Deployment Example: Deploying and Customizing the Documentum Portlet

Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, CA 95054
U.S.A.

Part No: 820–2541
June 2007

070621@17466

Contents

Preface 5

1 About Documentum Portlets 9
Overview of Documentum Portlets 9
Sample Deployment Model 10

2 Pre-Installation Tasks 13
Pre-Installation Requirements 13
Software Requirements 13
Software Requirements for Documentum 13
Software Requirements for Portal Server 14

3 Environment Configuration 15
Setting Up Profile for Portal User 15
Set Application Context with Sun Java System Application Server 16
DFC 16
Setting Application Context 16
▼ Set Application Context 16

4 Deploying Documentum Portlets 17
Deploying the Documentum Portlets 17
▼ Configuring Portal Server 17
▼ Configuring Portlet Authentication and User Principal Authentication 18
▼ Configuring Web Publisher Portlet 20
▼ Known Issue for Web Publisher Portlet 20
▼ Configuring eRooms Portlets 21
▼ Configuring ECIS portlets 22

▼ Deploying the Portlets 22
▼ Configuring the Desktop 23

5 Integrating an LDAP Directory Server with a Repository 25
Overview of Integrating an LDAP Directory Server with a Repository 25

6 Understanding WDK 27
WDK Functionality and Architecture 27
Documentum Portlet 28
Introduction 28
Portlet Files 29

7 Creating and Customizing Portlets 31
Documentum Portlet Class 31
Portlet User interface 32
Portal Styles 33
Namespace in Portlets 33
Portlet Refresh 34
Adding Portlet Help 35

8 Integrating with Sun Java Portal Environment 7.1 37
Where to Integrate WDK into a Portal 37
Environment Classes 38
`AbstractEnvironment` Class 40
`Environment` Class 40
`AppServerEnvironment` Class 41
`PortalEnvironment` Class 42
`Jsr168Environment` Class 42
`SunPortalEnvironment` Class 43

9 Troubleshooting 45
Tracing 45

Preface

This guide details the tasks to be performed to successfully deploy Documentum portlets onto Sun Java System Portal Server 7.1 software configured with Sun Java System Application Server Platform Edition 8.2 software.

Who Should Use This Book

This manual is intended for the following audiences:

- Java developers who are developing custom JSP-based Web and portal applications that incorporate Documentum functionality
- Web page designers who are configuring WDK client applications. To configure WDK-based applications, you must be familiar with the following technologies:
  - JSP 1.1 including tag libraries
  - Cascading style sheets (CSS)
  - HTML, particularly forms, tables, and framesets JavaScript, including client events and event handling, frame referencing, and form
  - Action methods
  - XML You must also be familiar with the following additional languages and standards:
  - Java 1.3.1 or 1.4
  - Servlet 2.3
  - Sun JSR-168 standard for portlets
- System Administrators and Product Evaluators

How This Book Is Organized

This document describes how to configure Documentum Portlets. This section is tailored specifically for System Administrators and product evaluators. It also includes some customization examples and debugging information.

Related Books

The following references are available on the Documentum product download site:

- *Web Development Kit Development Guide* contains general configuration, customization, and application-building information for application developers.
- *Web Development Kit and Client Applications Reference Guide* contains information about all of the configurable settings for controls, actions, and components in WDK.
- *Web Development Kit and Applications Tutorial* contains several tutorials on configuring and customizing WDK.
- *Web Development Kit and Applications Installation Guide* describes how to prepare for, install, and deploy WDK and custom WDK-based applications.
- *WDK for Portlets Installation Guide* describes how to prepare for, install, and deploy WDK for Portlets.
- WDK and DFC javadocs.
- *Documentum Foundation Classes (DFC) 5.3 SP4 Release Notes*, *Documentum WDK for Portlets 5.3 SP4 Release Notes*, and *Documentum WebPublisher Portlets Release Notes 5.3 SP1 Release Notes* contains information on Fixed and New Bugs.

Check the Documentum technical support Web site for revisions of the documentation. Click the Documentation link to search for documents related to your installed version of WDK or WDK client application. The support site also provides peer support forums that are monitored by technical support experts.

The Documentum developer Web site, provides sample code, tips, white papers, and a wealth of information to assist you in developing Documentum-enabled applications.

Related Third-Party Web Site References

Third-party URLs are referenced in this document and provide additional, related information.

Note – Sun is not responsible for the availability of third-party web sites mentioned in this document. Sun does not endorse and is not responsible or liable for any content, advertising, products, or other materials that are available on or through such sites or resources. Sun will not be responsible or liable for any actual or alleged damage or loss caused or alleged to be caused by or in connection with use of or reliance on any such content, goods, or services that are available on or through such sites or resources.

Documentation, Support, and Training

The Sun web site provides information about the following additional resources:

- Documentation (`http://www.sun.com/documentation/`)
- Support (`http://www.sun.com/support/`)
- Training (`http://www.sun.com/training/`)

Typographic Conventions

The following table describes the typographic conventions that are used in this book.

TABLE P–1 Typographic Conventions

Typeface	Meaning	Example
`AaBbCc123`	The names of commands, files, and directories, and onscreen computer output	Edit your `.login` file. Use `ls -a` to list all files. `machine_name% you have mail.`
`AaBbCc123`	What you type, contrasted with onscreen computer output	`machine_name%` **`su`** `Password:`
aabbcc123	Placeholder: replace with a real name or value	The command to remove a file is `rm` *filename*.
AaBbCc123	Book titles, new terms, and terms to be emphasized	Read Chapter 6 in the *User's Guide*. A *cache* is a copy that is stored locally. Do *not* save the file. **Note:** Some emphasized items appear bold online.

Shell Prompts in Command Examples

The following table shows the default UNIX® system prompt and superuser prompt for the C shell, Bourne shell, and Korn shell.

TABLE P–2 Shell Prompts

Shell	Prompt
C shell	`machine_name%`
C shell for superuser	`machine_name#`
Bourne shell and Korn shell	`$`
Bourne shell and Korn shell for superuser	`#`

◆ ◆ ◆ CHAPTER 1

About Documentum Portlets

This chapter provides an overview of Documentum portlets.

Overview of Documentum Portlets

Documentum's Portlets allows Portal vendors to include a set of Portlets developed by Documentum in their Portal offering. These Documentum Portlets are built on the WDK 5 framework. WDK 5 is an extension to WDK 4.2 utilizing WDK Forms technology. WDK Forms are designed to provide a web application framework and library for the development of web-based UI components that expose Documentum's functionality and services.

The Documentum Portlets provide a set of content management services which meets the needs of a majority of users within an enterprise. These services are packaged as a set of out-of-the-box Portlets (Java classes and JSP pages) that can be configured and supported by Portal vendors. These Portlets expose a different area of Documentum functionality with an easy-to-use interface and a consistent UI.

The Documentum Portlets significantly reduce the need to create new programming interfaces and integration code for enterprise Portal implementations. Building on the enterprise content management strengths of Documentum 5i eBusiness Platform, these services employ Portlets to create a fast and cost-effective way to expose content management functionality through an enterprise Portal.

