EMC® Documentum® Composer

Version 6.5 SP2

User Guide

P/N 300-009-462 A01

EMC Corporation
Corporate Headquarters:
Hopkinton, MA 01748-9103
1-508-435-1000
www.EMC.com

Published June 2009

Table of Contents

List of Figures

List of Tables

Preface

This guide describes how to use Documentum Composer to develop enterprise applications and deploy these applications on Documentum Content Server.

Intended audience

This guide is for users who are developing applications for Documentum Content Server. This guide assumes that the user has a basic understanding of the Documentum platform and content management in general.

Organization

This guide contains the following chapters:

- Chapter 1, Documentum Composer

  This chapter includes an overview of Documentum Composer and describes how to install Composer, configure a connection broker, enable tracing, and configure Java compiler preferences.

- Chapter 2, Managing Projects

  This chapter describes how to create a new Documentum project from scratch, create a project from a local DocApp archive, and create a project from a repository DocApp.

- Chapter 3, Converting Composer projects from DocApps and DocApp archives

  This chapter describes how to convert a DocApp or a DocApp archive created with Documentum Application Builder (DAB) to a new Composer project.

- Chapter 4, Managing Web Services

  This chapter describes how to view catalog services, import a client proxy and create services, and how to consume and export Web services.

- Chapter 5, Managing Alias Sets

  This chapter describes how to create and configure alias sets. An alias set is an object that defines one or more aliases and their corresponding values. An alias is a placeholder for usernames, group names, or folder paths.

- Chapter 6, Managing Aspects

  This chapter describes how to create and configure aspect types, aspect modules, and aspect attributes. An aspect customizes behavior or records metadata or both for an instance of an object

type. An aspect module consists of executable business logic and supporting material for an aspect, such as third-party software and documentation.

- Chapter 7, Managing Formats

  This chapter describes how to create and configure formats. A format object contains information about a file format recognized by Content Server.

- Chapter 8, Managing JARs and Java Libraries

  This chapter describes how to create and manage Java ARchive (JAR) files and Java libraries. JAR files are generally used to distribute Java classes and associated metadata, and can serve as building blocks for applications and extensions.

- Chapter 9, Managing Lifecycles

  This chapter describes how to create and manage lifecycles. A lifecycle specifies business rules for changes in the properties of an object, such as a document, as it moves through different states.

- Chapter 10, Managing Methods and Jobs

  This chapter describes how to create methods and jobs. Methods are executable programs that are represented by method objects in the repository. Jobs automate the execution of a method, for example how to transfer content from one storage place to another.

- Chapter 11, Managing Modules

  This chapter describes how to create and manage modules. Modules consists of executable business logic and supporting material, such as third-party software and documentation.

- Chapter 12, Managing Permission Sets (ACLs)

  Permission sets (also known as access control lists, or ACLs) are configurations of basic and extended permissions assigned to objects in the repository, including users, groups, and the actions they can perform.

- Chapter 13, Managing Procedures

  This chapter describes how to create and manage procedures. Procedures are applications that extend or customize the behavior of Documentum clients or Content Server.

- Chapter 14, Managing Relation Types

  This chapter describes how to create and manage relation types. Relation types define the relationship between two objects in a repository.

- Chapter 15, Managing Smart Containers

  This chapter describes how to construct and manage smart containers. Smart containers define objects and relationships in a template that gets instantiated at runtime.

- Chapter 16, Managing SysObjects

  This chapter describes how to create and manage SysObjects. SysObjects are the supertype, directly or indirectly, of all object types in the hierarchy that can have content.

- Chapter 17, Managing Types

  This chapter describes how to create and manage object types. An object type is similar to a template and represents a class of objects.

- Chapter 18, Managing XML Applications

This chapter describes how to view or modify an XML application configuration file.

- Chapter 19, Building and Installing a Project

  This chapter describes how to build an application and then install it in a repository.

- Chapter 20, Managing Projects and DAR Files Using Ant tasks and Headless Composer

  This chapter describes how to use headless Composer to import, build, and install an existing Documentum project.

- Chapter 21, Working with Source Control Systems

  This chapter describes how to use Composer in conjunction with source control systems.

Typographic conventions

The following table describes the typographic conventions used in this guide.

Typographic conventions

Typeface	Text type
Body normal	In running text: • Interface elements (button names, dialog boxes) • Java classes, interface names • Names of resources, attributes, pools, Boolean expressions, buttons, DQL statements, keywords, and clauses, environment variables, functions, menus, utilities • Pathnames, URLs, filenames, directory names, computer names, links, groups, service keys, file systems, environment variables (command line and text), notifications
Body normal double quotes	Chapter and section titles
Body Bold	In procedures: • User actions (what the user clicks, presses, selects, or types) in procedures • Interface elements (button names, dialog boxes) • Key names In running text: • Command names, daemons, options, programs, processes, notifications, system calls, man pages, services, applications, utilities, kernels

Typeface	Text type
Body Italic	• Book titles, emphasis (glossary terms, See and See also index references) • Variables in text (outside of command sample)
`Courier`	In procedures: • If shown on separate line, prompts, system output, filenames, pathnames, URLs, syntax examples
`Courier Bold`	• User input shown on separate line
`Courier Italic`	In procedures: • Variables in command strings • User input variables

Support information

Documentum's technical support services are designed to make your deployment and management of Documentum products as effective as possible. The *Customer Guide to EMC Software Support Services* provides a thorough explanation of Documentum's support services and policies. You can download this document from the Powerlink website (http://powerlink.EMC.com).

Related documentation

For related information about Content Server and the Documentum system, see the following publications:

- *EMC Documentum Content Server Fundamentals Version 6.5*
- *EMC Documentum System Object Reference Manual Version 6.5*
- *EMC Documentum Documentum Foundation Services Development Guide Version 6.5*
- *EMC Documentum Documentum Foundation Classes Development Guide Version 6.5*
- *Docbasic Reference Manual Release 4.0*
- *EMC Documentum XML Applications Developer Guide Version 6.5*
- *EMC Documentum Archive Developer Guide Version 6.5*

Revision history

The following changes have been made to this document.

Revision history

Revision date	Description
June 2009	Initial Publication

Chapter 1

Documentum Composer

This chapter contains the following topics:

Introduction to Composer

Documentum Composer provides tools to create and customize applications for Documentum Content Server. These applications specify how Content Server handles different types of content.

Composer is an Eclipse-based product, a stand-alone program built with the Eclipse platform. Since Composer is a stand-alone program, it contains all the required code and plug-ins. Composer is delivered in the form of a compressed .Zip file that is extracted to a directory on the local development machine.

Installing Composer

Documentum Composer is packaged as a compressed .Zip file that contains the Eclipse platform and all required plug-ins, as follows:

- Eclipse 3.4

  The basic Eclipse platform.

- EMF 2.2 (EMF, SDO, XSD)

  Eclipse modeling framework libraries (distributed as Eclipse plug-ins) for Ecore models. For example, EMF is used for generating Java models from Ecore schema definitions, and for creating and persisting Ecore model instances.

- GEF 3.2

  Graphical editing framework libraries (distributed as eclipse plug-ins) to generate visual editors for EMF Ecore models.

- VE 1.2

  Visual Editor plug-ins for creating SWT GUI components.

- Documentum plug-ins
- Workflow Manager

  Note: Workflow Manager is only bundled but not integrated with Composer. When you install Composer, Workflow Manager is extracted to the **../Composer/WorkflowManager** directory on your machine and you have to start Workflow Manager from that directory, as described in Starting Workflow Manager, page 19.

Composer supports Documentum 5.3 SP6 or later repositories and requires Java JDK 1.5.

To install Documentum Composer:

1. Extract the content of Documentum Composer .Zip file to a folder of your choice on your local machine, for example C:\Program Files. The Composer .Zip file has the format **DCTM_Composer_<version>.zip**, for example DCTM_Composer_R.6.5.zip.

   Composer is installed in the **..\Composer** folder.

2. If Java JDK 1.5 is not installed on the local machine or there is an older Java JDK installed, download the latest Java JDK 1.5 from Sun's Website at *http://java.sun.com/javase/downloads/index_jdk5.jsp* and follow Sun's installation instructions provided on the Website.

   Note: Documentum Composer requires Java JDK 1.5.

3. Set the **JAVA_HOME** environment variable on your local machine to point to the Java JDK 1.5.

   For example, if Java JDK is installed in the *C:/Program Files/Java/jdk1.5.0_15* directory on your local drive, set the JAVA_HOME variable to that path. The steps to access the environment variables may vary depending on the operating system running on the local machine. For example, in Windows XP, right-click **My Computer**. Select **Properties > Advanced > Environment Variables**.

4. Configure the connection broker to enable access to the repository, as described in Configuring the connection broker, page 19.

Installing the lightweight SysObject plugin

The lightweight SysObject plugin is not part of the main Composer distribution and must be installed in the <Composer_root>/plugins directory after you have installed Composer. To install the lightweight SysObject plugin:

1. Download the LightweightObject_<version>.zip file from the download site (http://powerlink.EMC.com).
2. Extract the plugin to the same directory as Composer. For example, if you extracted Composer to the C:\ directory, you also need to extract the LightweightObject_<version>.zip file to the C:\ directory.

3. Change to the<Composer_root>/plugins directory and verify that the com.emc.ide.artifact.lwdclass_1.0.0.jar and com.emc.ide.artifact.lwdclass_ui_1.0.0.jar files are in the directory.

Installing other Composer plug-ins

Composer plug-ins that offer additional functionality and are not part of the main Composer distribution must be installed in the **../Composer/plugins** directory after you have installed Composer.

Depending on how the plug-ins are packaged, you may need to extract the package to the main Composer directory on your local machine or extract the package to a temporary directory and then copy the plug-in file to the **../Composer/plugins** directory.

Starting Workflow Manager

Workflow Manager is only bundled but not integrated with Composer. Therefore you cannot start Workflow Manager directly from the Composer interface.

To start Workflow Manager:

1. Change to the **../Composer/WorkflowManager** directory on your machine.
2. Double-click **launch_wfm.bat**.

   The Workflow Manager editor and login dialog display. For more information about Workflow Manager, see the *Workflow Manager User Guide*.

UNIX and Linux support in Composer

You can use headless Composer on UNIX and Linux systems to install DAR files to Content Server repositories on UNIX, Linux, and Windows systems. Only the headless Composer distribution that is bundled with Content Server is supported in UNIX and Linux environments.

Alternatively, you can use the DAR Installer or headless Composer on Windows systems to install DAR files to Content Server repositories on UNIX and Linux systems.

See Chapter 20, Managing Projects and DAR Files Using Ant tasks and Headless Composer for information on how to run headless Composer with Ant tasks.

Configuring the connection broker

Every time you want to import or import a project or artifacts, you have to access a Documentum repository. Repository access is handled by the Documentum connection broker. Before you start Composer for the first time, you need to configure the connection broker with the repository information, such as the IP address and port number. Composer supports

To configure the connection broker:

1. On your local machine, change to the **..\Composer\plugins** directory.
2. Double-click the **com.emc.ide.external.dfc_1.00** folder.
3. Double-click the **documentum.config** folder.
4. Open the **dfc.properties** file with a text editor, such as Notepad. Add the DFC and connection broker information, similar to the following:

   ```
   dfc.docbroker.host[0]=[DocBroker IP address or host name]
   ```

5. Save your changes.

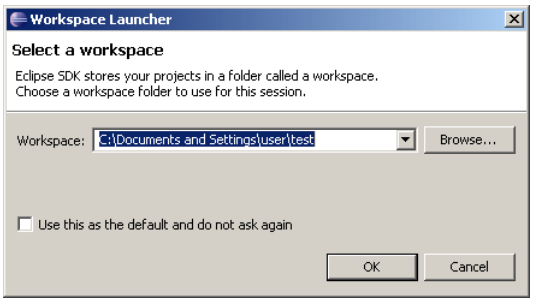
Starting and configuring Composer

Composer runs on top of the Eclipse platform and uses a similar development concept. In order to run Composer, you must first configure at least one so-called workspace. The workspace is the directory where Composer stores your work. You need to specify the location for the workspace before you can use Composer.

To start Composer and configure a workspace:

1. Go to the **..\Composer** installation directory on the machine where you extracted the Composer .Zip file and double-click the **eclipse.exe** icon.

   When you start Composer for the first time, you see the **Workspace Launcher** dialog that allows you to select the location of your workspace.

   The workspace is where Composer stores all the source files and dependencies for your projects. You can have more than one workspace in Composer, for example for different projects, but an individual project can only be stored in one workspace.

