
    private String principalsFileName;
}
```

```
private Map customConfig;
private String domain;
private static final String RECORD_USER = "USER";
private static final String RECORD_GROUP = "GROUP";

public int getBatchSize() {
    return batchSize;
}
public Map getCustomConfig() {
    return customConfig;
}
public String getDomain() {
    return domain;
}
public String getPrincipalsFileName() {
    return principalsFileName;
}
public String getPrincipalType() {
    return principalType;
}
public void setBatchSize(int batchSize) {
    this.batchSize = batchSize;
}
public void setCustomConfig(Map customConfig) {
    this.customConfig = customConfig;
}
public void setDomain(String domain) {
    this.domain = domain;
}
public void setPrincipalsFileName(String principalsFileName) {
    this.principalsFileName = principalsFileName;
}
public void setPrincipalType(String principalType) {
    this.principalType = principalType;
}
}
protected class FBPrincipalState{
    public FBConfig config;
    public File principalsFile;
    public BufferedReader brPrincipals;

    public void finalize() {
        try {
            brPrincipals.close();
        } catch (Exception e) {
            System.err.println(e);
        }
    }
}
}
```

If this is the first time the `getPrincipals` method is called (state parameter is null), perform the initializations required to begin collecting the information. Otherwise, copy the state into the `DSPrincipalCollection` object to propagate it to the next call.

In the following example, `dpc` is the `DirectoryProviderConfig` object, which is used to read the configuration preferences. The implementor-defined `prepEnumeration` method gathers the state information.

Example 3.3 Determining the state

```
FBPrincipalState fbState = null;
if (state == null)
    fbState = prepEnumeration(dpc);
else if (state instanceof FBPrincipalState)
    fbState = (FBPrincipalState) state;
else {
    // throw exception
}

ret.setState(fbState); //carry state forward
```

Collect the records, which are objects of the `DSPrincipalRecord` class. If there are no objects to collect, `null` is returned. It is recommended that implementations try to return fewer than 1000 records at a time for optimal performance.

After you obtain a new record, add it to the collection. In the following example, this task is done by invoking the `DSPrincipalCollection` object's `addDSPrincipalRecord` method.

Example 3.4 Collecting records

```
int batchcount = 0;
int maxbatch = fbState.config.getBatchSize();
String line = null;
while (batchcount < maxbatch &&
    (line = fbState.brPrincipals.readLine()) != null) {
    ++batchcount;
    DSPrincipalRecord rec = grabPrincipal(fbState, line);
    ret.addDSPrincipalRecord(rec);
}

if (batchcount == 0 && line == null)
    return null;
```

For each record, set the principal type (user or group) according to the configuration information. It is recommended that you consider the option of setting the organization name of the principal.

For all principals, you must set the following information:

- Domain name
- Canonical name
- Primary email address
- Full name

For users, you must set the User identification. For groups, you must set the type of group record.

In the following example, the `grabPrincipal` method performs these tasks. Each setting is used and is labeled as either required or optional within the example.

Example 3.5 *Setting the principal information*

```
private DSPrincipalRecord grabPrincipal(
    FBPrincipalState state,
    String line) {

    //Create a DSPrincipalRecordImpl object
    DSPrincipalRecord rec = new DSPrincipalRecord();

    rec.setPrincipalType(state.config.getPrincipalType());
    String[] parts = line.split("\t");

    // These settings apply to all principals:
    rec.setDomainName(parts[0]); // REQUIRED
    rec.setCanonicalName(parts[1]); // REQUIRED
    rec.setDescription(parts[2]); // OPTIONAL
    rec.setEmail(parts[3]); // REQUIRED
    rec.setCommonName(parts[4]); // REQUIRED
    rec.setOrg(parts[5]); // OPTIONAL

    //These settings apply only to users:
    if (state.config.getPrincipalType().equals(FBConfig.RECORD_USER)) {
        rec.setUserid(parts[6]); // REQUIRED
        //password is parts[7] (useful only for authentication providers)
        rec.setFamilyName(parts[8]); // OPTIONAL
        rec.setGivenName(parts[9]); // OPTIONAL
        rec.setInitials(parts[10]); // OPTIONAL
        rec.setTelephoneNumber(parts[11]); // OPTIONAL
        rec.setPostalAddress(parts[12]); // OPTIONAL
        rec.setLocale(parts[13]); // OPTIONAL
        rec.setTimezone(parts[14]); // OPTIONAL
    }

    // These settings apply only to groups:
    if (state.config.getPrincipalType().equals(FBConfig.RECORD_GROUP)) {
        rec.setGroupType(DSPrincipalRecord.GROUPTYPE_PRINCIPALS);
    }

    return rec;
}
```

The implementation of the `DirectoryUserProvider` interface has the same required methods as the `DirectoryPrincipalProvider` interface.

You may optionally extend the methods defined in the base class. The example reuses the base class method implementations.

DirectoryUserProvider interface

The `DirectoryUserProvider` interface inherits from the `DirectoryPrincipalProvider` interface and must be implemented for your custom user provider service.

You may optionally override the methods defined in the base class, as shown in the following example.

Example 3.6 *Implementing the DirectoryUserProvider interface*