The Documentum Portlets combine off-the-shelf, embeddable Portal components, with documentation and integration examples that enable Portal vendors and customers to rapidly deploy content management services through Portals. This capability facilitates the convenient management and delivery of structured and unstructured content beyond the firewall. The Documentum Portlets allow developers to quickly transform enterprise Portals into dynamic hubs for content-driven business processes such as real-time contract negotiation, supply chain management, and global project management.

Sample Deployment Model

This section describes one of the deployment model for Documentum Portlets. Please refer to *Documentum Content Server Installation Guide* and *Documentum Content Server Administration Guide* for additional model supported by EMC Documentum.

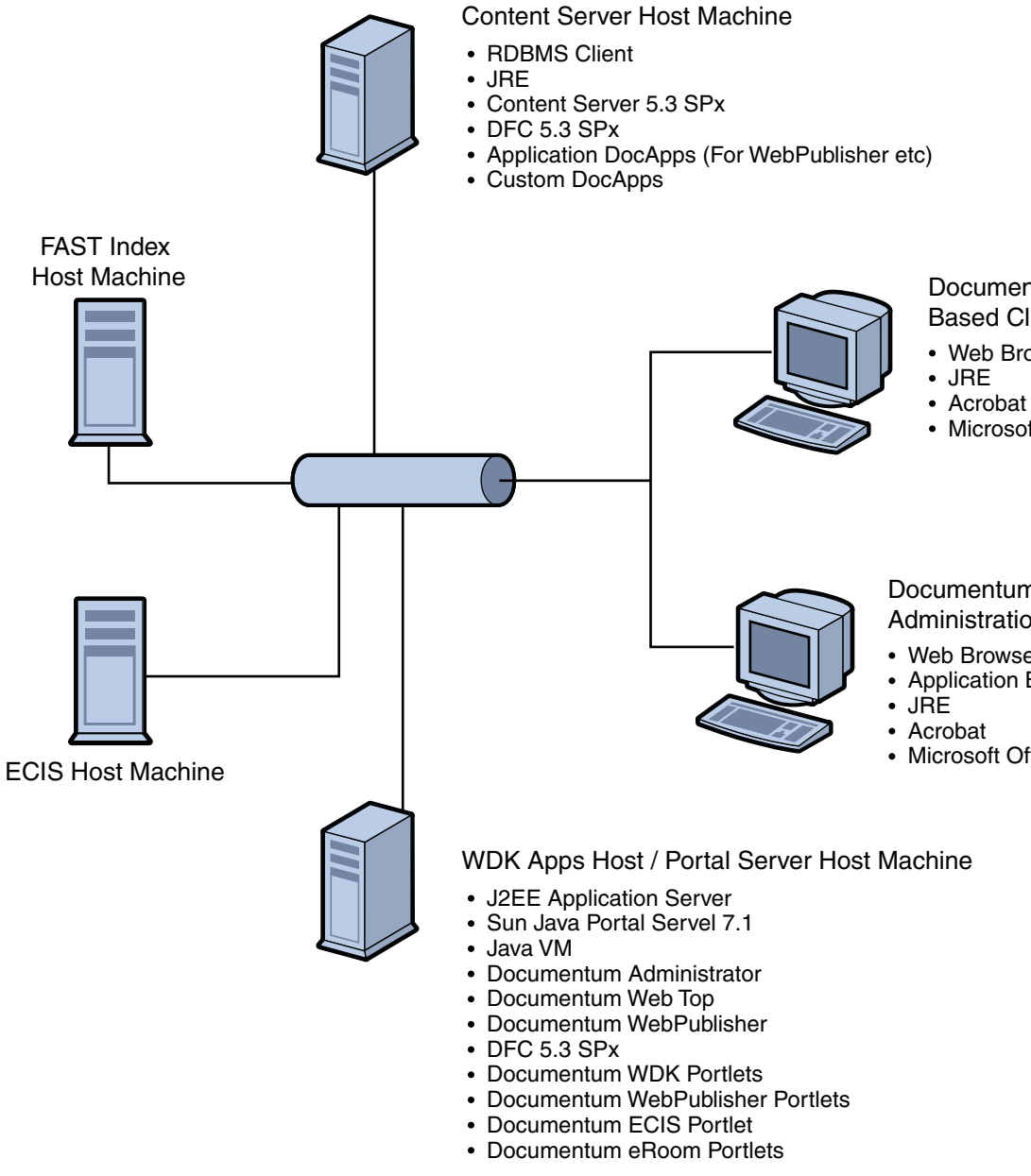

FIGURE 1–1 Deployment Model

Deploying Documentum portlets on Sun Java System Portal Server on Solaris involves 4 steps. First, download the portlets from EMC. Then, change the WAR files to include Sun Java System Portal Server related properties. Repackage the portlets, and lastly, deploy the portlets.

◆ ◆ ◆ CHAPTER 2

Pre-Installation Tasks

This chapter describes the installation requirements and pre-installation tasks for installing the Documentum Portlets with Sun Java System Portal Server 7.1 software configured with Sun Java System Application Server Platform Edition 8.2 software.

Pre-Installation Requirements

Install the Sun Java System Portal Server 7.1 software.

Sun Java System Portal Server software must be installed prior to deploying Documentum Portlets.

Note – Documentum Portlets are currently certified against Portal Server 7.1 configured with Sun Java System Application Server Platform Edition 8.2.

Software Requirements

This section describes the requirements to deploy Documentum portlets onto Sun Java System Portal Server.

- "Software Requirements for Documentum" on page 13
- "Software Requirements for Portal Server" on page 14

Software Requirements for Documentum

- Solaris 10 for SPARC
- Oracle database 9i or 10g
- Documentum Content Server for Solaris Oracle, Version 5.3 SP4 (SP3 can not be used due to a bug that is fixed in SP4)

- Documentum Webtop for Solaris, Version 5.3 SP4
- Documentum Administrator for Solaris, Version 5.3 SP4
- Documentum Web Publisher for Solaris, Version 5.3 SP4
- Documentum Web Publisher Server Files Installer for Solaris Version 5.3 SP4
- Documentum ECI Services, Version 5.3 SP4
- eRoom 7 for Windows

Software Requirements for Portal Server

- Solaris 9 and 10 for SPARC
- Sun Java Enterprise System 5 Sun Java System Portal Server 7.1 software
- Documentum Foundation Classes (DFC) for Solaris, Version 5.3 SP4

◆ ◆ ◆ CHAPTER 3

Environment Configuration

This chapter contains the following:

- "Setting Up Profile for Portal User" on page 15
- "Set Application Context with Sun Java System Application Server" on page 16

Setting Up Profile for Portal User

Add environment parameters into Portal Install user's (normally `root`) profile.