2. Accept the default location for your workspace or enter a new location in the **Workspace** field, then click **OK**.

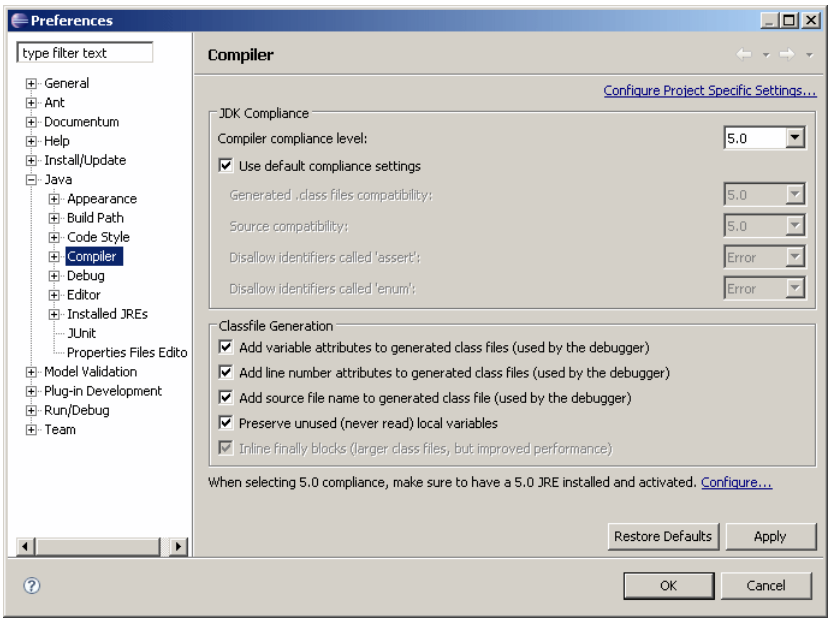
   The Composer workbench appears.

Configuring the Java compiler preferences

Composer requires Java JDK 1.5. Verify that Composer is using the correct Java compiler before you start working on any projects.

To verify that Composer uses the correct Java compiler:

1. In the Composer main toolbar, select **Window > Preferences**.
   The **Preference** dialog appears.



2. Verify that the compiler compliance level is set to 5.0 or if it is not set correctly, set the compliance level to 5.0.
3. Click **OK** to save your changes.

Configuring the Java JRE

The installed Java Runtime Environment (JRE) in the Composer preferences must match the Java Development Environent (JDK) that is configured in the environment variables on the local machine that is running Composer. If the JRE does not match, the Composer project may not install correctly in a repository.

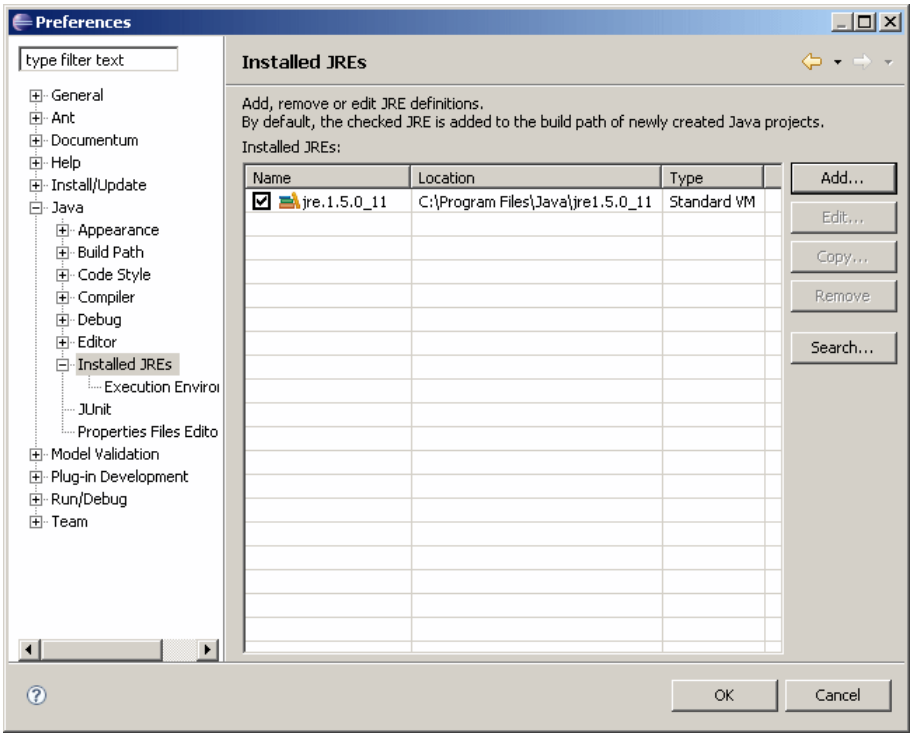
Determine which Java JRE version is installed on the local machine by verifying the path that is set in the JAVA_HOME environment variable. The JAVA_HOME variable should point to the associated Java JDK. When the Java JDK is installed, it automatically installs a matching JRE.

The steps to access the environment variables may vary depending on the operating system running on the local machine. For example, in Windows XP, right-click My Computer. Select **Properties > Advanced > Environment Variables**. After you have determined the Java JDK version on the local machine, set the installed JRE version in Composer to the same version as the JDK.

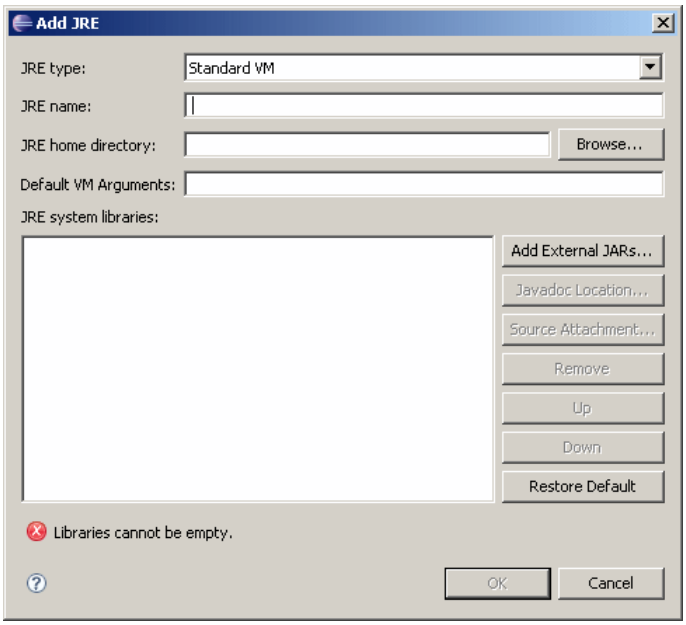
Note: Composer requires JRE 1.5. If your local machine has an earlier JRE version installed, you must upgrade Java before you proceed.

To set the Java JRE in Composer:

1. In the Composer main menu select **Window > Preferences**.
   The **Preferences** dialog appears.
2. Click the **Java** option to expand it, then click **Installed JREs**.
   The **Installed JREs** page appears.



3. If the installed JRE does not match the JRE on the local machine, click **Add** to add another JRE.
   The **Add JRE** dialog appears.



4. Click **Browse** and select the Java JRE 1.5 directory on your machine, for example **C:/Program Files/Java/jdk1.5.0_15/jre**. Once you entered the JRE home directory, the available JRE name automatically appears in the **JRE name** field, and the system libraries display in the JRE system libraries list box.
5. Click **OK**.
6. Verify that the new JRE is on the Installed JREs page.

Chapter 2

Managing Projects

This chapter contains the following topics:

Composer projects

A Composer project specifies the objects that make up an application. Therefore, you need to create a project before you can start developing a new application.

A project consists of a project folder and a number of subfolders that contain the artifacts, such as lifecycles, permission sets, jobs, and others. For a complete list of artifacts, see Table 1, page 32. A Composer project is marked with an  icon.

There are several ways to create a Composer project:

- Create a new, empty project as described in Creating a new project, page 25.
- Import an existing project into Composer as described in Importing a project, page 27.
- Create a Composer project from a local 5.3 DocApp archive as described in Converting a DocApp archive, page 51.
- Create a Composer project from a 5.3 DocApp, as described in Converting a DocApp to a Composer project, page 47.

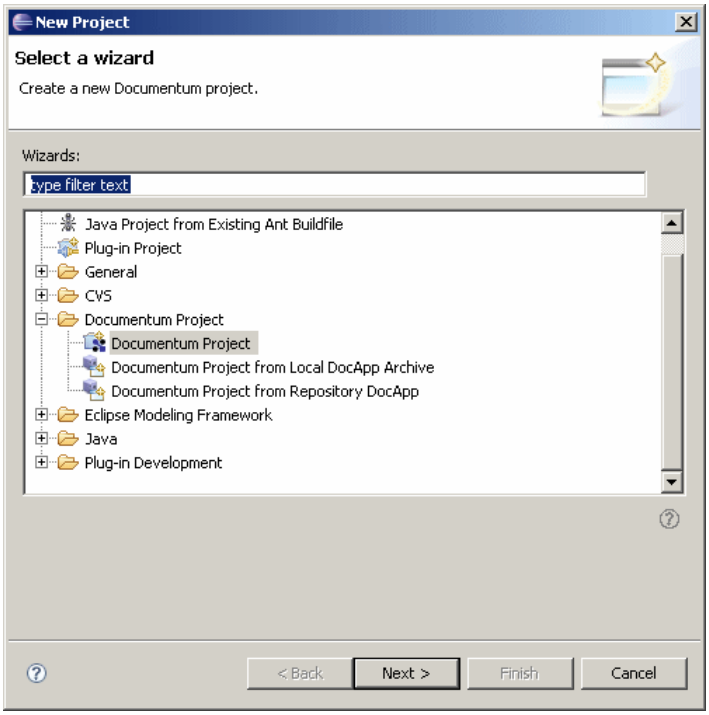
Creating a new project

You need to create new project whenever you want to create an application from scratch.

To create a new project:

1. Open the **New Project** wizard in one of the following ways:
   - From the main menu in Composer select **File** > **New** > **Project**.
   - Right-click in the Documentum Explorer space and select **New** > **Project.**

   The **New Project** dialog appears.



2. Double-click the **Documentum Project** folder to expand it, then select **Documentum Project** and click **Next**.

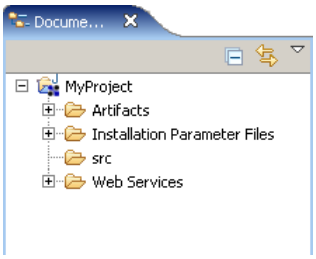
   The **New Documentum Project** dialog appears.

3. Enter a name for your project in the **Name** field and an optional description and click **Next**.
4. Select any projects that you want to designate as reference projects and click **Finish**. For more information about reference projects, see Composer reference projects, page 29

Note: When you create a project for the first time, a dialog box prompts you to select the associated **Documentum Development** perspective. Click **Yes** to select the **Documentum Development** perspective.

Composer creates the new project in the **Documentum Navigator** view (Figure 1, page 27).

Figure 1. Composer project folders



By default, a project contains an **Artifacts** folder, **Installation Parameter Files** folder, **src** folder, and a **Web Services** folder, as follows:

- **Artifacts**

  The **Artifacts** folder contains subfolders for all artifacts that are available in Composer. When you create a new project, these artifact subfolders are empty.

- **Installation Parameter Files**

  The **Installation Parameter Files** folder is used for storing the installation parameter files for installing a project. By default, this folder is empty when you create a new project. Once you have added artifacts and configured installation options for the project and artifacts, the associated **.installparam** installation parameter files are stored in the this folder.

- **src**

  The **src** folder is used for storing source files that you want to add to your project. By default, the **src** folder is empty when you create a new project.

- **Web Services**

  The **Web Services** folder contains Web services files, such as client libraries, WSDL files, and source code files. By default the **Web Services** folder is empty when you create a new project.

Importing a project

You can import existing projects from a local directory into the Composer workspace. If you are using a source control system to manage your files, you need to checkout the project from the source control system before you can import it into Composer.

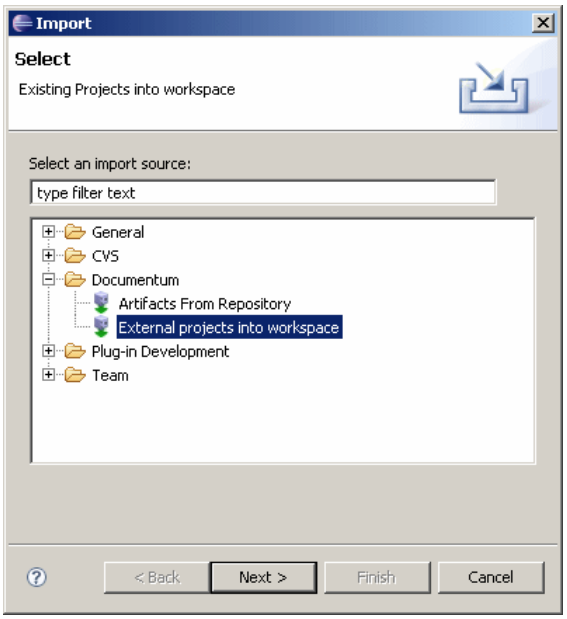
This section describes how to import projects from a local directory, for more information about using Composer in conjunction with a source control system, see Chapter 21, Working with Source Control Systems.