```
public class FBDirectoryUserProviderImpl
    extends FBDirectoryPrincipalProviderImpl
    implements DirectoryUserProvider {

    public FBDirectoryUserProviderImpl() {
        super();
    }
}
```

DirectoryGroupProvider interface

The `DirectoryGroupProvider` interface inherits from the `DirectoryPrincipalProvider` interface. You may optionally override the methods defined in the base class, but you must also implement one more method—the `getGroupMembers` method.

The `getGroupMembers` method accepts two parameters:

- A `DirectoryProviderConfig` object containing information used to connect to the repository
- A `DSPrincipalIdRecord` object that identifies the group to be retrieved

The method returns a `DSGroupContainmentRecord` object, which is a container for the user and group records belonging to a group.

The `getGroupMembers` method depends on an implementor-defined object that contains the state information as described in [“Determining the state” on page 19](#), as well as the group containment file name. An implementation is shown in the following example.

Example 3.7 *Group state information*

```
private class FBGroupConfig {
    private FBConfig commonConfig;
    private String groupContainmentFileName;
    public FBConfig getCommonConfig() {
        return commonConfig;
    }

    public String getGroupContainmentFileName() {
        return groupContainmentFileName;
    }

    public void setCommonConfig(FBConfig config) {
        commonConfig = config;
    }
}
```

```
        public void setGroupContainmentFileName(String string) {
            groupContainmentFileName = string;
        }
    }

private FBGroupConfig createFBGroupConfig(DirectoryProviderConfig dpc){
    FBGroupConfig ret = new FBGroupConfig();
    ret.setCommonConfig(createFBConfig(dpc));

    ret.setGroupContainmentFileName(
        (String)ret.getCommonConfig().getCustomConfig().get(
            "GROUPCONTAINMENT_FILE_NAME"
        )
    );
    return ret;
}
```

To retrieve the user records, implement the `getGroupMembers` method by performing these tasks:

1. Use the User Management configuration information stored in the `DirectoryProviderConfig` object (in this case by invoking the `createFBGroupConfig` method).
2. Set up a `DSGroupContainmentRecord` object.
3. Read in the user records and collect each one (in this case, it is accomplished by calling a method named `grabContainment`). Only those user or group records that belong to the group should be collected. It is also possible to collect nested groups. To do so, compare the group name contained in each record with the group name of the principal. When a group is found, store its domain and group name.
4. Collect all the principals belonging to that group.

Example 3.8 Retrieving group members

```
public DSGroupContainmentRecord getGroupMembers(
    DirectoryProviderConfig dpc,
    DSPrincipalIdRecord group
) throws IDPException {
    // Check to make sure the group domain is known:
    if (group == null) {
        // throw IDPException
    }

    FBGroupConfig config = createFBGroupConfig(dpc);
    DSGroupContainmentRecord ret = null;
    String groupName = group.getCanonicalName();

    // open file and search for this groupname.
    try {
        BufferedReader brContainment = new BufferedReader(
            new FileReader(config.getGroupContainmentFileName())
        );
    }
```

```
String line = null;
while ((line = brContainment.readLine()) != null) {

    ret = grabContainment(config, groupName, line);

    if (ret != null)
        break;
}
brContainment.close();
} catch (Exception e) {
    // throw IDPEException
}
return ret;
}

private DSGroupContainmentRecord grabContainment(
    FBGroupConfig config,
    String groupName,
    String line
) {
    String[] parts = line.split("\t");
    if (! parts[0].equals(groupName))
        return null;

    String domain = config.getCommonConfig().getDomain();

    DSGroupContainmentRecord ret = new DSGroupContainmentRecord();
    ret.setDomainName(domain);
    ret.setCanonicalName(groupName);

    for (int i=1;i<parts.length;i++) {
        DSPrincipalIdRecord memprin = new DSPrincipalIdRecord();
        memprin.setDomainName(domain);
        memprin.setCanonicalName(parts[i]);
        ret.addPrincipalMember(memprin);
    }