Here is an example of the environment parameters for a user using `ksh`:

```
DOCUMENTUM=<Documentum_Install_Dir>; export DOCUMENTUM;
DOCUMENTUM_SHARED=<Documentum_Share_Dir>; export DOCUMENTUM_SHARED;
DM_HOME=$DOCUMENTUM/product/5.3; export DM_HOME;
DMCL_CONFIG=$DOCUMENTUM/dmcl.ini; export DMCL_CONFIG;
JAVA_HOME=<JDK_1.4.x>; export JAVA_HOME;
PATH=$PATH:/usr/openwin/bin:$JAVA_HOME/bin:$DM_HOME/bin:$DOCUMENTUM/dba;
export PATH
CLASSPATH=$CLASSPATH:$DOCUMENTUM_SHARED/dctm.jar:$DOCUMENTUM_SHARED/config:
$DOCUMENTUM_SHARED/dfc/dfc.jar:$DM_HOME/bin:.;
export CLASSPATH
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$DOCUMENTUM_SHARED/dfc:$DM_HOME/bin:
$DOCUMENTUM/fulltext/fast40:$DM_HOME/fusion:$DOCUMENTUM/fulltext/IndexServer/lib:
$DOCUMENTUM/share/clients/unix/solaris;
export LD_LIBRARY_PATH;
```

Set Application Context with Sun Java System Application Server

This section contains the following:

- "DFC" on page 16
- "Setting Application Context" on page 16

DFC

DFC 5.3 SP4 must be installed on Portal Server machine to set required DFC environment. For UNIX systems, it assumes general familiarity with shells, permissions, and environment variables. Please follow *Documentum DFC 5.3 SP3 Installation Guide* for instructions on how to install DFC.

Setting Application Context

This sections contains instructions to "Set Application Context" on page 16.

▼ Set Application Context

1 **Copy DFC JAR files into application server's `lib` directory.**

For example, type **`cp $DOCUMENTUM/dfc/*.jar <AppServer_Base_Dir>/appserver/lib`**.

2 **Set Application Server's classpath and add the classpath to application server instances.**

For example, type the following:

```
<App_Server_Dir>/domains/domain1/config/domain.xml
<java-config ... classpath-suffix="$DOCUMENTUM/config: ...>
```

3 **Set the `enablePooling` parameter.**

In the `<App_Server_Dir>/domains/domain1/config/default-web.xml` file, in the Default Servlet section, add the following:

```
<servlet>
<init-param>
<param-name>enablePooling</param-name>
<param-value>false</param-value>
</init-param>
</servlet>
```

◆ ◆ ◆ CHAPTER 4

Deploying Documentum Portlets

This chapter contains instructions for "Deploying the Documentum Portlets" on page 17 on Sun Java System Portal Server on Solaris.

Deploying the Documentum Portlets

This section contains the following:

- "Configuring Portal Server" on page 17
- "Configuring Portlet Authentication and User Principal Authentication" on page 18
- "Configuring Web Publisher Portlet" on page 20
- "Known Issue for Web Publisher Portlet" on page 20
- "Configuring eRooms Portlets" on page 21
- "Configuring ECIS portlets" on page 22
- "Deploying the Portlets" on page 22
- "Configuring the Desktop" on page 23

▼ Configuring Portal Server

Perform the steps in this section on Portal Server.

1 Download the portal WAR file from EMC download site.

You must download the following portlets. To download, , login to `https://emc.subscribenet.com/` using your DCTM download credentials. On the left hand menu, click on `Product Search` and enter the following phrases each time to search for the corresponding portlets.

- Documentum Web Development Kit for Portlets for Solaris, Version 5.3 SP4
- Documentum Web Publisher Portlets for Solaris, Version 5.3 SP1
- Documentum ECI Portlet, Version 5.3

- Documentum eRoom (Version 7) Portlets

2 **Create a directory and save the Web Development Kit for Portlets WAR file in this directory.**

3 **Expand the WAR file.**

Use the `jar` command from JDK1.4.x. For example, type **`jar -xvf wdkforportlets.war`**.

4 **Change** `ServletContextKeywords_sunportal` **in the** `WEB-INF/classes/com/documentum/web/env/EnvironmentService.properties` **file to include Sun-Java-System/Application-Server.**

For example:

```
ServletContextKeywords_sunportal=Sun-Java-System/Application-Server
```

5 **Change the** `PortalDescription` **and** `PortalVersion` **values in the** `WEB-INF/classes/com/documentum/web/env/sun/SunPortalEnvironment.properties` **file.**

For example:

```
PortalDescription=Sun Java System Portal Server
PortalVersion=7.0
```

6 **From the top directory (created in Step 2) that contains the WAR file, create the WAR file that's ready for deployment.**

For example, type:

```
rm wdkforportlets.war
jar -cvf wdkforportlets.war *
```

Note – If you plan to configure authentication, perform this step after configuring authentication.

▼ Configuring Portlet Authentication and User Principal Authentication

The following configuration is for Web Development Kit Portlets and Web Publisher Portlets.

1 **Expand the WAR file using the** `jar` **command from JDK1.4.x.**

For Web Development Kit Portlets, type **`jar -xvf wdkforportlets.war`** and for Web Publisher Portlets, type **`jar -xvf wpportlets.war`**.

2 **Ensure that the** `UserPrincipalAuthenticationScheme` **is above the** `DocbaseLoginAuthenticationScheme` **in the** `AuthenticationSchemes.properties` **file in** `WEB-INF/classes/com/documentum/web/formext/session/` **directory.**

3. **Open the** `WEB-INF/classes/com/documentum/web/env/sun/SunPortalEnvironment.properties` **file, add the following line, and save the file:**

   ```
   AuthenticationMode=trusted
   ```

4. **Encrypt the superuser's password and paste the encrypted form of the password into** `WEB-INF/classes/com/documentum/web/formext/session/TrustedAuthenticatorCredentials.pr` **file.**

   You can encrypt the Superuser's password for the repository with the trusted authenticator tool (`com.documentum.web.formext.session.TrustedAuthenticatorTool`).

   a. **From the command line, run the following command.**

      Substitute the actual repository password to be encrypted:

      ```
      cd <war_file_director>/WEB-INF/classes
      java TrustedAuthenticatorTool password
      ```

      The output will look similar to the following:

      ```
      Encrypted: [d7d1d6e383d6d4e1d0], Decrypted: [my.pwd6\]
      ```

   b. **Paste the encrypted form of the password into** `WEB-INF/classes/com/documentum/web/formext/session/TrustedAuthenticatorCredentials` **file.**

      Each repository must have three entries (substitute the actual repository name in the sample entries below):

      Repository_name`.user`
      Repository_name`.password`
      Repository_name`.domain`

      If no domain, then enter the following:

      Repository_name`.domain=`

      For example:

      ```
      myrepository.user=ecmdocbase
      myrepository.password=d7d1d6e383d6d4e1d0
      myrepository.domain=
      ```

5. **Add** `docbase` **name in** `<war_file_directory>/wdk/app.xml` **file.**

   For example:

```
<authentication>
    <!-- Default domain and docbase to authenticate against -->
    <domain></domain>
    <docbase>ecmdocbase</docbase>
```

```
    <!-- Class that provide the authentication service -->
    <service_class>com.documentum.web.formext.session.AuthenticationService</service_class>
    <!-- Single Sign-On authentication scheme configuration -->
    <sso_config>
        <ecs_plug_in></ecs_plug_in>
        <ticket_cookie></ticket_cookie>
        <user_header></user_header>
    </sso_config>
</authentication>
```

Configuring Web Publisher Portlet

1. **Create a directory, save the WAR file into this directory, and expand the WAR file.**

   Use the `jar` command from JDK1.4.x. For example, type **`jar -xvf wpportlets.war`**.