Note: You cannot import a DAR file into a project. A DAR file is the executable version of a project that gets installed in a Documentum repository. A DAR file contains only the binary files of a project but not the source files.

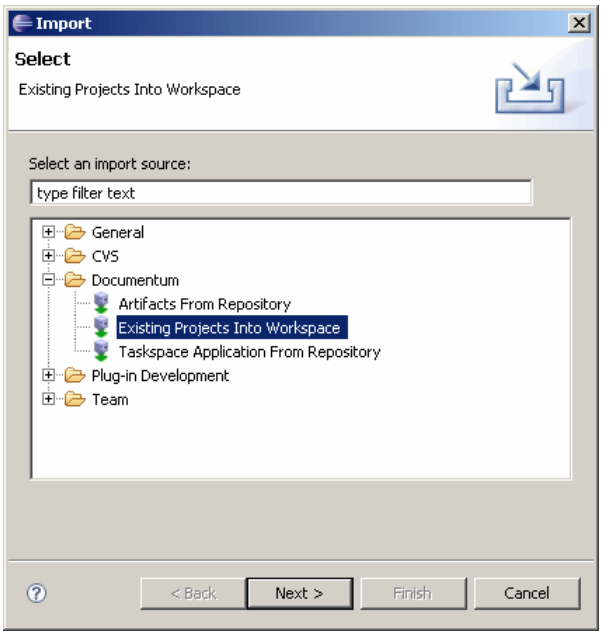
To import an existing project:

1. Start the **Import** wizard in one of the following ways:
   - Right-click the workspace and select **Import** from the pop-up menu.
   - In the main Composer menu, click **File > Import**.

The **Import** wizard appears.



2. Expand the **Documentum** folder, select **Existing projects into workspace**, then click **Next**.
   The **Import Projects** dialog appears.



3. Do one of the following:
   - Click the **Select root directory** radio button. Enter the project directory or click **Browse** to search for the directory.
   - Click the **Select archive file** radio button. Enter the archive file or click **Browse** to search for the file.

Note: The archive file must have the format .jar, .zip, .tar, .tar.gz, or .tgz. You cannot import Documentum archive (DAR) files. DAR files contain only the binary files of a project but not the source files.

Composer displays the available projects in the **Projects** list box.

4. Select one or more projects to import and select **Copy projects into workspace**, then click **Finish** to import the projects.

Composer imports the projects and displays them in the **Documentum Navigator** view.

Note: Composer does not support importing renditions of documents.

Composer reference projects

Composer lets you create references between projects. This is useful if you have projects that share resources such as Documentum artifacts, libraries. or JAR files. You can specify reference projects when you create a new project or by editing an existing project.

In general, you can designate any project as a reference project if it has resources that you want to share with other projects. Documentum also supplies special reference projects that allow you to access Documentum functionality.

Documentum supplied reference projects

Documentum supplied reference projects are non-buildable projects that you need if you want to use or extend Documentum artifacts (more specifically, Documentum artifacts with names that begin with 'dm').

Every project that you create within Composer has the DocumentumCoreProject designated as a reference project by default. The DocumentumCoreProject contains all of the artifacts that are provided by Content Server, so you can use or extend these artifacts out of the box. The project is read only and should not be modified. It is marked with the  icon and is displayed only in the **Package Explorer** view, and not the **Documentum Navigator** view.

If you need to use or extend an artifact from another Documentum product, obtain the reference project that contains the artifacts that you want to use. The various Documentum products supply their Composer reference projects in their respective download areas on EMC's software download site, https://emc.subscribenet.com.

It is useful to know the following points, which will help you understand when you need to download and reference a Documentum supplied reference project:

- Your Composer project cannot contain artifacts with names that begin with 'dm'. 'Dm' is a reserved prefix for Documentum. Because 'dm' is a reserved prefix, 'dm' artifacts that are present in user projects are detected as errors by Composer. A 'dm' artifact, however, can exist in Documentum supplied reference projects, such as DocumentumCoreProject. This provides you with a mechanism to use and extend 'dm' artifacts.
- You can use or extend any 'dm' artifact that is provided by Content Server without needing to download a separate reference project, because the DocumentumCoreProject is already provided

and referenced for your convenience. You need to obtain the reference projects for all other 'dm' artifacts that are not provided by Content Server (for example, Taskspace 'dm' artifacts).

- If you import an artifact from the repository, it might depend on other artifacts to function. If these other artifacts are not present in your project or in reference projects, Composer automatically imports these artifacts from the repository. If the automatically imported artifacts have names that begin with 'dm,'' it will cause the following error:

  "Type name is invalid. Type names must not begin with 'dm'. For more information, see the 'Reference projects' section in the Composer User Guide."

  If this error occurs, delete the newly imported artifacts, import and designate the appropriate projects as reference projects, and re-import the desired artifacts.
- If you import an artifact that indirectly references a 'dm' artifact, you still need to import the project that contains the 'dm' artifact and designate it as a reference project. For example, if you are importing a type named my_child_type that depends on a type named my_parent_type that depends on a 'dm' type, then you must download the project that contains the 'dm' type, import it into your workspace, and designate it as a reference project.
- The previous points also apply to converting DocApps and DocApp archives as well. If the DocApp or DocApp archive uses or extends 'dm' artifacts that are not in DocumentumCoreProject, you must import all required Documentum supplied reference projects into your workspace before converting the DocApp or DocApp archive. During the conversion, Composer will prompt you to specify the necessary reference projects.

Designating projects as reference projects

There are two ways to designate projects as reference projects:

- During the creation of a new project
- By editing an existing project

If you are converting a DocApp or DocApp archive, you must designate reference projects during the creation of the project.

If you are importing an artifact from the repository that requires a Documentum supplied reference project, you must designate the reference project first, before importing the artifact. You can do this when creating a project or by editing an existing project.

Designating reference projects for new Composer projects

If you know beforehand that your project will use or extend 'dm' artifacts that are not in DocumentumCoreProject, obtain the appropriate reference projects and import them into your workspace. When this is completed, you can designate the projects as reference projects.

You must follow this procedure if you are converting DocApps or DocApp archives into Composer projects and those DocApps or DocApp archives use or extend a 'dm' artifact that is not in DocumentumCoreProject.

You must also follow this procedure if you are importing 'dm' artifacts or artifacts that extend a 'dm' artifact that are not in DocumentumCoreProject.

To obtain and import Documentum supplied reference projects into your workspace:

1. Go to the EMC download site,https://emc.subscribenet.com/, to download the necessary reference projects. The reference projects should be located in the Documentum product's download area.
2. Import the .zip file of the reference project into your Composer workspace as described in Importing a project, page 27. When the import is complete, the project appears in the Documentum Navigator view of Composer.
3. Create a new project from scratch or from a DocApp or DocApp archive as described in the following sections. When prompted, designate the appropriate reference projects.
   - To create a new project from scratch, see Creating a new project, page 25.
   - To create a Composer project from a 5.3 DocApp. see Converting a DocApp to a Composer project, page 47.
   - To create a Composer project from a local 5.3 DocApp archive, see Converting a DocApp archive, page 51.

Designating reference projects for existing Composer projects

This procedure describes how to specify reference projects for an existing project. You can also specify reference projects when creating a new project with the New Project wizard.

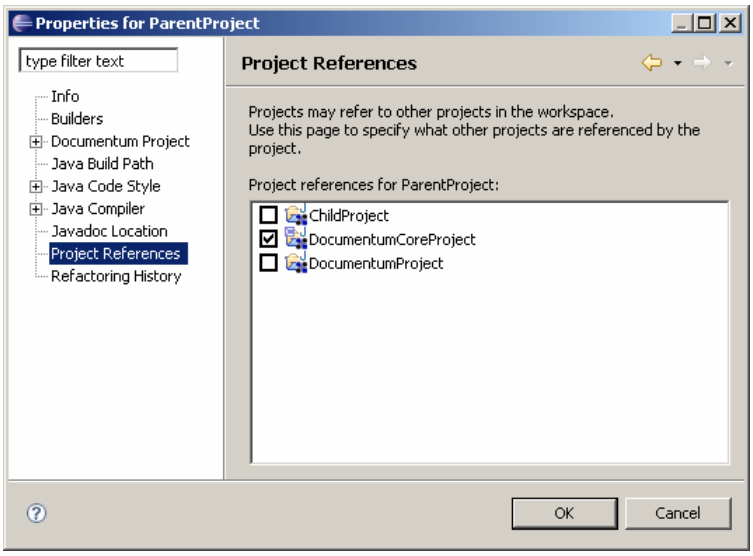
Before you can designate a project as a reference project, the project must be in your Composer workspace. If it is not, you must import the project first, as described in Importing a project, page 27

Note: If you created a new project from a DocApp and want to reference Documentum supplied reference projects, do not follow this procedure. You must import the Documentum supplied reference projects into your workspace first and select them when prompted by the New Project wizard. If you do not, you will encounter errors during the import process.

To create a reference to another project:

1. In the **Documentum Navigator** view, right-click the project for which you want to create a reference and select **Properties** from the drop-down list.

   The **Properties** dialog appears.



2. Select **Project References**. The projects that are available for referencing are displayed in the **Project references for ParentProject** list. Select one or more projects that are referenced by this project.
3. Click **OK**.

Note: When you are ready to install your project and the project references other projects, be sure to install all projects in the correct order. For example, if project B references artifacts in project A, then project A must be installed first.

Composer artifacts

Artifacts are Documentum resources, for example object types, modules, and procedures. In Composer you can create new artifacts or add artifacts to an application using different kinds of wizards and dialogs.

Documentum Composer offers a variety of artifacts, as described in Table 1, page 32.

Table 1. Documentum artifacts

Artifact name	Description
Alias Set	Collection of aliases. An alias is a symbolic name that is used as a reference to an actual user, group, cabinet or folder name. A collection of aliases is called an alias set.
Aspect Module	Customizes behavior for an instance of an object type.
Aspect Type	Specifies the metadata for an instance of an object type.
Format	Contains information about a file format recognized by Content Server. A predefined set of file formats is installed by default when a repository is configured.

Artifact name	Description
Form Template	Identifies a functional element for use within a DocApp. You cannot create a form template by using Composer. However you can import forms from an existing DocApp.
Installation Parameter	Specifies installation options that apply to the entire project, such as pre- and post-installation procedures and upgrade options.
Jar Definition	Encapsulates a JAR file. A Java ARchive or JAR file is an archive file that aggregates many files into one.
Java Library	Encapsulates a Java library. A Java library contains interface JARs and implementation JARs that can be linked to other artifacts, such as modules.
Job	Automates the execution of a method, for example how to transfer content from one storage place to another. The attributes of a job define the execution schedule and turn execution on or off.
Lifecycle	Specifies business rules for changes in the properties of an object, such as a document, as it moves through different stages during a business process.
Method	Executable program that is represented by method objects in the repository.
Module	Units of executable code.
Permission Set	Configurations of basic and extended permissions assigned to objects in the repository that lists users and user groups and the actions they can perform.
Procedure	A Docbasic or Java program. Procedures are generally used to specify pre- and post-installation tasks.
Relation Type	Defines the relationship between two objects in a repository.
SysObject	The parent type of the most commonly used objects in the Documentum system. The SysObject type has properties that it passes on to all its subtypes.
Type	Represents a class of objects. The definition of an object type is a set of attributes. The attribute values describe individual objects of the type.