    return ret;
}
```

Connecting to the directory

This section provides information on how to retrieve directory properties from the directory service provider configuration.

Getting the directory properties

The directory service provider interface implementations all rely on the directory service provider configuration. To obtain this information, use the `DirectoryProviderConfig` object, which provides these methods:

- `getDomain`—Retrieves the domain name
- `getGroupConfig`—Retrieves the group provider configuration information by returning an instance of a class that implements the `GroupConfigBO` interface (see [GroupConfigBO interface](#))
- `getUserConfig`—Retrieves the user provider configuration information by returning an instance of a class that implements the `UserConfigBO` interface (see [UserConfigBO interface](#))

Note: The `getGroupConfig` method and `getUserConfig` method both return implementations of interfaces that define a `getCustomConfiguration` method, which makes it possible to access the configuration settings within a `java.util.Map` object that contain key-value pairs of strings that represent the first-level configuration settings. These strings may be parsed as needed.

See [“Implementing the directory service provider interfaces” on page 16](#) for examples of the `DirectoryProviderConfig` object’s usage.

GroupConfigBO interface

Classes implementing the `GroupConfigBO` interface store configuration information about the group provider. This information is obtained from the XML-based User Management configuration file (see [“Configuring User Management for custom directory service providers” on page 25.](#))

For your convenience, the User Management SPI provides a generic implementation called `GenericGroupConfigBO`.

UserConfigBO interface

Classes that implement the `UserConfigBO` interface store configuration information about the user provider. This information is obtained from the XML-based User Management configuration file (see [“Configuring User Management for custom directory service providers” on page 25.](#))

For your convenience, the User Management SPI provides a generic implementation called `GenericUserConfigBO`.

Testing the connection

This section describes how you can test the connection to the directory.

► **To test the connection to the directory:**

1. Log into the Adobe LiveCycle Administration Console and click **Settings > User Management > Domain Management**.
2. Under **Synchronize All Directories Manually**, click **OK**.

Note: This test will not call your `testConfiguration` method; it calls the `getPrincipals` method.

Configuring User Management for custom directory service providers

When configuring User Management for custom directory service providers, you must also modify its configuration file.

► **To configure User Management for custom directory service providers:**

1. Log into the Adobe LiveCycle Administration Console and click **Settings > User Management > Configuration > Import and export configuration files**.
2. Click **Export** and, in the File Download dialog box, click **Open**. The config.xml file is downloaded.
3. Edit the config.xml file according to the directions provided in [“Configuring User Management to use directory service providers” on page 33](#).
4. Click **Browse** and then click **Import**. The config.xml file is uploaded to User Management.

4

Registering Custom Service Providers

You must register custom service providers with User Management after you have developed them using the User Management SPI.

Most of the custom service providers that you can develop for User Management require you to configure User Management appropriately. For example, User Management requires some service providers to be associated with a domain. You will need to create a new User Management domain that includes settings for your custom service providers. This is because you cannot edit domains from the user interface that contain custom service providers. Otherwise, you will be unable to use the user interface to modify the LDAP settings in that domain.

This chapter contains the following information.

Topic	Description	See
User Management configuration settings	Describes how User Management configuration settings are implemented and explains the structure of the configuration file	page 26
Defining domains for custom service providers	Describes the settings you need to configure to create a new User Management domain	page 28
Configuring User Management to use custom authentication providers	Describes the settings you need to configure to integrate your custom authentication service provider with User Management	page 29
Configuring User Management to use directory service providers	Describes the settings you need to configure to integrate your custom directory service provider with User Management	page 33

User Management configuration settings

User Management configuration settings include information about the service providers that User Management employ.

User Management can export the configuration settings to a file on your local file system. The settings are expressed in XML format. XML elements represent the different product components and their configuration settings.

Initially, XML includes settings only for the service providers packaged with User Management. You need to add settings for your custom service providers.

To retrieve the XML file (config.xml), log into the User Management web pages by using an administrator account and export the User Management configuration. After you edit the file, import it to apply the changes.

For more information about exporting and importing configuration settings, see the *User Management Help*.

Caution: When you are editing the config.xml file, change only the configuration settings that affect your custom service providers. When you import the file, User Management applies all the settings that the file defines.

XML configuration file

The configuration file that User Management exports is structured so that each configurable aspect of the product is represented by a different XML element.

The following code shows the top levels of the config.xml file. The lowest level shown includes elements that represent the configurable components.

Example 4.1 XML configuration file

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE preferences (View Source for full doctype...)>
- <preferences EXTERNAL_XML_VERSION="1.0">
- <root type="system">
  <map />
- <node name="Adobe">
  <map />
- <node name="LiveCycle">
  <map />
- <node name="Config">
  <map />
- <node name="UM">
  <map />
  + <node name="AuthProviders">
  + <node name="AuthSchemes">
  + <node name="DirectoryServices">
  + <node name="DisplaySettings">
  + <node name="Domains">
  + <node name="GroupProviders">
  + <node name="SAML">
  + <node name="StorageProviders">
  + <node name="Synchronization">
  + <node name="UserProviders">
  </node>
  </node>
  </node>
  </node>
  </root>
</preferences>
```

The `AuthProviders`, `Domains`, `GroupProviders`, and `UserProviders` elements apply to custom service providers. For example, the `node` element that has the `name` attribute value of `Domains` contains an XML element for each User Management domain.

XML element types in the configuration file

In general, the configuration file uses three element types to structure the configuration information:

- `node` elements represent configurable components in the environment. For example, a `node` element exists for each authentication provider that User Management uses. `node` elements each have a `name` attribute and contain other `node` elements, and must include at least one `map` element.
- `map` elements contain the elements that hold the property-value pairs that constitute configuration settings. The configuration settings are associated with the `map` element's parent `node` element. `map` elements can contain `entry` elements.
- `entry` elements use attributes to specify property-value pairs. `entry` elements have no content, but they have the attributes `key` and `value` that represent property names and values, respectively. An `entry` element can define only one property-value pair.

Defining domains for custom service providers

Domains define different user bases. The boundary of a domain is usually defined according to the way your organization is structured or how your user store is set up.