2. **Open `wpportlets/web/wpportlets/app.xml` file and change the following configuration to point to Web publisher application.**

   ```
   <wpurl>http://<web_publisher_server_hostname>:<portnumber>/wp</wpurl>
   ```

3. **From the top directory that contains the WAR file, create the WAR file that's ready for deployment.**

   For example, type:

   ```
   rm wpportlets.war
   jar -cvf wpportlets.war *
   ```

Known Issue for Web Publisher Portlet

It has been observed that single-sign-on does not work for non Documentum admin users once Web Publisher Portlet is deployed on the Sun Java System Portal Server. In addition to authentication changes described in "Configuring Portlet Authentication and User Principal Authentication" on page 18, perform the following steps.

1. **Download the `WPPortlets_BUG.zip` file and unzip it.**

   Note – Please open a Sun Support (`http://www.sun.com/support/`) case or ticket to obtain the `WPPortlets_BUG.zip` file.

2. **In the machine where you have deployed your WPP portlets, make a backup copy of your current `web.xml` file in the `/var/opt/SUNWappserver/domains/domain1/applications/j2ee-modules/wpportlets/WEB-INF/` directory and replace it with the `web.xml` from the `WPPortlets_BUG.zip` file.**

Note – If you made changes to the `web.xml` file in the `/var/opt/SUNWappserver/domains/domain1/applications/j2ee-modules/wpportlets/WEB-INF/` directory, you have to re-do it after copying the `web.xml` from the `WPPortlets_BUG.zip` file.

3 **In the machine where you have deployed your WPP portelets, make a backup copy of `TicketedRedirect.class` in the `/var/opt/SUNWappserver/domains/domain1/applications/j2ee-modules/wpportlets/WEB-INF/c` directory and replace it with `TicketedRedirect.class` file from the `WPPortlets_BUG.zip` file.**

4 **Restart the Sun Java System Application Server.**

5 **Test WPP portlets by logging as an user who is not the DCTM Admin user.**

▼ Configuring eRooms Portlets

1 **To configure `Log4j`, open `$DOCUMENTUM/config/log4j.properties` file and add the following lines into `log4j.properties` file.**

```
# Enable log messages from eRoom Portlets
log4j.logger.com.documentum.eroom=DEBUG,eRoomAppender
#eRoom Portlets Appender
log4j.appender.eRoomAppender=org.apache.log4j.RollingFileAppender
log4j.appender.eRoomAppender.File=<$Documentum>/logs/eRoomPortlets.log
log4j.appender.eRoomAppender.MaxFileSize=500KB
log4j.appender.eRoomAppender.layout=org.apache.log4j.PatternLayout
log4j.appender.eRoomAppender.layout.ConversionPattern=%d{HH:mm:ss} %p %c %m %n
```

2 **Create a directory, save the WAR file into this directory, and expand the WAR file.**

Use the `jar` command from JDK1.4.x. For example, type **`jar -xvf MyeRoom.war`**.

3 **Open `WEB-INF/classes/com/documentum/eroom/utils/eRoom.properties` file and change the path in `portlet.eroom.server.path`.**

For example, modify `portlet.eroom.server.path=http://<eroom_server_hostname>/`.

4 **From the top directory that contains the WAR file, create the WAR file that's ready for deployment.**

For example, type the following:

```
rm MyeRoom.war
jar -cvf MyeRoom.war *
```

Note – Refer to *eRoom 7 Installation, Upgrade, and Configuration Guide* for installing and configuring eRoom Server.

Configuring ECIS portlets

1. **Open** `$DOCUMENTUM/config/dfc.properties` **file and add the following lines:**

```
dfc.search.ecis.enable=true
dfc.search.ecis.host=<eci_server_hostname>
dfc.search.ecis.port=<eci_server_RMI_portnumber_default_3005>
dfc.search.ecis.login=<username_defautl_guest>
dfc.search.ecis.password=<password_defautl_guest_password_askonce>
```

2. **Create a directory, save the WAR file into this directory, and expand the WAR file.**

   Use the `jar` command from JDK1.4.x: For example, type **`jar -xvf eciPortlet.war`**.

3. **Open** `WEB-INF/web.xml` **file and change** `RMI_REGISTRY_HOST` **and** `RMI_REGISTRY_PORT`**.**

```
<context-param>
    <param-name>RMI_REGISTRY_HOST</param-name>
    <param-value>eci_server_hostname</param-value>
    <description>The name of the RMI server</description>
</context-param>
<context-param>
    <param-name>RMI_REGISTRY_PORT</param-name>
    <param-value>eci_server_RMI_portnumber</param-value>
    <description>The port for the RMI server</description>
</context-param>
```

4. **Copy the** `WEB-INF/lib/dlcc.jar` **file to** `<AppServer_Base_Dir>/appserver/lib` **directory.**

5. **Copy the following configuration file from ECI_Server to portal server.**

```
$DOCUMENTUM/eci/www/docs/conf/client.conf
```

6. **From the top directory that contains the WAR file, create the WAR file that's ready for deployment.**

   For example, type the following:

```
rm eciPortlet.war
jar -cvf eciPortlet.war *
```

Deploying the Portlets

Deploy each WAR file onto Portal Server.

1. **Create a password file that contains portal user `amadmin`'s password in `/tmp/password` or any other directory.**

2. **Go to directory where the Documentum Portlets WAR files are located.**

   a. **Deploy WDK Portlets.**

      Type **`<PortalServer_Base_Dir>/bin/psadmin deploy-portlet -u amadmin -f /tmp/password -p portal1 -g wdkforportlets.war`**

   b. **Deploy Web Publisher Portlets.**

      Type **`<PortalServer_Base_Dir>/bin/psadmin deploy-portlet -u amadmin -f /tmp/password -p portal1 -g wpportlets.war`**

   c. **Deploy ECI Server Portlets.**

      Type **`<PortalServer_Base_Dir>/bin/psadmin deploy-portlet -u amadmin -f /tmp/password -p portal1 -g eciPortlet.war`**

   d. **Deploy eRoom Portlets.**

      Type **`<PortalServer_Base_Dir>/bin/psadmin deploy-portlet -u amadmin -f /tmp/password -p portal1 -g MyeRoom.war`**

3. **To Undeploy, for example, type `/opt/SUNWportal/bin/psadmin undeploy-portlet -u amadmin -f /opt/SUNWportal/bin/pass -p portal1 -g wdkforportlets.war`**

▼ Configuring the Desktop

1. **Create JSPTableContainer for each WAR file.**

   Due to parallel execution of all portlets in the above layout, sometimes a generic message `Content not available` is displayed in the portlets. This is due to some synchronization issues. To avoid this problem, turn off parallel execution of portlets by changing `portletRenderModeParallel=false` to `portletRenderModeParallel=true` in `<PortalServer_Instance_Dir>/portals/portal1/config/desktopconfig.properties` file. Please repeat this process for all portals and restart the application server.

2. **Add channel onto appropriate container.**

◆ ◆ ◆ CHAPTER 5

Integrating an LDAP Directory Server with a Repository

This chapter provides information on integrating an LDAP directory Server with a repository.