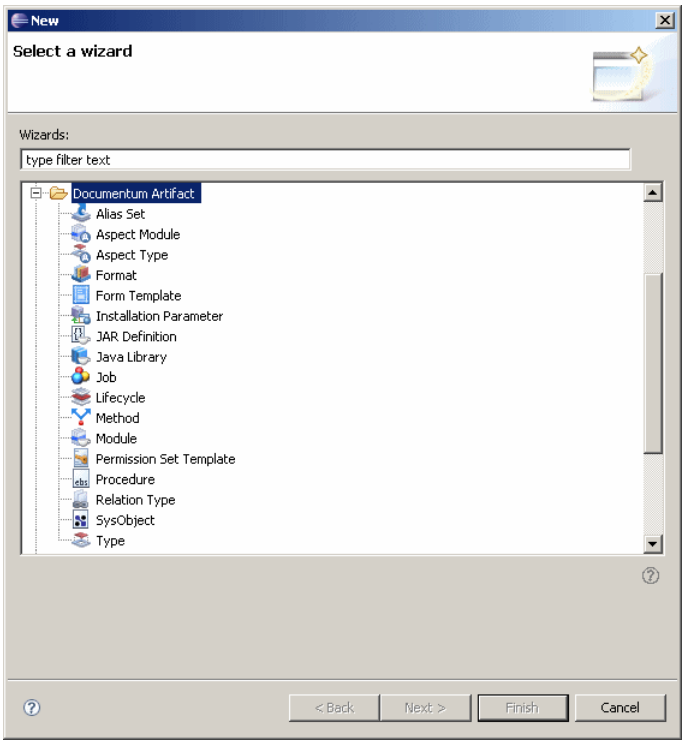
Creating an artifact

Use the artifact wizard to create a new artifact.

To create a new artifact:

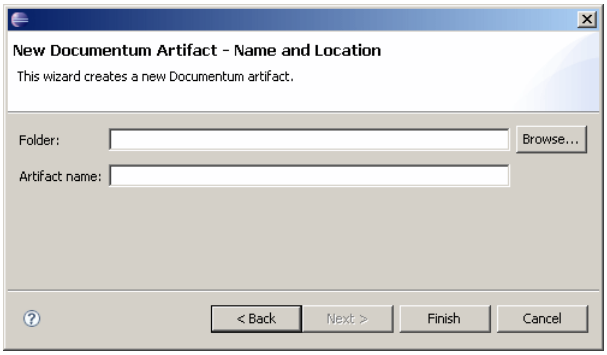
1. Open the **Select a wizard** dialog in one of the following ways:
   - From the Composer menu, select **File > New > Other**.
   - In your project, expand the **Artifacts** folder and right-click the artifact folder in which you want to create a new artifact. Select **New > Other.**

   The **Select a wizard** dialog appears.



2. Double-click the **Documentum Artifact** folder to expand it, select the artifact you want to create from the artifact list, then click **Next**.

   The **New Documentum Artifact – Name and Location** dialog appears.



3. Specify the folder in which you want to create the artifact in the **Folder:** field or accept the default folder.

   Note: You should always create a new artifact in the **Artifacts** folder. If you create a new artifact directly under the project root, the artifact is not installed properly in the repository.

4. Enter a name for the artifact in the **Artifact name:** field or accept the default artifact name. The default artifact name varies depending on the type of artifact you are creating.
5. Click **Finish**.

   The editor for the new artifact appears. For more information about the individual artifact editors and how to configure each artifact's properties, see the associated chapters in this guide.

Note: Composer supports copying and pasting of artifacts only within the same project. You cannot copy artifacts from other projects.

Importing artifacts

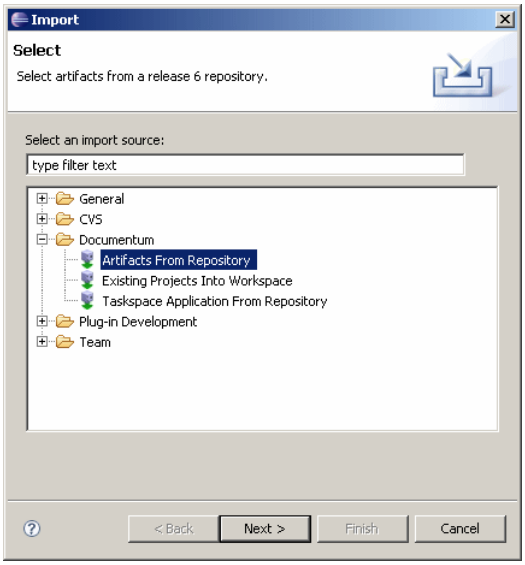
Documentum Composer lets you import individual artifacts from a repository into an existing project. Before importing artifacts, make sure to import and reference all relevant Composer projects that are needed for the artifacts that you are importing. If the artifact that you are importing depends on other artifacts that are not in your project or reference projects, Composer tries to import all other required artifacts from the repository. For more information, see Composer reference projects, page 29

Note: You can only import artifacts from a repository. You cannot import artifacts from a local project into another local project.

To import individual artifacts:

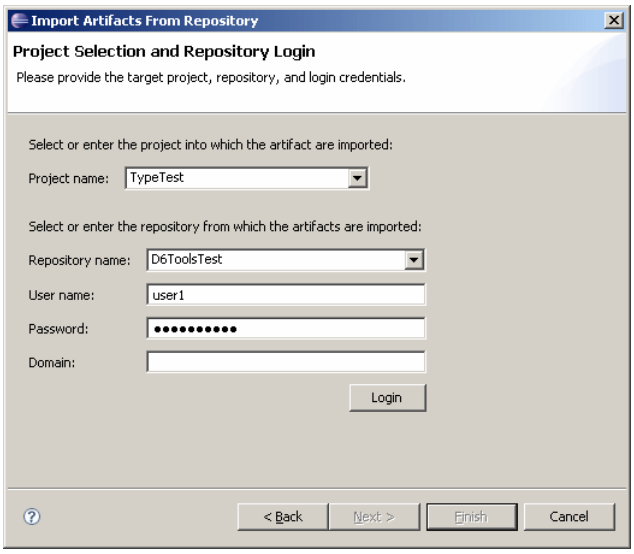
1. Start the **Import** wizard in one of the following ways:
   - Right-click the workspace and select **Import** from the pop-up menu.
   - In the main Eclipse menu, click **File > Import**.

   The **Import** wizard appears.



2. Double-click **Documentum** to expand the folder structure, then select **Artifacts From Repository** and click **Next**.

   The **Project Selection and Repository Login** dialog appears.

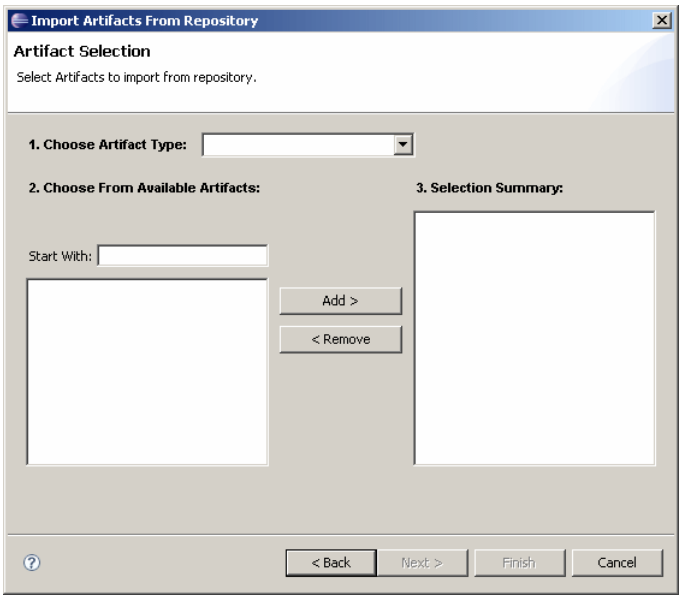


3. Enter the project and repository information, as described in Table 2, page 36, then click **Login**.
   If the login credentials for the repository are correct, you are logged into the repository.

Table 2. Project and repository information

Property	Description
Project name	The name of an existing project into which the artifacts are imported. If you do not have an existing project, you must create a project before you can import any artifacts. For more information about creating a project, see Creating a new project, page 25.
Repository name	The name of the repository that contains the artifacts.
User name	The name used to login to the repository that contains the artifacts.
Password	The password used to login to the repository that contains the artifacts.
Domain	The domain name of the repository. The domain name only needs to be specified if the repository resides in a different domain than the client from which the repository is accessed.

4. Click **Next >**.
   The **Artifact Selection** dialog appears.



5. Select the artifact object type from the **Choose Artifact Type** list. The available artifacts of that type appear in the **Available Artifacts** list.

   The **Selection Summary** field displays information about the selected artifacts, such as artifact object type and object ID.

6. Select one or more objects from the **Available objects** list, then click **Add**.

   Note: Composer only lists user-defined objects and not those created by Content Server when a repository is configured.

7. When you are done selecting artifacts, click **Finish** to import the artifacts from the repository. The artifacts are imported into the project.

   Note: If you import an artifact from a repository, move the artifact to a different location within the project, then import the artifact from the repository again, you will end up with duplicate artifacts in two different locations in your project.

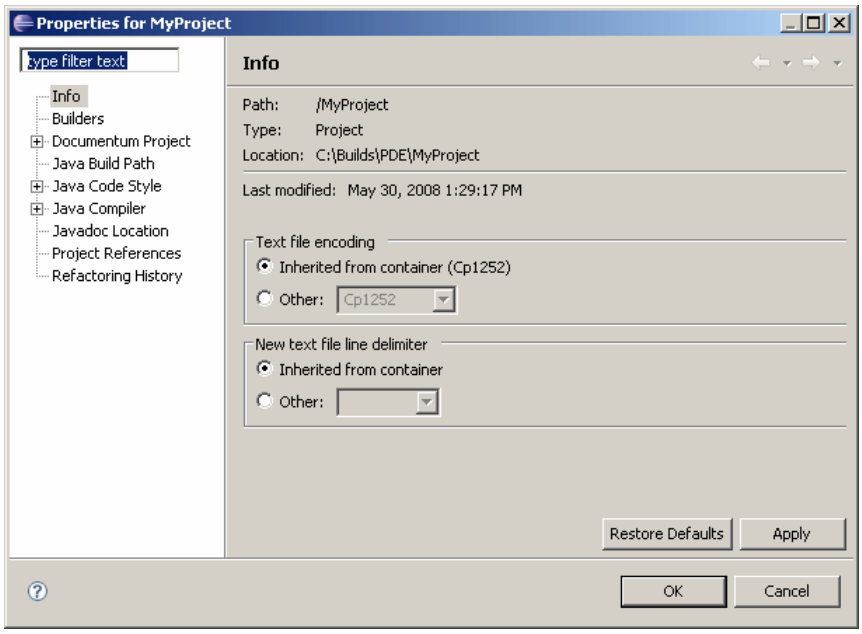
Configuring project properties

You can configure a variety of project properties using the **Properties** dialog, such as install options, DFS module options, and project install procedures.

To access the **Properties** dialog for a project, right-click the project and select **Properties** from the drop-down list.

The **Properties** dialog appears (Figure 2, page 38).

Figure 2. Project properties



For more information about configuring:

- Project install options, see Configuring the project installation options, page 194.
- Project install procedures, see Configuring pre- and post-installation procedures, page 196.
- Project references, see Designating reference projects for existing Composer projects, page 31.
- DFS module options, see Configuring DFS module options, page 59.
- DFS library options, see Configuring the DFS services library, page 60.

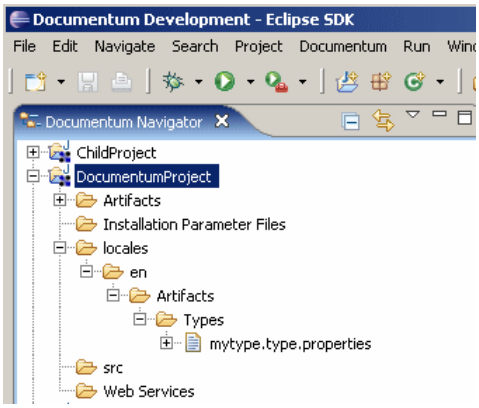
Localizing a project's types

Composer currently only supports localization of Type artifacts.

To localize a project's types:

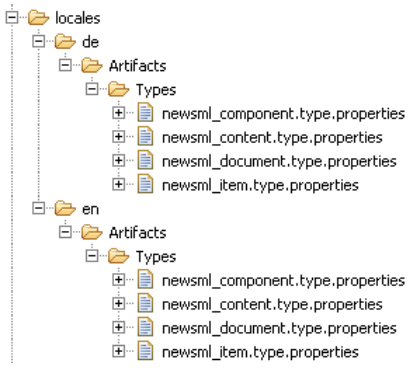
1. For every type in the project:
   a. In the **Attributes** tab, expand an attribute node and click **Application Interface Display**. The **General** section appears to the right.
   b. In the **General** section, ensure that a value for the **Label** field is specified.
   c. Complete steps a and b for every attribute.
   d. In the **Display** tab, ensure that a value for the **Type label** field is specified.
2. In the **Documentum Navigator** view, right-click the project that contains the types that you want to localize.
3. Select **Generate Localization Template** from the drop-down list.

Composer generates a **locales** directory under the project's root directory. By default, the **locales** directory contains an English "**en**" folder, that has the same **Artifacts** directory structure as the main project folder.