User Management domains provide configuration settings that authentication providers and directory service providers use.

In the configuration XML that User Management exports, the `root` node that has the `name` attribute value of `Domains` contains an XML element for each domain defined for User Management. Each of these elements contain other elements that define aspects of the domain associated with specific service providers.

It is strongly recommended that you define a new domain that provides configuration settings for your custom service providers. For example, if you are implementing a custom authentication provider and a custom directory service provider that connect to the same user store, you should create a new domain.

The following example shows the structure of the XML element that represents a domain in the User Management configuration settings (the attribute values in italics are the values you can customize). Your domains should never be local (see the `map` entry whose `key` is `isLocal`).

Example 4.2 Defining a domain

```
<node name="your_domain_name">
  <map>
    <entry key="name" value="your_domain_name"/>
    <entry key="description" value="a_description_for_your_domain"/>
    <entry key="isLocal" value="false"/>
  </map>
  <node name="AuthConfigs"> ... </node>
  <node name="DirectoryConfigs"> ... </node>
</node>
```

To create a new domain, insert similar XML tags into the exported configuration file. Place the tags within the `node` element that has the `name` attribute value of `Domains`.

Note: The `node` element that has the `name` attribute value of `AuthConfigs` contains configuration information that authentication providers need. The `node` element that has the `name` attribute value of `DirectoryConfigs` contains configuration information that directory providers need.

Configuring User Management to use custom authentication providers

User Management configuration settings are used to register authentication providers, specify domains used for authentication, and associate authentication providers with domains.

You can also include domain information that your authentication provider requires at run time, such as parameters for connecting to the user store.

For information about User Management configuration settings, see [“User Management configuration settings” on page 26](#).

Identifying authentication providers

The User Management configuration settings must identify each authentication provider that User Management uses.

Each authentication provider is identified by a corresponding element in the User Management XML configuration file. It is necessary to add an element to the first-level `AuthProviders` node that indicates which authentication providers should be globally loaded. These providers are associated with the specific domains listed in each of the child nodes.

Each domain element includes a `node` element that has the `name` attribute of the `AuthProviders` node. The `AuthProviders` node contains a separate `node` element that describes each authentication provider.

These nodes include elements that specify all the information that User Management requires to use the authentication provider, such as the name of the class that User Management should use to initiate the authentication provider and a pointer to the domain settings associated with the authentication provider.

The following example shows an XML `node` element within `AuthProviders`, arbitrarily named `SPIFB` in this case, that corresponds to an authentication provider. As shown by the highlighted portions, several relationships apply within both the `AuthProviders` node and the `Domains` node:

- The path specified in the `AuthProviders` node's entry named `configInstance` contains `SPIDom`, which is arbitrarily named and corresponds to an identically named node specified within `Domains`. It also corresponds to part of the path specified in the `AuthProviders` node's entry named `configInstance`.
- The `AuthProviders` node's `SPIFB` element corresponds to both the last part of the path specified in its entry named `configInstance` and an identically named node within the `SPIDom` node specified within `Domains`.
- The `SPIDom` node specified within `Domains` must be identical to the `value` used in its `map` node's `name` entry.

Because these relationships apply, be careful when renaming nodes to update the corresponding nodes.

Example 4.3 Identifying an authentication provider

```
- <node name="AuthProviders">
  <map />
  + <node name="JAAS">
  + <node name="Kerberos">
  + <node name="LDAP">
  - <node name="SPIFB">
    - <map>
      <entry key="configured" value="true" />
      <entry key="enabled" value="true" />
      <entry key="visibleInUI" value="false" />
      <entry key="allowMultipleConfigs" value="false" />
      <entry key="className" value="com.adobe.idp.samples.spi.authentication.FBAuthenticationProviderImpl" />
      <entry key="configInstance" value="/Adobe/LiveCycle/Config/UM/Domains/SPIDom/AuthConfigs/SPIFB" />
      <entry key="order" value="2" />
    </map>
  + <node name="DefaultConfig">
  </node>
<node name="Domains">
  <map />
  + <node name="DefaultDom">
  - <node name="SPIDom">
    - <map>
      <entry key="description" value="SPI Testing Domain" />
      <entry key="name" value="SPIDom" />
      <entry key="isLocal" value="false" />
    </map>
  - <node name="AuthConfigs">
    <map />
    - <node name="SPIFB">
      - <map>
        <entry key="authProviderNode" value="/Adobe/LiveCycle/Config/UM/AuthProviders/SPIFB" />
        <entry key="FILENAME" value="c:\users500" />
      </map>
    </node>
  </node>
</node>
```

Therefore, to identify an authentication provider, insert similar tags into the exported configuration file. Place the tags within the `node` element that has the `name` attribute value of `AuthProviders`, and be aware of related names used in `Domains`.

Specify a name for the authentication provider in the `node` element you insert. The name must be unique to all authentication providers in the context of the configuration settings.

The following table describes the attribute values you need to provide for the `entry` elements inside the first `map` element.

Key name	Value
enabled	true or false Specifies whether User Management should use this provider. User Management uses this provider if <code>value</code> is <code>true</code> .
configured	true or false Specifies whether the domain has been configured for this authentication provider. If the domain has been configured, <code>value</code> should be <code>true</code> .