Overview of Integrating an LDAP Directory Server with a Repository

When you add an LDAP directory server to an existing Documentum installation, the users and groups defined in the LDAP directory server are given precedence. If a user or group entry in the LDAP directory server matches a user or group in the repository, the repository information is overwritten by the information in the LDAP directory server when the `dm_LDAPSynchronization` job executes.

All users in a repository must have three required attributes set. The attributes are `user_name`, `user_login_name`, and `user_address`. Users defined in the LDAP directory server must have LDAP attributes that can be mapped to these attributes. The LDAP configuration set up provides mapped defaults for the required repository attributes or you can define custom mappings. Refer to the Documentum Administrator online help for instructions.

If your LDAP directory server is Active Directory, `user_name` must be mapped to either a common name (`CN`) or a display name. If `user_name` is not mapped to either a `CN` or display name, the repository inactivation operation does not function properly for the users.

Please refer to *Documentum Content Server 5.3 SP1 Administrating Guide* to implement LDAP Directory Server.

◆ ◆ ◆ CHAPTER 6

Understanding WDK

The Documentum Web Development Kit (WDK) is a Web application tool set.

This introduction to WDK includes the following topics:

- "WDK Functionality and Architecture" on page 27
- "Documentum Portlet" on page 28

Note – The audience from this chapter onwards are developers who may write new portlets or customize the existing ones.

WDK Functionality and Architecture

The Documentum Web Development Kit (WDK) is a Web application tool set.

WDK provides the following functionality:

- A Java tag library of easily configured Web-based UI widgets
- A Java framework that supports application-server based state management, messaging, branding, history, internationalization, and content transfer
- A set of configurable components that generate HTML widgets and provide access to repository functionality

WDK's architecture incorporates three models: A presentation model that uses JSP tag libraries to separate Web page design from behavior, a component model that encapsulates repository functionality in configurable server-side components, and an application model that consists of a set of components and the WDK application framework, DFC, and native libraries.

In the diagram below of a Documentum portal application, an environment layer is shown. This layer is conceptual only and does not represent an actual application layer like wdk or

webcomponent. The environment layer consists of all of the parts of a Web application that differ depending on whether the application is a standalone Web application or a portal application.

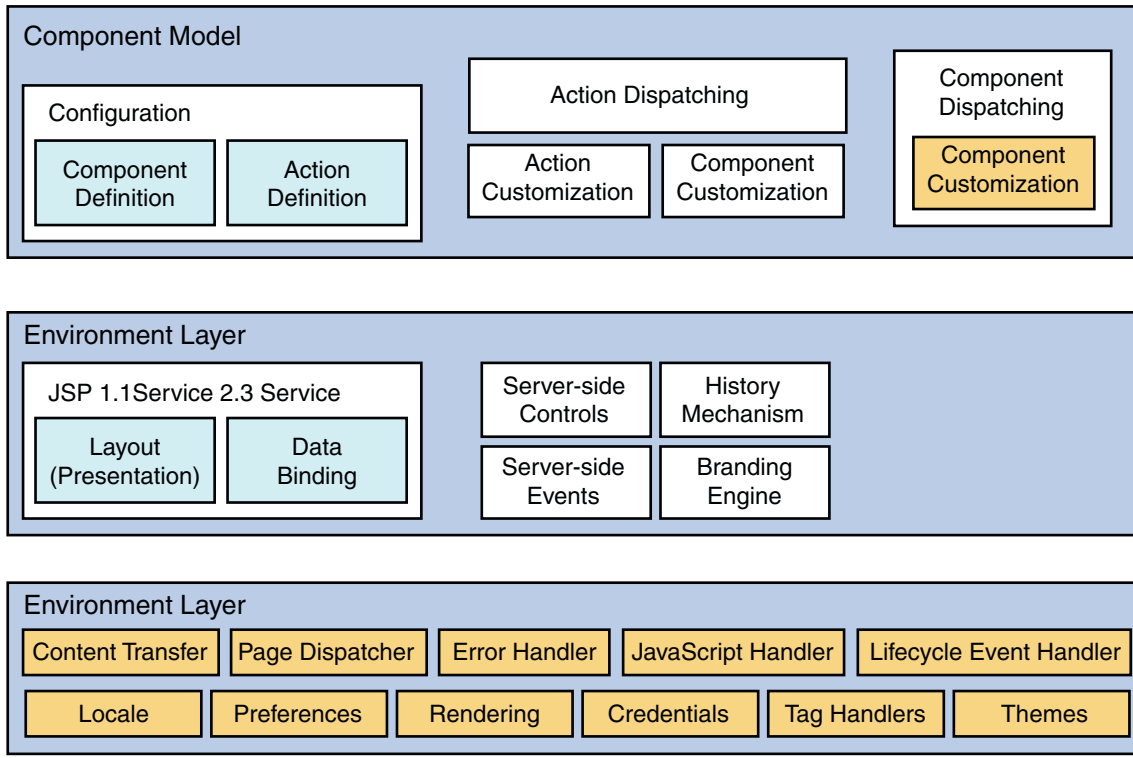

FIGURE 6–1 WDK for Portlets Architecture

Documentum Portlet

This section contains the following:

- "Introduction" on page 28
- "Portlet Files" on page 29

Introduction

Based on the Sun portlet specification JSR-168, a portlet is a Java-based web component, managed by a portlet container, that processes requests and generates dynamic content. Portlets are used by portals as pluggable user interface components that provide a presentation layer to Information Systems.

WDK for Portlets installs Documentum portlets, WDK run time, and WDK components into a portal Web application in a certified portal server. A single portlet class, `DocumentumComponent`, dispatches WDK components for each portlet. A portlet that dispatches a WDK component must specify its portlet class as `com.documentum.web.env.jsr168.DocumentumComponent`.

The components that are dispatched for portlet view and edit modes are specified for each portlet in the `portlet.xml` file. The context-sensitive portlet help file is dispatched to a new browser window based on the value of the help mode in the `portlet.xml` file.

Caution – If your IDE creates portlets by creating the portlet class, make sure you do not create a Documentum portlet specifying the portlet class as `DocumentumComponent`. The IDE will overwrite the Documentum portlet class and render all Documentum portlets invalid. If this happens, you can restore the portlet class from your installation WAR file.

Portlet Files

WDK for Portlets installs the following portlets and supporting files:

`/custom`	Top Documentum portlet application layer. Use this directory for your custom portlet files
`/help`	Contains localized help files for portlets
`/portlets`	Contains portal-specific files as required by the portal vendor
`/wdk`	Contains WDK components and supporting files
`/webcomponent`	Extends the `/wdk` layer and defines more components
`/WEB-INF/portlet.xml`	Contains definitions for all JSR-168 compliant portlets including Documentum portlets
`/WEB-INF/classes`	Contains Documentum and custom classes for portlet components
`/WEB-INF/lib`	Contains Documentum JAR files
`/WEB-INF/tlds`	Contains Documentum tag libraries
`/DOCUMENTUM_HOME`	DFC is installed to this directory. The directory is selected during installation. If Documentum products have already been installed on the host, the pre-existing home directory is used.