The **Artifacts** folder in the **locales** directory lists the artifacts that contain the data that can be localized.

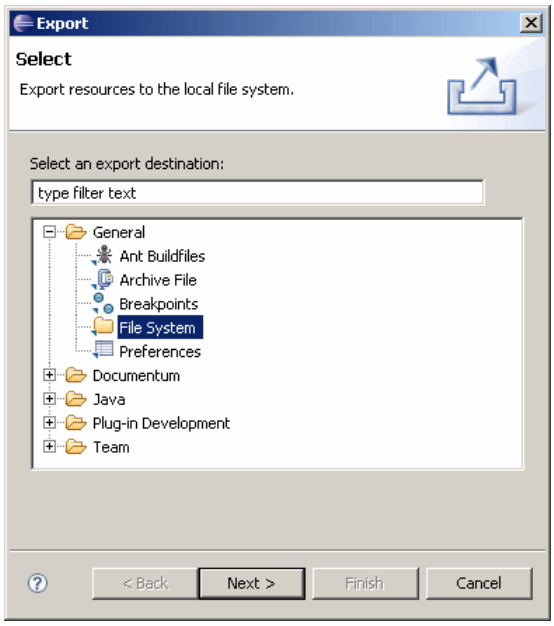
4. Make a copy of the complete **en** folder under the **locales** directory and rename the folder to the language locale you want to make available. For example, if you want to provide German labels for your application, create a **de** folder.



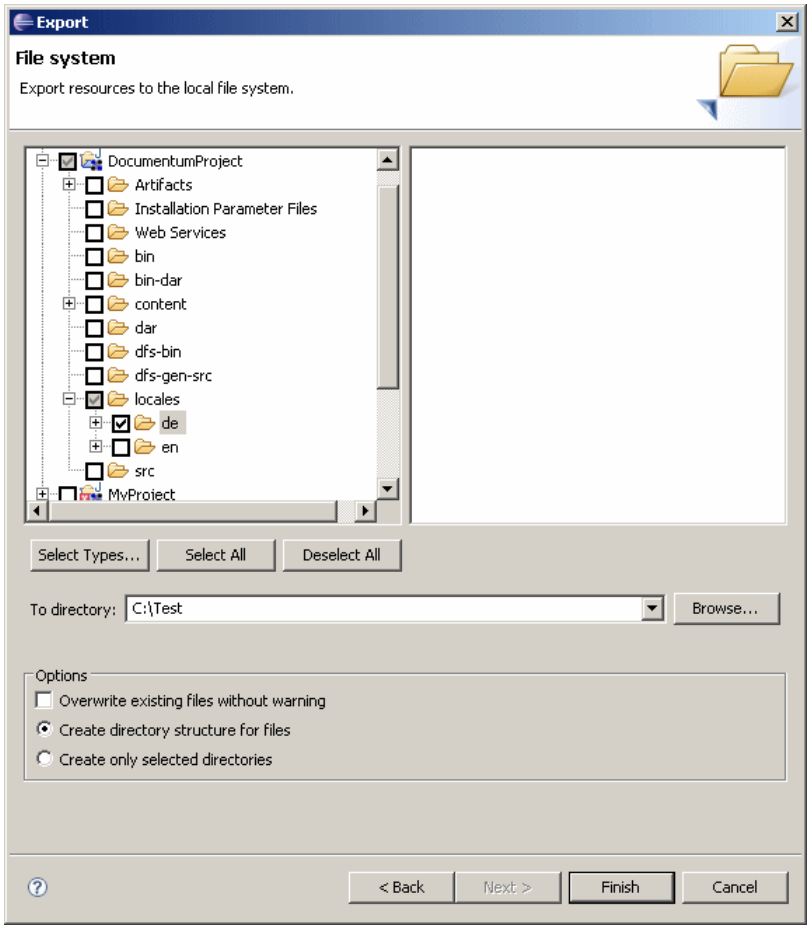
If you are sending out the .**properties** files to be translated externally, use the following procedure to export the files:

a. In Composer, select **File > Export**.

   The **Export** dialog appears.



b. Select **File System**, then click **Next**.
The **File System** dialog appears.



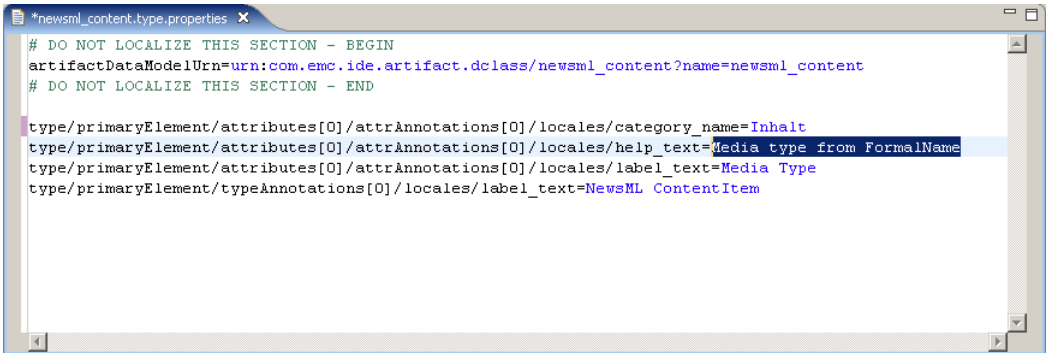
c. Expand the project that contains the **locales** folder you want to export, then check the language locales, for example **de**.

d. Enter the directory to which you want to export the files in the **To directory:** field.

e. Select the **Create directory structure for files** option, then click **Finish**.

Composer exports the content of the directory structure and content of language locales folder to the selected directory on the local machine. Deliver the files to the translation team.

The translation team needs to translate the strings on the right side of the equal sign (=) in the .**properties** file in the **locales** folder. The **locales** folder directory structure and .**properties** file names should not be changed.



Note: The *type/primaryElement/typeAnnotations[0]/locales/value_mappings[xx]/map_data_string* key cannot be localized.

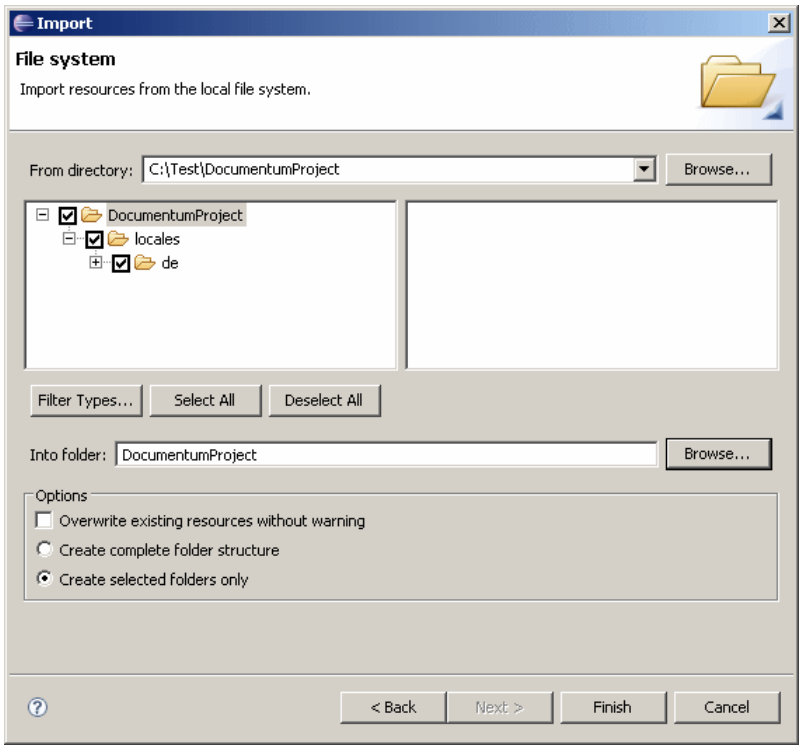
5. After the .properties files have been translated, import the files back into the project using the following procedure:

a. In Composer, select **File > Import**.

The Import dialog appears.

b. Select **File System**, then click **Next**.

The **File system** dialog appears.



c. Enter the directory path to the project folder that contains the **locales** folder with the translated files in the **From directory:** field.

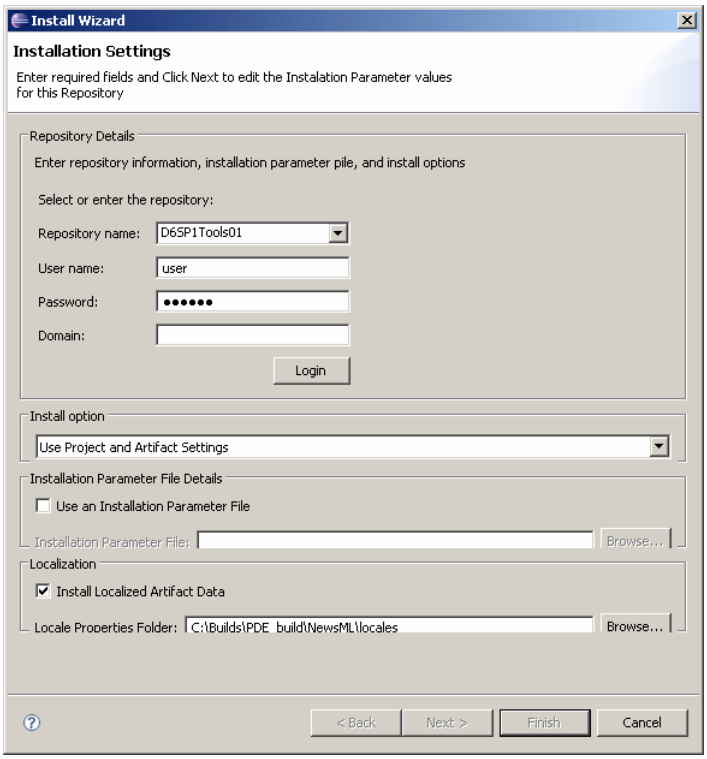
d. Select the project and the locales folder.

e. Enter the project name of the Documentum project from which the **locales** folder was originally exported in the **Into folder:** field. If the translation team did not change the files names and folder structures, the project name should be identical to the project name in the **From directory** field.

f. Select **Create selected folder only**. If you do not want to confirm the import of each file, check **Overwrite existing resources without warning**.

g. Click **Finish**.

6. When you are ready to install your project in a repository, check the **Install Localized Artifact Data** option in the **Install Wizard** and enter <Composer_project_root>/locales as the path to the localized data.



For more information about installing your project, see Installing a project, page 199.

When your project is installed in a repository, the new language strings are automatically merged and the new language becomes available in the repository.

Note: We do not recommend changing labels, descriptions, or tabs, or moving the .properties file after you created the localization template because the new language strings may not merge properly. If you rename labels and other localizable application data, you need to regenerate the localization template before you translate the strings.

Enabling tracing

You can enable tracing to monitor processes, for example when building or importing a project. By default, Eclipse has tracing disabled.

Note: Tracing should be used for debugging purposes only because it impacts Composer performance.

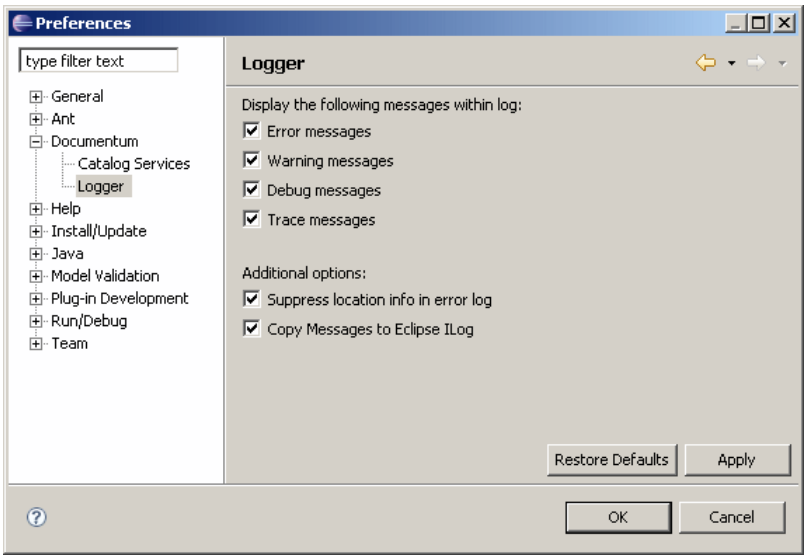
To enable tracing:

1. In the Eclipse main toolbar, select **Window > Preferences**.

   The **Preference** dialog appears.