Key name	Value
<code>visibleInUI</code>	<code>true</code> or <code>false</code> Specifies whether this authentication provider should appear as a selectable option in the User Management web pages. For your authentication provider, <code>value</code> should be <code>false</code> .
<code>allowMultipleConfigs</code>	<code>true</code> or <code>false</code> Specifies whether more than one domain configuration instance exists for this authentication provider. If multiple configuration instances exist, <code>value</code> should be <code>true</code> .
<code>order</code>	Any integer, beginning at 1. This integer is reserved for future use.
<code>classname</code>	The fully qualified name of the class used to initiate this authentication provider. User Management calls the <code>authenticate</code> method of this class to request authentication.
<code>configInstance</code>	The value of the <code>name</code> attribute of the <code>node</code> element that contains the domain configurations for this authentication provider. You must provide cross-references to all the domains that use this authentication provider. This enables you to indicate which domain to use for this authentication provider. Each node element inside <code>AuthConfigs</code> represents a configuration instance.

Configuring domains for authentication providers

The configuration settings for User Management domains include information to support each associated authentication provider. You must add configuration settings to the domain with which your authentication service provider is associated.

If you have not already set up a domain, you need to set one up immediately. For more information, see [“Defining domains for custom service providers” on page 28](#).

The following example shows a node that provides the domain configuration for an authentication provider and provides a reference from inside your domain up to the authentication provider entry. For a more complete example, see [“Configuring User Management to use directory service providers” on page 33](#).

Example 4.4 Configuring a domain for an authentication provider

```
<node name="your_authentication_provider">
  <map>
    <entry key="authProviderNode"
      value="/Adobe/LiveCycle/Config/UM/AuthProviders/name"/>
    <entry key="additional_key_name" value="corresponding_value"/>
  </map>
</node>
```

To configure your domain, you must add a similar XML tag in the exported configuration file. Place the tag in the `node` element that has the `name` attribute value of `AuthConfigs`. This `AuthConfigs` node is in the domain that the authentication provider uses, as shown in [“Defining a domain” on page 28](#).

The `node` element for your authentication provider can include any number of properties that the provider requires. The `node` element must include `entry` elements that contain the two property-value pairs described in the following table.

Key name	Value
<code>authProviderNode</code>	The path to the node in the XML configuration file that represents your authentication provider.
<i>additional_key_name</i>	Any additional information that the authentication service provider needs at run time. Use this key name to create as many custom keys as you need to store the configuration information for your authentication provider to connect to the authentication system.

Note: Ensure that your domain for custom providers is uniquely named. Assume that the name will be treated in a case-insensitive manner.

In the following example, the two property-value pairs are `authProviderNode` and `FILENAME`. The `FILENAME` entry is an arbitrary example of a key that you may specify.

Example 4.5 Configuring a domain

```
<node name="Domains">
  <map />
+ <node name="DefaultDom">
- <node name="SPIDom">
  + <map>
  - <node name="AuthConfigs">
    <map />
    - <node name="SPIFB">
      - <map>
        <entry key="authProviderNode" value="/Adobe/LiveCycle/Config/UM/AuthProviders/SPIFB" />
        <entry key="FILENAME" value="c:\users500" />
      </map>
    </node>
  </node>
</node>
```

Note: Do not add children nodes; only first-level keys can be used.

Configuring User Management to use directory service providers

After you download the XML file, edit the file according to the following example. Look for a node named `Domains` and add the `DirectoryConfigs` subnode after `</map>`, as shown in the example that follows.

Note: The highlighted entries must match corresponding nodes within the `GroupProviders` and `UserProviders` nodes, which are discussed in [“Configuring the custom group provider” on page 34](#) and [“Configuring the custom user provider” on page 34](#).

Example 4.6 Configuring the domain to use custom directory service providers

```
<node name="Domains">
  <map />
+ <node name="DefaultDom">
- <node name="SPIDom">
  - <map>
    <entry key="description" value="SPI Testing Domain" />
    <entry key="name" value="SPIDom" />
    <entry key="isLocal" value="false" />
  </map>
+ <node name="AuthConfigs">
- <node name="DirectoryConfigs">
  <map />
- <node name="sdkspi_DirectoryConfig01">
  <map />
- <node name="NameDoesntMatterGroupConfig">
  - <map>
    <entry key="BATCH_SIZE" value="150" />
    <entry key="PRINCIPAL_FILE_NAME" value="c:\groups500" />
    <entry key="GROUPCONTAINMENT_FILE_NAME" value="c:\containment500" />
    <entry key="groupProviderNode" value="/Adobe/LiveCycle/Config/UM/GroupProviders/GroupSPIFB" />
  </map>
</node>
- <node name="NameDoesntMatterUserConfig">
  - <map>
    <entry key="BATCH_SIZE" value="150" />
    <entry key="PRINCIPAL_FILE_NAME" value="c:\users500" />
    <entry key="userProviderNode" value="/Adobe/LiveCycle/Config/UM/UserProviders/UserSPIFB" />
  </map>
</node>
</node>
</node>
</node>
</node>
```

Note: The names of the nodes within `DirectoryConfigs` may be arbitrarily chosen. Therefore, `sdkspi_DirectoryConfig01`, `NameDoesntMatterGroupConfig`, and `NameDoesntMatterUserConfig` are arbitrary names. The only requirement is that the names within the same node depth must be distinct. In addition, the `GROUPCONTAINMENT_FILE_NAME` and `PRINCIPAL_FILE_NAME` entries are arbitrary examples of keys you may specify. In such cases, do not add children nodes; only first-level keys may be used.