◆ ◆ ◆ CHAPTER 7

Creating and Customizing Portlets

If you are writing portlets to run within a JSR-168 compliant portal application server, you need to follow the guidelines outlined in the following topics:

- "Documentum Portlet Class" on page 31
- "Portlet User interface" on page 32
- "Portal Styles" on page 33
- "Namespace in Portlets" on page 33
- "Portlet Refresh" on page 34
- "Adding Portlet Help" on page 35

For information on adding preferences to a portlet, refer to Chapter 3. For information on how to add help for a portlet, refer to Adding portlet help.

Documentum Portlet Class

Documentum portlets use a single portlet class, `com.documentum.web.env.jsr168.DocumentumComponent` which extends `javax.portlet.GenericPortlet`. This class will launch the component that you specify for the portlet as the value of the view mode in the `portlet.xml` file and will also provide access to preferences and help for the component using the preferences and help mode values in the `portlet.xml` file.

Caution – If your IDE creates portlets by creating the portlet class, make sure you do not create a Documentum portlet within the IDE by specifying the portlet class as `DocumentumComponent`. The IDE will overwrite the Documentum portlet class and render all Documentum portlets invalid. If this happens, you can restore the portlet class from your installation WAR file.

Portlet User interface

JSP pages in portlets should contain HTML fragments only. The portlet contributes its content to a larger page, so it cannot be a fully formed HTML page. Use the tags `<dmf:html>`, `<dmf:head>`, and `<dmf:body>` instead of `<HTML>`, `<HEAD>`, `<TITLE>`, or `<BODY>` elements. These tags will be rendered appropriately for portal environments. They enable icon rendition and error handling, so they should not be omitted from JSP pages that are used in portlets.

Your portlet content will probably be contained within a table cell (`<TD>`) on the portal page. You can use default width and height and add width and height preferences to your portlet XML configuration. The portlet will then use scroll bars if the content is too large for the available table space. Avoid unnecessary layout elements to reduce the amount of scrolling needed for your portlet.

Tip – Avoid placing `<nobr>` tags around table cells to prevent line breaks. This will cause your portlet to push the portal page wide and may force excessive scrolling when users view other portlets. To allow portal users with disabilities to be able to use your portlet, the JSPs should be fully enabled for keyboard-only control and other assistive technologies. For guidelines on making your portlet components accessible, refer to Web Development Kit and Client Applications Reference Guide.

In general, you should minimize the use of iFrames and JavaScript in your portlets, because support differs between browsers and versions. Do not use popup browser instances, which will cause the portal to lose track of the current portlet's state and history. Some portals provide a widget on the titlebar that allows the user to pop the portlet up as a new browser window.

Use Java comments rather than HTML comments (`<! - comment ->`) in your files. HTML comments are rendered into the output even though they are not visible, increasing the document size and download time. Java comments similar to the following example are removed from the rendered source along with Java code:

```
<% //this is a comment %>
```

Tip – Give each control in your portlet components a unique name, because they will be rendered by the portal container into a single HTML page.

Portal Styles

Your portlet should use portal style classes that are defined in the portal's cascading style sheet. Consult your portal documentation for instructions on setting portlet styles. In the following example from a portlet JSP page, an input element uses a portal style class called `portlet-form-input-field`:

```
<input class="portlet-form-input-field" type="submit">
```

The `.css` file is loaded by the portal and should not be reloaded by a portlet. For class definitions and associated values, refer to the JSR-168 specification. This interface converts a WDK standard CSS style into one of the portal styles. By default, this interface returns the WDK style unchanged. Implement the method `mapCssClass()` to perform the mapping. This method has the following signature:

```
public String mapCssClass (String strCssClass)
```

Where `strCssClass` is the WDK CSS class name. Returns the mapped portal style. You can add a branding image to brand your portlets, replacing the Documentum logo image in the lower right-hand corner of the portlet rendition. To do this, locate the path to an image within your portal directories and enter it as the value of the `BrandingImage` key in the `Environment.properties` file for your portal. For example, the file for the SUN portal is named `SunPortalEnvironment.properties`, and it is located in `/WEB-INF/classes/com/documentum/web/env/sun` directory. A custom image placed in `/custom/images` is registered as follows:

```
BrandingImage=/custom/logos/myproduct.gif
```

This image is rendered after the portal is restarted, similar to the following. The image launches the About component which can be customized to describe your application.

Namespace in Portlets

The JSR-168 specification for portal servers specifies that the URIs, element name attributes, and JavaScipt methods should be encoded with the namespace of that portlet. Use the `IRender` method `namespaceElement(strPortletInstanceName)` to encode these elements. In the following example, a form element is encoded for portal `PC_202`:

```
<% IRender render = EnvironmentService.getEnvironment().getRenderContract();%>
<FORM method=POST name="<%=render.namespaceElement("form1")%>" action="">
```

The form element will then be rendered as follows:

```
<FORM method=POST name="PC_202_form1" action="">
```

To encode JavaScript with the portal namespace, WDK provides a servlet and VirtualJS. This servlet rewrites static JavaScript files to portlet-specific versions, changing method names and any form variable names that are used to the appropriate portal-specific namespace. The servlet is used by tag rewrite methods in a portal environment, so that the tags no longer point to static JavaScript files but to dynamic versions hosted by the servlet. For example, the following JavaScript declaration in a WDK page:

```
<script src="/wdk/wdk/include/events.js" language="javascript1.2">
</script>
```

is rewritten to:

```
<script src="/wdk/virtualjs/PC_202/wdk/include/events.js"
language="javascript1.2">
</script>
```

The servlet delegates the rewriting of JavaScript files to Java classes that implement the `IJavaScriptHandler` interface. JavaScript files that must be rewritten are registered for a particular handler class by adding entries into the `VirtualJS.properties` file located in `/WEB-INF/classes/com/documentum/web/env` directory.

Portlet Refresh

You can enable a refresh link in your portlet UI by implementing an `onRefreshData()` method in your portlet component class. You must then register your portlet in `PortalEnvironment.properties` file located in `WEB-INF/classes/com/documentum/web/env` directory

A refresh link will be generated for your portlet.

Note – The inherited `onRefreshData()` method is not sufficient to generate a Refresh link. Your component class must explicitly override `onRefreshData()`.

EXAMPLE 7–1 Refreshing data in a portlet component

The following example from the `MyObjects` class refreshes data when the user clicks on the Refresh link.

```
public void onRefreshData()
{
    super.onRefreshData();
    readConfig();
    updateControl();
```

EXAMPLE 7–1 Refreshing data in a portlet component *(Continued)*

```
    // force refresh on dataproviders
    ((Datagrid)getControl(
    MYOBJECTS_GRID, Datagrid.class)).getDataProvider().refresh();
}
```

Adding Portlet Help

You can amend the help that is installed by WDK for Portlets in your portal application and add new help files for your custom portlet components.

When you add a new portlet to `portlet.xml` file, add a value for the help mode preference. For example, for the portlet `DocumentumDrilldown`, the help mode is set to the value cabinets:

```
<preference>
    <name>help</name>
    <value>cabinets</value>
</preference>
```

This help value maps to an HTML file that must be located in */app_root_directory/language_code/*`cabinets` directory, for example, */PORTAL_APP_HOME/*`help/en/DocumentumPortlets/cabinets/default.htm`.