2. Double-click Documentum to expand the tree structure and select **Logger**.

   The **Logger** options display.



3. Check the **Trace Messages** and the **Copy Messages to Eclipse ILog** options, then click OK.

By default, Composer stores all error log files in the .metadata subdirectory of the workspace.

Installing a project

You can install Composer projects directly within Composer or export your project as a DAR file and install it with the DAR Installer plugin or headless Composer. For more information about building and installing a project within Composer, see Chapter 19, Building and Installing a Project. For more information on installing DAR files, see Installing a DAR file with the DAR Installer, page 44 or Chapter 20, Managing Projects and DAR Files Using Ant tasks and Headless Composer

Installing a DAR file with the DAR Installer

A DAR file is a deployable, package representation of a Composer project. You can use the DAR Installer to install a DAR file to a repository if you do not want to use the interface within Composer. The DAR Installer requires Composer to be installed, but does not launch the full Composer IDE.

This plugin is useful for installing Documentum product DAR files or in cases where you want to decouple the development of DAR files from the installation of DAR files. When you open the DAR Installer, it creates three folders in your Composer installation directory:

- darinstallerconfig — contains configuration files for the DAR Installer Plugin
- darinstallerlogs — the default location of the log files
- darinstallerworkspaces — workspaces that are created and used by the DAR Installer Plugin. The DAR Installer Plugin does not delete these workspaces automatically after installation of the DAR file. The workspace directories are named in the following form: darinstaller_workspace_yyyy-mm-dd-hh-mm-ss. Moving, deleting, or adding projects manually to the workspace might have adverse effects on DAR installations.

The DAR Installer requires you to fill in certain values that are marked with an asterisk (*). All other fields are optional. For a description of the fields for the DAR Installer plugin, see Table 3, page 45.

Note: If the DAR file that you are installing depends on other, required DAR files, you must install those required DAR files first. If DAR files that you want to install depend on DAR files that you have already installed, you need to have the DAR files in the same location as the DAR file that you want to install. The DAR installer plugin will look for them there.

To install a DAR file:

1. Download the DAR Installer zip file. You can find the DAR Installer Plugin by going to https://emc.subscribenet.com/control/dctm/search and searching for "Composer" to reach the Documentum Composer download site.
2. Unzip the DAR Installer zip file to the root of your Composer or headless Composer installation directory.
3. Run darinstaller.exe, which is located in the Composer root directory, to start the DAR Installer Plugin.
4. In the **DAR Details** section, specify values for the fields.
5. In the **Connection Broker Details** section, specify values for **Connection Broker Host** and **Connection Broker Port** and click **Connect**.
6. In the **Repository Details** section, specify values for the fields and click **Install** to install the DAR file to the repository.

You can view the log for the DAR installation by selecting the log file from the **Log File** drop down menu and clicking **Open**.

Table 3. DAR Installer fields

Parameter	Required	Description
DAR	Yes	The absolute file path to the .dar file that you want to install. The file path cannot contain any I18N characters or the installation will fail.
Input File	No	The absolute file path to the install-based parameter file
Local Folder	No	The absolute file path to localized .properties files. If you want to make your application available in other languages, you need to localize the project data such as labels, tabs, and descriptions.
Log File	No	The file to save the log to. If this is not specified, the file defaults to <DAR>.log

Parameter	Required	Description
Connection Broker Host	Yes	The address of the Connection Broker
Connection Broker Port	Yes	The port of the Connection Broker Repository
Repository	Yes	The name of the repository that you want to install the DAR file to. Click on the Connect button after entering the Docbroker host and port to retrieve the available repositories.
User Name	Yes	The login name for the repository
Password	Yes	The password for logging into the repository
Domain	No	The domain where the repository resides

Chapter 3

Converting Composer projects from DocApps and DocApp archives

DocApps and DocApp archives are Documentum applications that were created using Documentum Application Builder (DAB) prior to release 6.0. With releases 6.0 and later, you create, edit, and install applications with Composer, so you will have to convert existing DocApps and DocApp archives to Composer projects if you want to modify them. The conversion is necessary because DocApps and DocApp archives were packaged with proprietary binary files and Composer uses XML files to represent these proprietary binary files.

You can convert DocApps into Composer projects by importing the DocApp straight from the repository. You can also convert DocApp archives into Composer projects by having Composer install the DocApp archive into a repository and converting the project for you.

The following rules apply when converting DocApps and DocApp archives:

- You can convert any existing version 5.3 or later repository DocApp or DocApp archive into a Composer project.
- You can install the resulting Composer project to any 5.3 or later repository. You can install DocApps that you have converted to Composer projects to an older repository as long as the functionality is supported in the older repository. For instance, if your DocApp only has custom subtypes of dm_document, you can convert a 6.0 DocApp into a Composer project and install it to a 5.3 repository. However, you cannot do this if the DocApp contained artifacts not supported by Documentum 5.3, such as Smart Containers or Aspects.
- You cannot convert a DocApp or DocApp archive that is stored in a repository prior to version 5.3. If you want to convert, you must first upgrade the DocApp to version 5.3.
- If you are upgrading the repository, do so before converting the DocApp.

Converting a DocApp to a Composer project

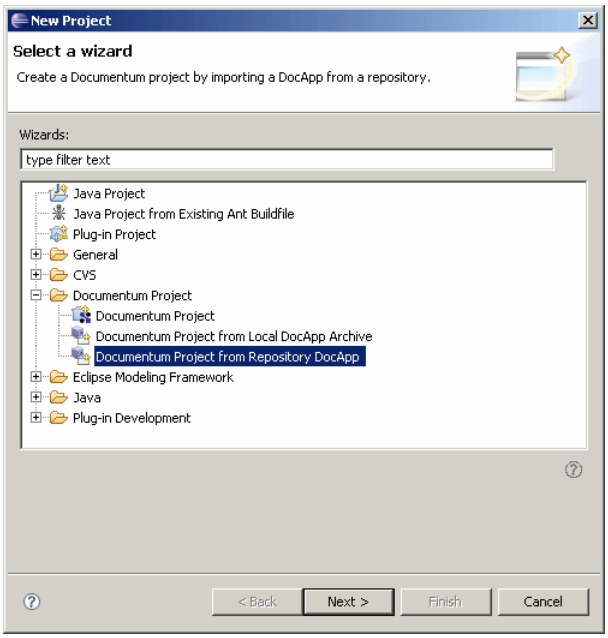
Composer lets you convert a DocApp that is stored in a repository directly into your Composer workspace. This allows you to modify and install your existing DocApps in Composer.

To convert a DocApp to a Composer project:

1. If the DocApp you are about to convert has dependencies on other DocApps or projects, convert those DocApps first. For example, if the DocApp you are converting uses JAR files that are part of another DocApp, you must convert the DocApp containing the JAR files first.

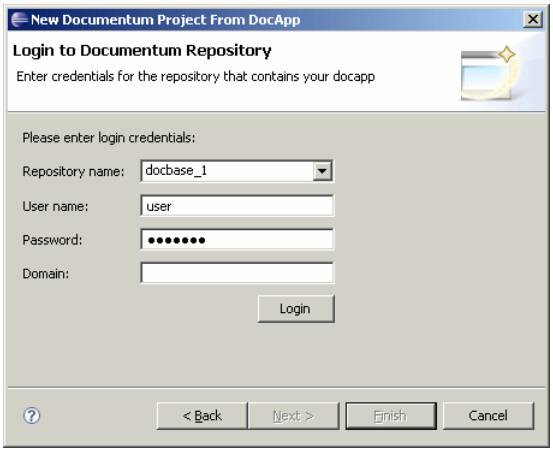
2. Import all Composer projects into the workspace that you want to designate as reference projects. All Documentum supplied reference projects that are needed for your DocApp must be in your workspace before you convert the DocApp. For more information about reference projects, see Composer reference projects, page 29.
3. Open the **New Project** wizard in one of the following ways:
   - From the main menu in Composer select **File** > **New** > **Project**.
   - Right-click in the **Documentum Navigator** view and select **New** > **Project.**

   The **New Project** dialog appears.



4. Double-click the **Documentum** folder to expand it, then select **Documentum Project from Repository DocApp** and click **Next**.

   The **New Documentum Project from DocApp** dialog appears.



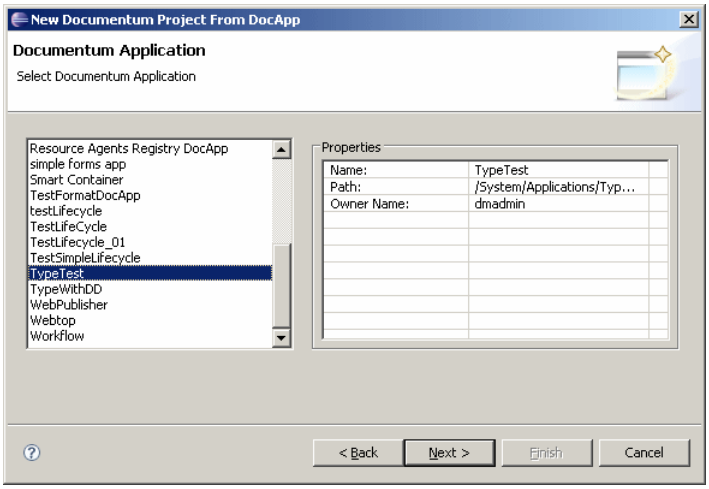
5. Enter your login credentials for the repository that contains the DocApp you want to import, as described in Table 4, page 49, then click **Login**.

Table 4. Repository information

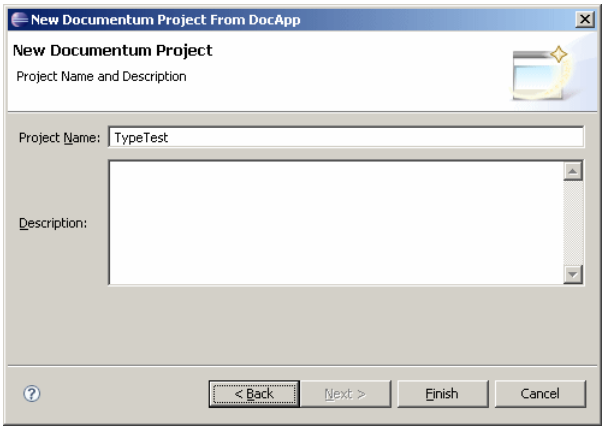
Property	Description
Repository	The name of the repository. Mandatory parameter. You must have SUPERUSER privileges to access the repository.
User name	The user name for the repository.
Password	The password for the repository.
Domain	The domain name of the repository. The domain name only needs to be specified if the repository resides in a different domain than the client from which the repository is accessed.

Composer connects to the repository and verifies your login credentials.

6. After your login credentials have been verified, click **Next**.
   The **Documentum Application** dialog appears.

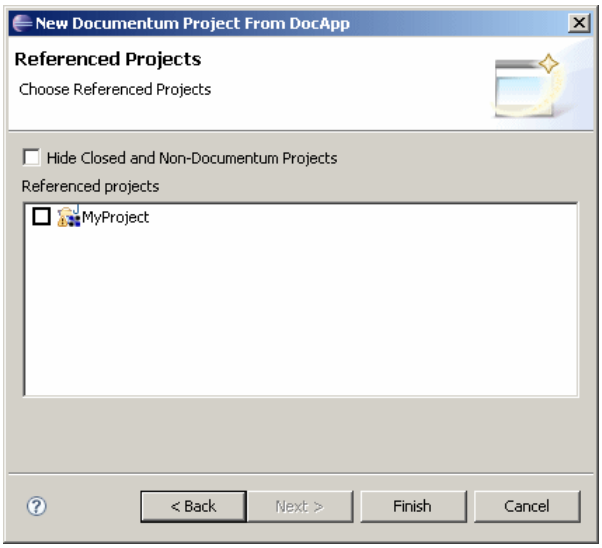


7. Select the DocApp you want to convert from the listbox, then click **Next.**
   The **New Documentum Project** appears.



8. Accept the default project name or enter a new name and an optional description for your project, then click **Next**.

The **Referenced Projects** dialog appears.



9. Select the projects that your new project needs to reference and click **Finish**.

   Composer imports the DocApp and creates a project. The new project is displayed in the **Documentum Navigator** view.

Converting a DocApp archive to a Composer project

A DocApp archive is a mobile version of a DocApp that was archived using Documentum Application Builder (DAB). To convert a DocApp archive to a Composer project, Composer installs it to a target repository and creates a new project from the DocApp archive.