To configure your custom group provider, specify the name of the class that implements the `DirectoryGroupProvider` interface. In the following example, this class is `FBDirectoryGroupProviderImpl`. Look for a node named `GroupProviders` and add the following subnode after `</map>`. In this case, it has been arbitrarily named `GroupSPIFB`. You may give it a name you prefer, but it must match the name in the path that is provided in the `groupProviderNode` entry within the `Domains` node, as shown by the yellow highlighted text in [Configuring the domain to use custom directory service providers](#) and in the following example.

Example 4.7 *Configuring the custom group provider*

```
<node name="GroupProviders">
  <map />
+ <node name="LDAP">
- <node name="GroupSPIFB">
  - <map>
    <entry key="className" value="com.adobe.idp.samples.spi.directoryservices.FBDirectoryGroupProviderImpl" />
    <entry key="enabled" value="true" />
    <entry key="configured" value="true" />
    <entry key="allowMultipleConfigs" value="true" />
  </map>
</node>
</node>
```

To configure your custom user provider, specify the name of the class that implements the `DirectoryUserProvider` interface. In the following example, this class is `FBDirectoryUserProviderImpl`. Look for a node named `UserProviders` and add the following subnode after `</map>`. In this case, it has been arbitrarily named `UserSPIFB`. You may give it a name you prefer, but it must match the name in the path that is provided in the `userProviderNode` entry within the `Domains` node, as shown by the blue highlighted text in [Configuring the domain to use custom directory service providers](#) and in the following example.

Example 4.8 *Configuring the custom user provider*

```
<node name="UserProviders">
  <map />
+ <node name="LDAP">
- <node name="UserSPIFB">
  - <map>
    <entry key="className" value="com.adobe.idp.samples.spi.directoryservices.FBDirectoryUserProviderImpl" />
    <entry key="enabled" value="true" />
    <entry key="configured" value="true" />
    <entry key="allowMultipleConfigs" value="true" />
  </map>
</node>
</node>
```

5

Deploying Custom Service Providers

This chapter describes how to deploy a custom service provider to the J2EE application server on which User Management is deployed. You must perform the tasks described in this chapter in order for User Management to use the custom service provider that is created by using the User Management SPI.

To deploy your custom service provider, package it into a JAR file. After you create the JAR file, you must place the file into the LiveCycle.ear file by running Adobe Configuration Manager. After you run Configuration Manager, the custom service provider is deployed and User Management can use it to authenticate users.

This chapter provides the following information.

Topic	Description	See
Packaging your custom service provider	Explains packaging your Java application into a JAR file	page 35
Repackaging the LiveCycle EAR file	Explains how to use Configuration Manager to repack the LiveCycle.ear file to include the JAR file	page 35

Packaging your custom service provider

After you finish developing a custom service provider, package your Java project into a JAR file. Include all CLASS files that are located in your Java application in the JAR file. For information on creating a JAR file, go to <http://java.sun.com/>.

If your custom service provider is dependent on external JAR files, you must package these JAR files along with your custom service provider.

Repackaging the LiveCycle EAR file

You must repack the LiveCycle.ear file to include the JAR file(s) that contain your custom service provider and external JAR files that your custom service provider is dependent on. To repack the LiveCycle.ear file, you run Configuration Manager, which is a wizard-like tool that automates the product configuration and assembly process. For more information about Configuration Manager see the *Installing and Configuring* guide that accompanies your LiveCycle product.

Before you run Configuration Manager, you must place your JAR file in the following directory:

```
C:\Adobe\LiveCycle\components\um\<app_server>\ext
```

where C:\ is the drive on which User Management is installed and *app_server* is the J2EE application on which User Management is deployed. Assume that User Management is deployed on JBoss. In this situation, place your JAR file in the following directory:

```
C:\Adobe\LiveCycle\components\um\jboss\ext
```

Note: After you place the JAR in the appropriate directory, you can run Configuration Manager. It is recommended that you configure your LiveCycle product the same as you did when you originally installed it by selecting the same options that you selected when you first ran Configuration Manager.

Deploying custom service providers

If User Management is deployed on JBoss, you must update the application.xml file to include the name of the JAR file(s) that make up your custom service provider. Configuration Manager merges the changes that you make to the application.xml file to the application.xml file that is located in the repackaged LiveCycle.ear file.

If User Management is deployed on IBM® WebSphere or BEA Weblogic, you must also perform an additional step. You must open the um.jar file and modify the manifest file to include references that specify all the JAR files that you placed in the LiveCycle.ear file. This step is not necessary if User Management is deployed on JBoss.