◆ ◆ ◆ CHAPTER 8

Integrating with Sun Java Portal Environment 7.1

The WDK environment package contains APIs that enable WDK to operate on J2EE application servers, JSR-168 compliant portal servers, or non-JSR-168 compliant portal servers. Environments are provided for J2EE standalone Web applications and for several JSR-168 compliant portal servers.

For certified J2EE application servers, refer to Web Development Kit Release Notes. For certified portal servers, refer to WDK for Portlets Release Notes.

The following topics describe the configurable and customizable environments in WDK:

Where to Integrate WDK into a Portal

If you are using a portal that is listed in the certification table of the WDK for Portlets release notes, your portal environment is provided by the installer and you can simply use Documentum portlets and components in your application after running the installer.

If you are using a JSR-168 compliant portal that does not have an environment provided by Documentum, you must extend the `Jsr168Environment` class and provide methods that write user preferences using the portal-specific APIs. For an example, refer to Writing user-level preferences. If you are using a portal that is not JSR-168 compliant, you must extend the `PortalEnvironment` class and provide methods that both read and write user preferences. For examples, refer to Writing user-level preferences and Reading user-level preferences

Environment Classes

The environment layer is a set of classes in the `com.documentum.web.env` package that comprise base implementations, hooks and interfaces. The Environment class encapsulates an environment's specific behavior and functionality. WDK instantiates and uses the registered environment class whenever it requires information from its environment.

A portal-specific environment is implemented as a single Java class that extends the abstract Environment class and implements one or more of the `com.documentum.web.env` interfaces. WDK environments are packaged in `com.documentum.web.env.xxx` where `xxx` is the name of the environment; for example, `com.documentum.web.env.sun`.

WDK instantiates an environment as a singleton object, and all threads call into the single instance. All methods implemented for a custom environment must be thread-safe. If you need to store state, the base Environment class provides instance methods to store and retrieve your attributes in the environment implementation.

The environment layer of WDK consists of a set of classes that provide a number of starting places for your integration depending on the type of portal server you have. The environment APIs are shown in the diagram below.

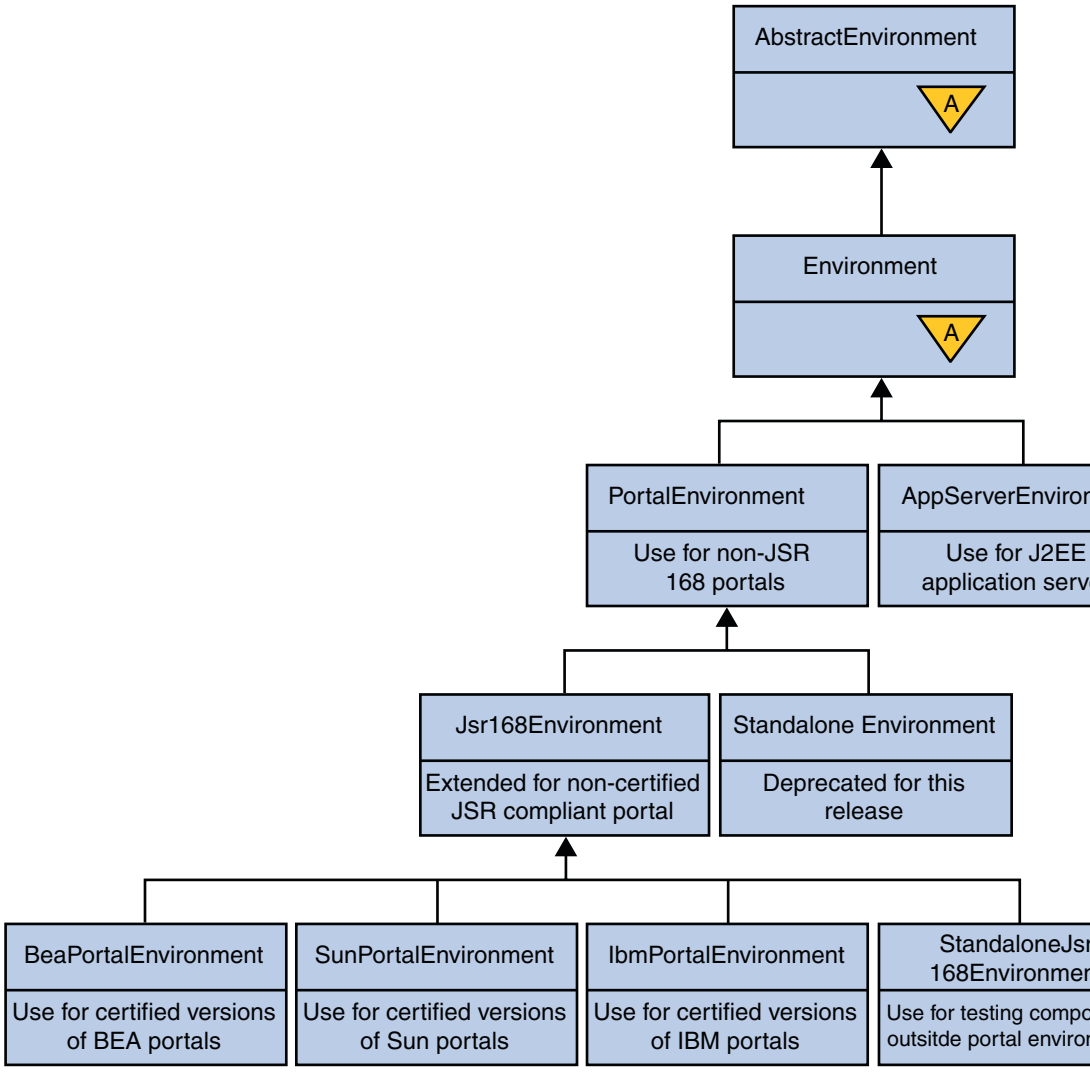

FIGURE 8–1 Environment APIs

The environment classes are the following:

`AbstractEnvironment` class	The `AbstractEnvironment` class is abstract and provides default implementations of environment APIs.
`Environment` class	The Environment class is also abstract. It extends `AbstractEnvironment` and adds default implementations for session and request access, dispatching, encryption, and portal API access.

Note – Many of these APIs must be overridden for specific portals.

`AppServerEnvironment` class	The `AppServerEnvironment` Class provides a configured environment for integrating with J2EE Application Servers.
`PortalEnvironment` class	The `PortalEnvironment` class provides a configured environment for non-JSR-168 compliant portal servers. If you are integrating with a non-JSR-168 compliant Portal Server then you should extend the `PortalEnvironment` class.
`Jsr168Environment` class	The `Jsr168Environment` class provides a configured environment for integrating with JSR-168 compliant portal servers.
`SunPortalEnvironment` class	The `SunPortalEnvironment` Class integrates with the Sun Portal Server. (For specific versions, refer to the release notes.)

`AbstractEnvironment` Class

The `AbstractEnvironment` class provides a set of methods that return the environment interface contracts. The `AbstractEnvironment` class returns whether the environment is a Portal environment using the `isPortalEnvironment()` method. The current `HttpServletRequest`, `HttpServletResponse`, and `HttpSession` are returned from the `getRequest()`, `getResponse()`, and `getSession()` methods.