Preparing for DocApp archive conversion

Before you start converting your DocApp archive into Composer, make sure you meet the following requirements:

- The target repository for the DocApp archive should be a clean repository, meaning it should not contain any remnant DocApps or artifacts. This repository should be the repository where you plan on deploying future changes to the resulting Composer project.
- Verify that the DocApp archive you are converting is at least version 5.3. You cannot convert DocApp archives prior to version 5.3. If your DocApp archive is an earlier version than 5.3, you must first upgrade the DocApp archive to version 5.3.
- Verify that the connection broker that is configured in your dfc.properties file points to the target repository. For more information about configuring the connection broker, see Configuring the connection broker, page 19.

- Verify that you have SUPERUSER access for the target repository.
- If the DocApp archive you want to convert depends on other DocApps, you need to convert those DocApps and ensure that the resulting Composer project is in your workspace. If your DocApp archive depends on other reference projects, you need to import the reference projects into your workspace as well. See Composer reference projects, page 29 for more information.

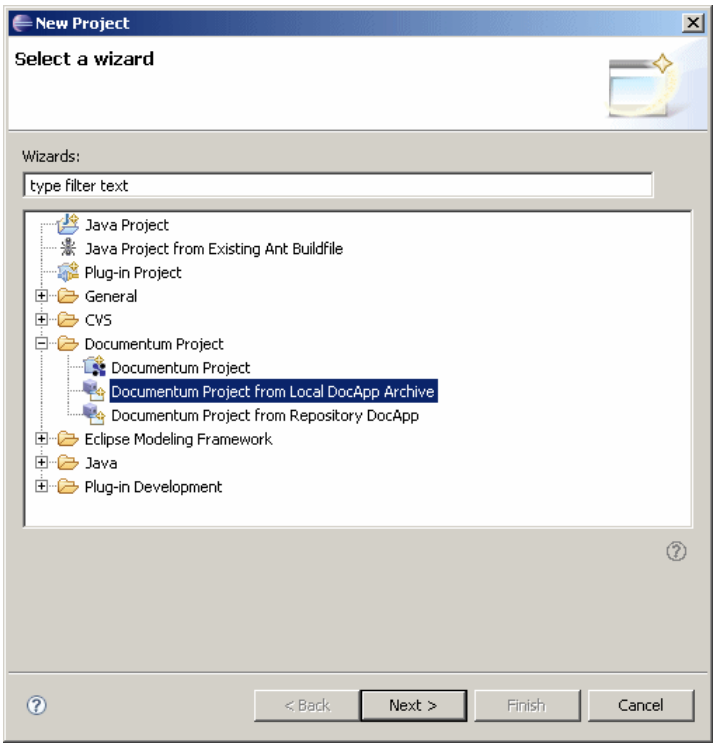
Converting a DocApp archive

Composer lets you convert a DocApp archive into a Composer project. A DocApp archive is a DocApp that was archived with Documentum Application Builder (DAB).

To convert a DocApp archive:

1. Unzip the DocApp archive to a folder on your local machine or a network drive.
2. Open the **New Project** wizard in one of the following ways:
   - From the main menu in Composer select **File** > **New** > **Project**.
   - Right-click in the **Documentum Navigator** view and select **New** > **Project.**

   The **New Project** dialog appears.
3. Double-click the Documentum folder to expand it, then select **Documentum Project from Local DocApp Archive** and click **Next**.



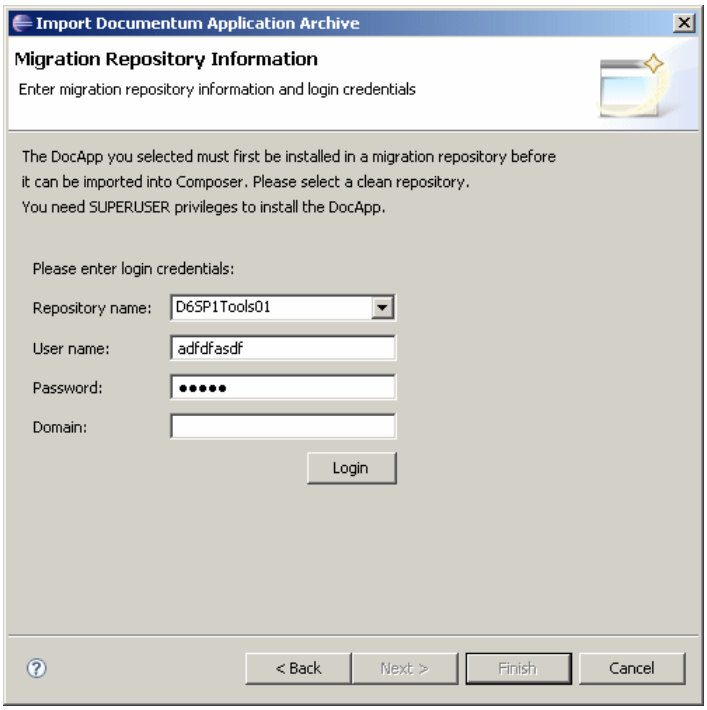
The **Import Documentum Application Archive** dialog appears.

4. Enter the folder name for the DocApp archive, and a project name, as described in Table 5, page 52, then click **Next**.

Table 5. DocApp archive import properties

Parameter	Description
DocApp Archive folder	The folder that contains the unzipped DocApp archive. Mandatory parameter. Type the path name or click **Browse** to search for the folder.
Project name	The name of the project into which the DocApp is imported. Mandatory parameter. By default, the project name is derived from the DocApp name. You can either accept the default name or enter a new name for the project.

The **Migration Repository Information** dialog appears.



5. Enter the information for the target repository, as described in Table 6, page 52.

Table 6. Migration repository information

Parameter	Description
Repository	The name of the target repository. Mandatory parameter. The target repository must be a clean repository. It is used to install the DocApp archive before it is imported into Composer as a new project. You must have SUPERUSER privileges to access the target repository.
User name	The user name for the target repository.

Parameter	Description
Password	The password for the target repository.
Domain	The domain name of the target repository. The domain name only needs to be specified if the target repository resides in a different domain than the client from which the repository is accessed.

6. After you have entered the target repository name and your login credentials, click **Login**. If your login credentials are valid, the **Next** button becomes available. Click **Next**.
7. Select any projects that you want to designate as reference projects and click **Finish** to start the conversion process.

Composer creates a project from the DocApp and the project is displayed in the **Documentum Navigator** view.

Post-conversion tasks

After Composer has converted the DocApp or DocApp archive into a Composer project, you should do the following:

- Review and correct any validation warnings and errors that occurred during the conversion.
- Verify that all artifacts contained in the DocApp were converted and appear in the Composer project.
- Review pre-install and post-install scripts.

  Certain scripts may have to be modified to avoid artifact duplication or conflicts. For example, Composer creates install parameters for users. These users must either be created by the pre-install script or must already exist in the target repository where the project is going to be installed.

Converting TaskSpace applications

Composer provides a separate plug-in to convert Documentum TaskSpace applications from earlier Documentum releases by packaging them in a DAR file, so they can be deployed in a release 6 or later repository.

Documentum TaskSpace is a separate application that provides a user interface optimized for task processing and document retrieval. TaskSpace, in combination with the rest of the Documentum Process Suite, EMC Captiva, and the Documentum repository, delivers an end-to-end solution for managing high-volume Transactional Content Management (TCM) applications.

Converting a TaskSpace application to a Composer project

Importing a TaskSpace application involves downloading, importing, and referencing the TCM reference project, creating a new project, and then importing the TaskSpace application. The TCM Reference Project includes required TaskSpace resources that you need to import TaskSpace applications correctly in Composer.

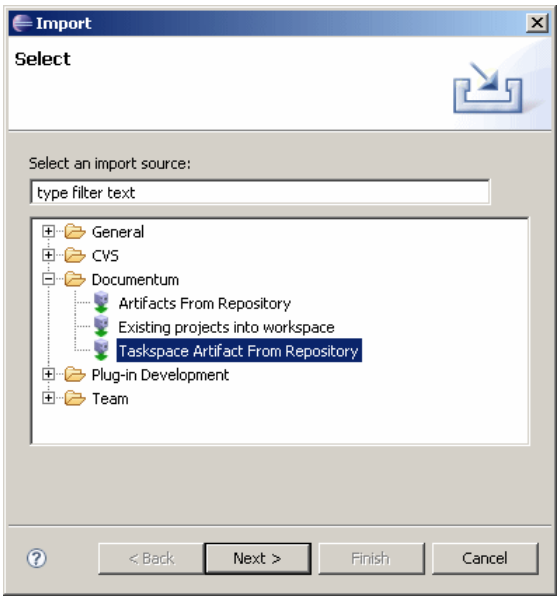
The following rules apply when importing TaskSpace artifacts into Composer:

- If you modify a TaskSpace artifact in Composer, these changes are overwritten the next time the artifact is imported from the repository. You must modify TaskSpace artifacts in Documentum TaskSpace.
- If you add a new artifact to a TaskSpace project in Composer, the artifact is preserved the next time the TaskSpace application is imported from the repository.

To convert a TaskSpace application to a Composer project:

1. Download the TCMReferenceProject.zip file from the EMC software download site. Go to https://emc.subscribenet.com/control/dctm/index to find the Documentum TaskSpace download area.
2. Import the TCMReferenceProject.zip file into your Composer workspace by following the instructions in Importing a project, page 27.
3. Create a new Composer project, as described in Creating a new project, page 25. When prompted, select the **TCMReferenceProject** checkbox to designate it as a reference project.
4. Right-click the new project that you just created in the Documentum Navigator and select **Import** from the drop-down menu.

   The **Import** dialog appears.



5. Expand the **Documentum** folder, select **Taskspace Artifact From Repository**, then click **Next**.

The **Project Selection and Repository Login** dialog appears.

6. Enter the project and login information, as described in Table 7, page 55.

Table 7. TaskSpace project and repository information

Parameter	Description
Project name	The name of the TaskSpace project you created in step 1.
Repository name	The name of the repository that contains the TaskSpace artifacts to be imported.
User name	The name used to login to the repository that contains the TaskSpace artifacts to be imported.
Password	The password used to login to the repository that contains the TaskSpace artifacts to be imported.
Domain	The domain name of the repository. The domain name only needs to be specified if the repository resides in a different domain than the client from which the repository is accessed.

7. Click **Login**.
   The **Artifact Selection** dialog appears.
8. Select the TaskSpace artifacts from the **Choose From Available Artifacts** list, then click **Add**.
9. When you are done selecting TaskSpace artifacts, click **Finish** to import the artifacts from the repository. The artifacts are imported into the TaskSpace project.

Building a TaskSpace application

Building a TaskSpace application is similar to building any other Composer project. If you have the **Project > Build Automatically** option turned on, you can obtain the **<project>.dar** file from the **...\<workspace>\<project>\bin-dar** directory of the Composer workspace. If you have the **Project > Build Automatically** option turned off, complete the following steps:

To build a TaskSpace application:

1. Right-click the TaskSpace project you want to build and select **Build Project** from the drop-down list.
   Composer builds the TaskSpace project and generates a **<project>.dar** file in the **...\<workspace>\<project>\bin-dar** directory, where <workspace> is the name of your workspace and <project> is the name of your project.

Note: You can also build a DAR file using Ant tasks and headless Composer. For more information about building a project using Ant tasks, see Generating a DAR file, page 210.

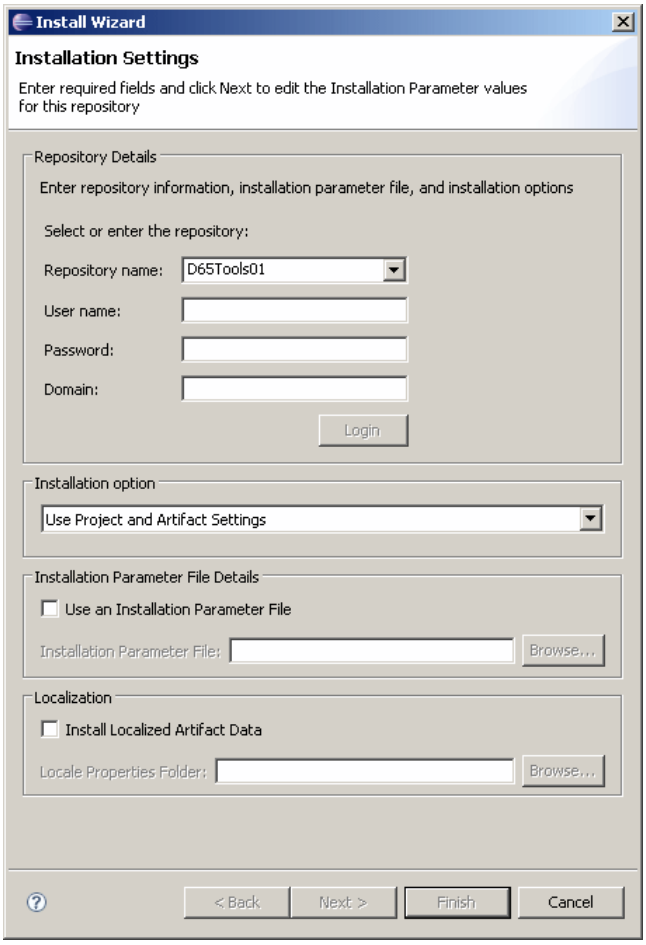
Installing a TaskSpace application

Similar to installing a Composer project, you also use the **Install Documentum Project** option to install a TaskSpace application.