► To deploy a custom service provider on JBoss:

1. Copy the custom JAR file(s) to the C:\Adobe\LiveCycle\components\um\jboss\ext directory.
2. Update the application.xml file.
3. Run Configuration Manager to repackage the LiveCycle.ear file. For information, see [To repackage the LiveCycle.ear file by running Configuration Manager:](#)

► To deploy a custom service provider on WebSphere:

1. Copy custom JAR file(s) to the C:\Adobe\LiveCycle\components\um\websphere\ext directory.
2. Open the C:\Adobe\LiveCycle\components\um\websphere\ejb\um.jar file and edit the manifest to include the custom JAR file(s) in the **Class-Path:** entry.
3. Run Configuration Manager to repackage the LiveCycle.ear file. For information, see [To repackage the LiveCycle.ear file by running Configuration Manager:](#)

► To deploy a custom service provider on Weblogic:

1. Copy custom JAR file(s) to the C:\Adobe\LiveCycle\components\um\weblogic\ext directory.
2. Open the C:\Adobe\LiveCycle\components\um\weblogic\ejb\um.jar file and edit the manifest to include the custom JAR file(s) in the **Class-Path:** entry.
3. Run Configuration Manager to repackage the LiveCycle.ear file. For information, see [To repackage the LiveCycle.ear file by running Configuration Manager:](#)

► To repackage the LiveCycle.ear file by running Configuration Manager:

1. Navigate to the *LiveCycle root*/ConfigurationManager directory and start Configuration Manager:
 - (Microsoft® Windows®) Double-click **ConfigurationManager.exe**.
 - (Linux®) Run the configurationmanager.sh file.
2. On the Configuration Manager Welcome screen, click **Next**.
3. Select **Manual Configuration**, select one of the options below it, and then click **Next**.
4. Select the product(s) you want to configure and click **Next**.

For information about configuring your LiveCycle product, see the *Installing and Configuring* guide that accompanies your LiveCycle product.

5. Choose the application server that you are deploying to and click **Next**:
 - WebSphere (supported on Linux)
 - JBoss (supported on Windows)

6. On the Application Deployment Configuration screen, accept the following default information for the application that is being assembled, or type custom descriptions:
 - File name:** The default is LiveCycle.ear, or type another name for the final EAR file that you deploy.
 - Application context:** The default is LiveCycle, or use a descriptive name of your choice.
 - Application description:** The default is LiveCycle, or use a brief description of the application.
7. Select **User Management** in the product assembly. (You must select User Management because you are repackaging the LiveCycle.ear file with a JAR that contains a custom service provider for User Management.)

Note: This configuration dialog box appears only if LiveCycle Forms is selected.
8. On the Data Manager Module Configuration screen, select **Enable SSL** if you are using SSL security on your application server.
9. Type the SSL credential password. (If you have not yet set up your SSL credential, you can type a password here and use it when you create an SSL credential.)
10. (Optional) Enter a directory to use for Adobe LiveCycle products temp file.
11. Click **Next** and, on the Data Manager Module Configuration Continued screen, accept the default values for the following properties or type new values:
 - localDocumentStorageSweepInterval
 - globalDocumentStorageSweepInterval
 - defaultDocumentMaxInlineSize
 - globalDocumentStorageRootDir
 - defaultDocumentDisposalTimeout
12. (Optional) Select **globalDocumentStorageForceNFS** to set this property to `true`.
13. Click **Next** and, on the Font Manager Module Configuration screen, click **Next**.

The Configuration and Assembly Progress screen displays the progress of the configuration process. The Configuration and Assembly Details screen displays information about the product configuration and assembly.
14. Click **Next**, and then click **Finish** to exit Configuration Manager.

The deployable files are located in the following directory:

 - (Windows) *LiveCycle root*\ConfigurationManager\export
 - (Linux) *LiveCycle root*/lifecycle/ConfigurationManager/export
15. Redeploy the LiveCycle.ear file to the J2EE application server that is hosting your LiveCycle product. For information, see the *Installing and Configuring* guide that accompanies your LiveCycle product.
16. Restart your J2EE application server.

Note: If you originally used the JBoss turnkey option, you can use it again to repackage the LiveCycle.ear file. For information, see the chapter that discusses how to install your LiveCycle product by using the turnkey in the product's *Installing and Configuring* guide.