`Environment` Class

The `Environment` class is also an abstract class and cannot be instantiated directly. The Environment class provides default implementations for the lifecycle events, security, error handling, page dispatching, action processing, content transfer, branding, environment properties, and attribute storage. These implementations should be sufficient for most environments:

Lifecycle	The lifecycle methods store the request, response and session objects in thread safe storage. If you are overriding any of these methods and you need to continue to store the objects in the thread safe storage, then you must call the superclass implementation.

Security	The security methods encode and decode passwords and encrypt and decrypt the trusted authenticator password using a Caesar cipher. For details on trusted authentication, refer to User principal authentication. Methods also get the user name, password, and domain.
Error Handling	The error methods get the error URL and handle errors.
Page Dispatching	Page dispatching methods include forward and inline dispatch and access to the object. By default, pages are included and actions are processed before the page is rendered.
Action Processing	The `queryExecute()` method is provided to test action preconditions
Content Transfer Mechanism	The content transfer method returns the content transfer mechanism, which is UCF-based by default in portal environments.
Branding	The Environment class looks for mapping of WDK CSS classes to portal styles. For details on this mapping, refer to Portal styles
Environment Properties	The `getResourceBundle()` method returns the properties file that configures the environment.
Attribute Storage	Attribute storage methods get, set, and remove attributes from the environment's thread-safe local store. A `log()` method is also provided.

`AppServerEnvironment` Class

The `AppServerEnvironment` class inherits from Environment. The `AppServerEnvironment` class is used by WDK for non-portal J2EE applications such as Webtop. This class includes all of the implementations for running on a J2EE environment.

For backwards compatibility with previous versions of WDK, the following implementations are in the `AppServerEnvironment` class:

- The `encodeUrl()` method returns the full path to the JSP page that is to receive the next post-server event.
- The implementations of the `IPreference` interface hook into the WDK HTTP cookie store used in previous versions of WDK.

`PortalEnvironment` Class

The `PortalEnvironment` class inherits from Environment and serves as the superclass for all portal environment implementations. You can extend this class to support WDK portlets on a non-JSR-168 compliant portal server.

The following methods are implemented:

`preprocess()`	Turns on pre-processing.
`encodeUrl()`	Removes the action URL in order to force all server events to be posted through the component dispatcher.
`handleError()` and `errorAcknowledged()`	Report errors to the WDK's error message service and handles the user event that is generated when the user acknowledges an error dialog.
`postProcess()`	Turns on post-processing.

`Jsr168Environment` Class

The `Jsr168Environment` class inherits from `PortalEnvironment`. You can extend `Jsr168Environment` to integrate WDK portlets into a JSR-168 compliant portal server that does not have an environment supplied by Documentum. The following methods are implemented:

Lifecycle events	The `onRequestStart()` and `onRequestFinish()` methods provide access to the request events
Session access	The `getUsername()`, `getUserPassword()`, `getDomain()`, `getDocbaseName()`, `getUserPrincipal()`, and `getLocale()` methods provide access to the user's portal session.
Pre-processing	The `preProcess()` method turns on pre-processing.
URL encoding	The `encodeUrl()` method returns a JSR-168 action URL.
JavaScript rewriting	The `namespaceJsMethodName()` method changes the supplied JavaScript function name to a name unique within the generated web page by prefixing that function with the portlet ID. The `rewriteScriptTag()` method prefixes the portlet ID to a supplied JavaScript function name.
Link rewriting methods	The `rewriteATag()` method rewrites the `href` and `onclick` attributes in a JavaScript call to prefix the call with the portlet ID. This gives the JavaScript call a unique name in the scope of the whole web page that is generated.

Preferences methods	Several methods enumerate preferences and look up or write preference values (user-level or portlet-level, or application-level). The `Jsr168Environment` class handles the writing of Portlet level preferences through the `writePortletLevelPreference()` method. The writing of user level preferences is not covered by the JSR-168 specification: They are written by vendor-specific environment classes.
Error methods	The `getErrorUrl()` method handles errors in a JSR-168 environment, passing the error to the stack trace.

`SunPortalEnvironment` Class

The `SunPortalEnvironment` class inherits from `Jsr168Environment`. The `SunPortalEnvironment` integrates with the Sun portal server. (For specific versions refer to the release notes.) This class implements the following methods:

`getUserPrincipal()`	Looks up the user principal for trusted authentication mode using classes in the `com.iplanet.sso` package.
`getWriteModePreferenceScopes()`	Returns an enumeration of the preference types that can be written.
`getNamespace()`	Uniquely identifies the HTML for two portlets on the same page in a Sun portal environment.
`writeUserLevelPreference()`	Writes preferences using APIs in the `com.iplanet.am.sdk` package.

◆ ◆ ◆ CHAPTER 9

Troubleshooting

This chapter provides information on addressing any errors that occur.

If an error occurs, portlets provide information in two ways:

- Error messages tell you the nature of the problem and include a more details link that opens an error page with additional information to help you troubleshoot the error.
- Tracing tracks and logs what is happening behind the UI of your application.

Tracing

Tracing is one of the most effective ways to track what is happening behind the UI of your application. You can start and stop different levels of tracing, and view the tracing logs to help your team and our team resolve any application issues.

Tracing takes up memory and considerably slows application performance. You should not leave tracing running unless you are trying to narrow down a specific issue. You can stop and start portlet tracing and you can view the tracing logs. Tracing flags are enumerated in the WDK resource file `TraceProp.properties` located in `/WEB-INF/classes/com/documentum/debug` directory. This file contains all tracing flags that are defined in your application. If there is an unknown flag in this file, the Trace class initialization will generate a warning message but will continue.

Note – You must enable tracing for the current session by opening the `/wdk/tracing.jsp` file and checking the box that enables tracing. You can enable tracing for all sessions by setting `SESSIONENABLEDBYDEFAULT` to `true` in `/wdk/source/com/documentum/debug/TraceProp.properties` file.

The following tracing flags can be used to trace portlets:

DISPATCHER	Traces the Component Dispatcher. The dispatcher is an internal Documentum tool that does not have any public tracing capabilities.
ENVIRONMENT	Traces current environment, properties file, environment class, environment creation, and environment-specific calls.
HTML_PAGE_PROCESSOR	Traces tags rewritten for portlets.
HOOK	Traces environment lookup, user name, and repository.
PUBLISHED_CONTENT	Web Publisher portlet specific flag used to trace searching in the Published Content portlet. If this flag is turned on it displays an error for invalid entries in the XML configuration file for example, invalid attribute name and invalid object type name. This flags helps determine why containers are not displaying in the portlet, and displays the DQL used to search the portlet for each container.
SEARCH	Web Publisher portlet specific flag used to trace searching in the Published Content and Submit Content portlets. If this flag is turned on it displays an error for invalid entries in the XML configuration file in which the search component is used for example, invalid object type name, or invalid attribute name.
VIRTUALJS	Traces HTTP request and response including response status 304 and 200.
WDK_API_TRACE	Traces the local path for multi-part HTML file upload.