Index

A

- adding
 - exceptions to authentication response 14
 - user and group records to collection 19
- Adobe Configuration Manager 35
- AuthConfigBO interface 10
- AuthConfigBO objects 11
- authenticate method 8
- authentication
 - performing operations 12
 - process 6, 9
 - result errors 14
 - retrieving configuration information 11
 - retrieving values 10
 - sending results to User Management 14
- authentication providers
 - configuring domains for 31
 - configuring User Management to use 29
 - creating 8
 - identifying 29
- AuthProvider interface 10
- AuthResponse interface 10
- AuthResponse object, creating 14
- AuthResponseImpl class 10
- AuthScheme interface 10

C

- checkValues method 12
- collecting user and group records 19
- config object 17
- ConfigFactory class 10
- configuration file
 - importing and exporting settings 26
 - modifying 25
 - structure of 27
 - XML element types 28
- Configuration Manager. *See* Adobe Configuration Manager
- connection, directory, testing 24
- creating
 - authentication providers 8
 - AuthResponse object 14
 - directory service providers 15
 - domains for custom service providers 28
 - Iterator object 11
 - JAR files 35
 - service providers 7
- custom authentication providers. *See* authentication providers
- custom service providers. *See* service providers

D

- defining domains 28
- deploying custom service providers 36
- directory service providers
 - about 15
 - configuring User Management for 25
 - configuring User Management to use 33
 - implementing interfaces 16
 - interfaces 16
 - retrieving directory properties from configuration 23
 - sample implementation 15
- DirectoryGroupProvider interface 16, 21
- DirectoryPrincipalProvider interface 17
- DirectoryProviderConfig object 17, 21, 24
- DirectoryUserProvider interface 16, 21
- domains
 - configuring for authentication providers 31
 - configuring to use directory service providers 33
 - defining for custom service providers 28
- DSGroupContainmentRecord object 21
- DSPrincipalCollection class 17
- DSPrincipalIdRecord object 21

E

- element types, configuration file 28
- entry elements 28

G

- get method 10
- getCustomConfiguration method 24
- getDomain method 24
- getGroupConfig method 24
- getGroupMembers method 21
- getPrincipals method 17
- getUserConfig method 24
- grabBatchPrincipals method 17
- group providers
 - about 15
 - configuring 34
 - retrieving configuration information 24
- GroupConfigBO interface 24

I

- Iterator object, creating 11

J

- JAR file 7, 35
- Java API 8
- JBoss, deploying custom service providers on 36

L

- LDAP directories 15
- library files 7
- LiveCycle.ear file, repackaging 35

M

- map elements 28
- methods
 - authenticate 8
 - checkValues 12
 - get 10
 - getCustomConfiguration 24
 - getDomain 24
 - getGroupConfig 24
 - getGroupMembers 21
 - getPrincipals 17
 - getUserConfig 24
 - grabBatchPrincipals 17
 - setExceptions 14

N

- node elements 28

P

- packaging custom service providers 35

R

- repackaging LiveCycle.ear file 35
- retrieving
 - authentication values 10
 - configuration information 11
 - group members 22
 - user and group provider configuration information 24
 - user and group records 16

S

- service providers
 - about deploying 35
 - about registering 26
 - creating 7
 - defining domains for 28
 - deploying 36
 - packaging Java project into JAR file 35
- setExceptions method 14
- setting principal information 20
- SPI classes 10
- SPI interfaces 10
- state object 17

T

- tab-delimited file, searching 12
- testing directory connection 24

U

- um-spi.jar file 7
- user providers
 - about 15
 - configuring 34
 - retrieving configuration information 24
- UserConfigBO interface 24

W

- WebLogic, deploying custom service providers on 36
- WebSphere, deploying custom service providers on 36

X

- XML configuration file. *See* configuration file
- XML element types, configuration file 28



What's New

Adobe® LiveCycle™ Form Manager

Version 7.2

- Platform Support [More ...](#)
- Improved Installation and Configuration Documentation [More ...](#)
- Improved Configuration Manager [More ...](#)

July 2006

Platform Support

Adobe® LiveCycle™ Form Manager is now supported on a standard set of operating systems, application servers, and databases.

For a complete list of the supported application servers, operating systems, and databases, see the *Installing and Configuring LiveCycle for JBoss*, *Installing and Configuring LiveCycle for WebSphere*, or *Installing and Configuring LiveCycle for WebLogic* guide.

[Back to top](#)

Improved Installation and Configuration Documentation

To improve the interoperability experience of LiveCycle products, the installation and configuration instructions for LiveCycle Form Manager have been combined with the instructions for Adobe LiveCycle Assembler, Adobe LiveCycle Forms, Adobe LiveCycle PDF Generator, Adobe LiveCycle Print, and Adobe LiveCycle Workflow.

The installation and configuration guides are specific to each application server:

- *Installing and Configuring LiveCycle for JBoss*
- *Installing and Configuring LiveCycle for WebSphere*
- *Installing and Configuring LiveCycle for WebLogic*

[Back to top](#)

Improved Configuration Manager

LiveCycle Form Manager now includes an enhanced Adobe Configuration Manager. Using Configuration Manager, you can perform the following tasks:

- Configure and assemble LiveCycle products for deployment
- Configure your IBM® WebSphere® application server or BEA WebLogic Server® for LiveCycle products
- Validate application server settings for LiveCycle products
- Automatically deploy LiveCycle products to an application server
- Initialize database schemas for deployed LiveCycle products
- Verify deployed LiveCycle products are available and operational

[Back to top](#)

Adobe, the Adobe logo, and LiveCycle are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States and/or other countries.

BEA WebLogic Server is a registered trademark of BEA Systems, Inc.

IBM and WebSphere are trademarks of International Business Machines Corporation in the United States, other countries, or both.

All other trademarks are the property of their respective owners.