To install a TaskSpace application:

1. In the Documentum Project Navigator view, right-click the TaskSpace application you want to install and select **Install Documentum Project ...** from the drop-down menu.

   The **Installation Settings** dialog appears.



2. Enter the installation information, as described in Table 8, page 56.

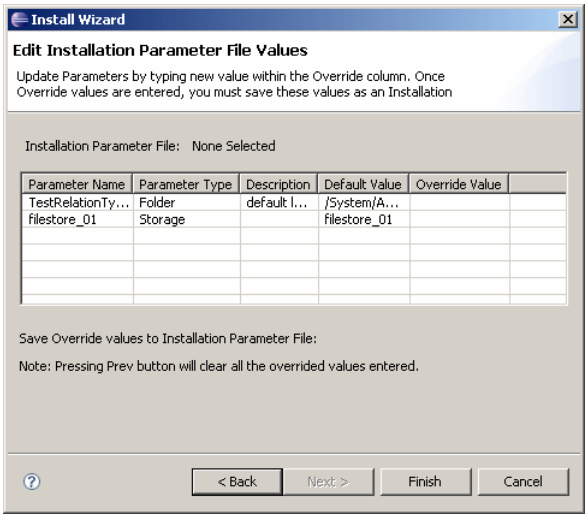
Table 8. TaskSpace install parameter information

Install Parameter	Description
Repository	The name of the installation repository. Mandatory parameter. Type the repository name or select a repository from the drop-down list. You must have SUPERUSER privileges to access the repository.
User name	The login user name for the repository.

Install Parameter	Description
Password	The login password for the repository.
Domain	The domain name of the repository. The domain name only needs to be specified if the repository resides in a different domain than the client from which the repository is accessed.
Install options	Specifies how the TaskSpace application is installed in the repository. Select **Version** from the drop-down list. If a TaskSpace application already exists in the target repository, the installation process overwrites and versions the existing artifacts.
Use an Installation Parameter File	Do not select this option.
Install Localized Artifact Data	Do not select this option.

3. Click **Next**.

   The **Edit Installation Parameter File Values** dialog appears.



   The installation parameter table lists the name, type, and default value for each installation parameter. Do not change any of the parameter values.

4. Click **Finish** to install the TaskSpace application in the repository.

Chapter 4

Managing Web Services

This chapter contains the following topics:

Web services

On a high level, a Web service is a software system designed to support interoperable machine-to-machine interaction over a network. Web services are often Web APIs that can be accessed over a network, such as the Internet, and executed on a remote system hosting the requested services.

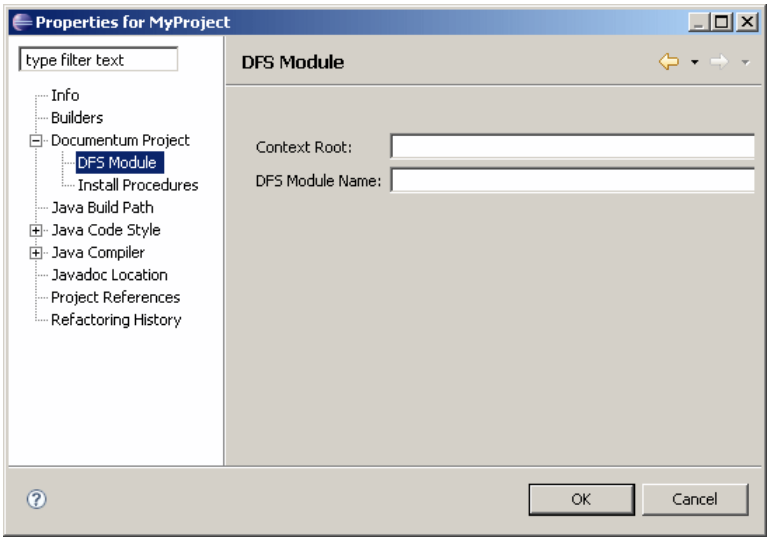
Composer supports Web services by providing an EMC Documentum Foundation Services (DFS) registry plug-in. The plug-in allows users to connect to a Web services registry, import WSDLs to create a Java client library, create services, and export the services to an EAR file. Composer includes a DFS Builder and a DFS Services Library for each new Documentum project. The DFS Builder and DFS Service Library can be configured in a project's property settings.

Configuring DFS module options

You can configure the DFS context root and module name for a project in the **DFS Module** dialog.

To configure the DFS module options:

1. Right-click the project and select **Properties** from the drop-down list.
   The **Properties** dialog appears (Figure 2, page 38).
2. Expand **Documentum Project** and select **DFS Module**.
   The **DFS Module** dialog appears.



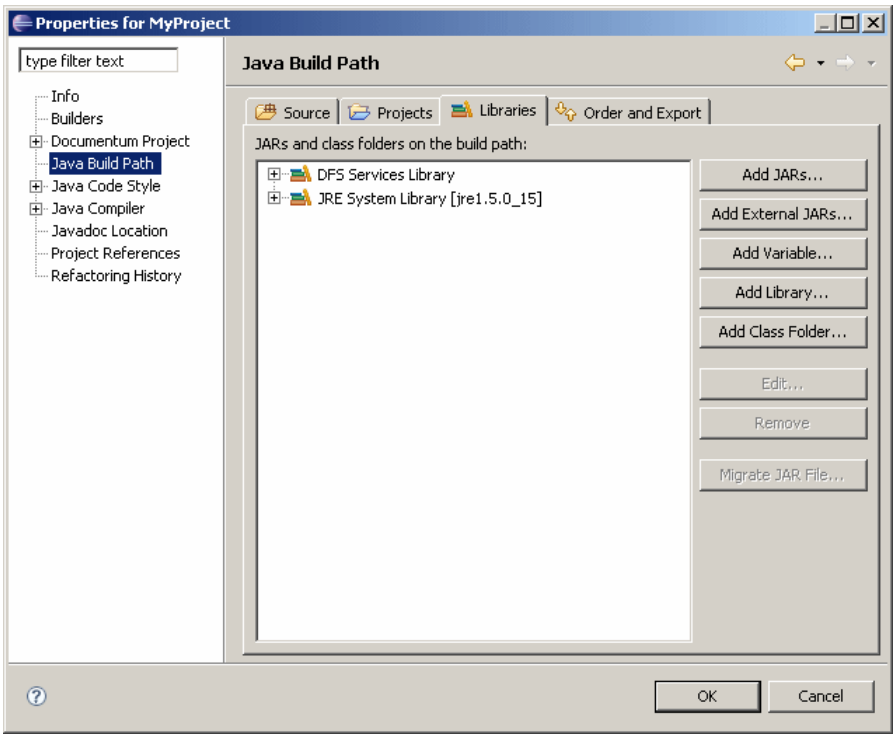
3. Enter the DFS context root and module name in the **Context Root** and **DFS Module Name** fields.
4. Click OK.

Configuring the DFS services library

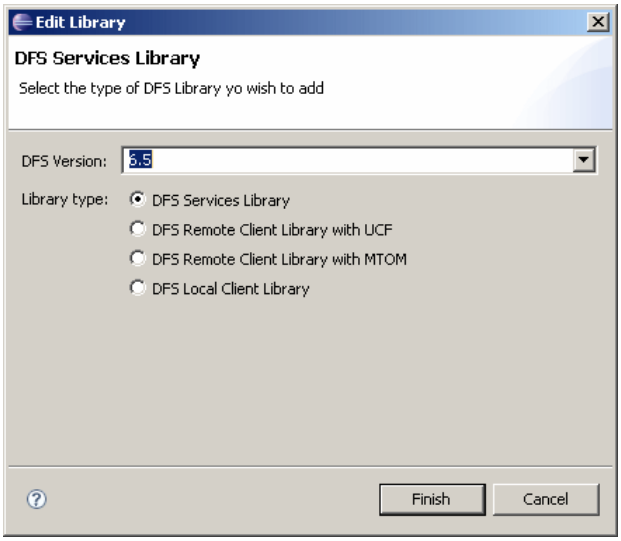
Composer lets you select the DFS service library you want to use for a project. The DFS service library is configured in the project properties. By default, Composer is shipped with one DFS services library, but can support multiple DFS services libraries.

To configure the DFS services library:

1. Right-click the project and select **Properties** from the drop-down list.
   The **Properties** dialog appears (Figure 2, page 38).
2. Select **Java Build Path**.
   The **Java Build Path** dialog appears.



3. Click the **Libraries** tab, select **DFS Services Library** from the list box and click **Edit**. The **DFS Services Library** dialog appears.



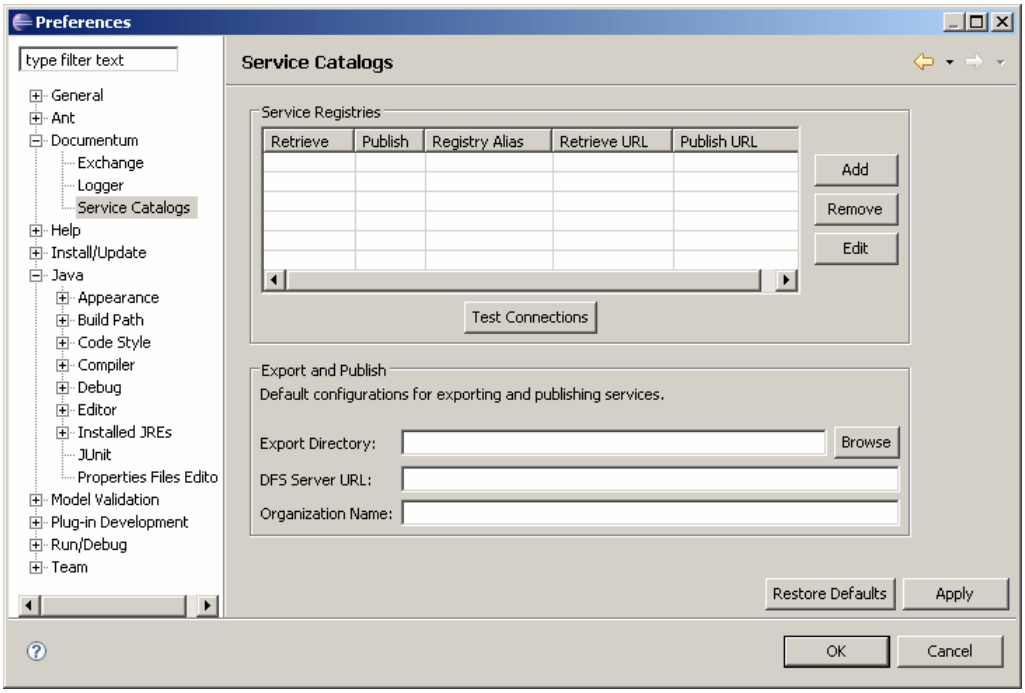
4. Select the type of DFS library you wish to add, then click **Finish**.

Configuring catalog services

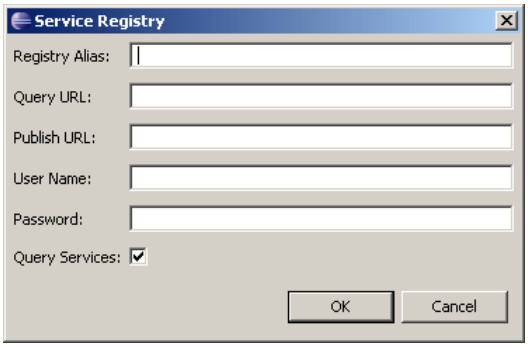
The catalog services connection options are configured in the **Preferences** dialog.

To configure catalog services:

1. In the Composer main menu, select **Window > Preferences ...**
   The **Preferences** dialog appears.
2. In the **Preferences** list, double-click **Documentum** and select **Catalog Services**.
   The **Catalog Services** dialog appears.



   The **Services Catalogs** table lists the services that are currently configured.
3. To configure another service registry, click **Add**.
   The **Service Registry** dialog appears.



4. Enter the configuration parameters for the service registry, as described in Table 9, page 62.

Table 9. Service registry options

Parameter	Description
Registry Alias	A string specifying a name for the service registry.

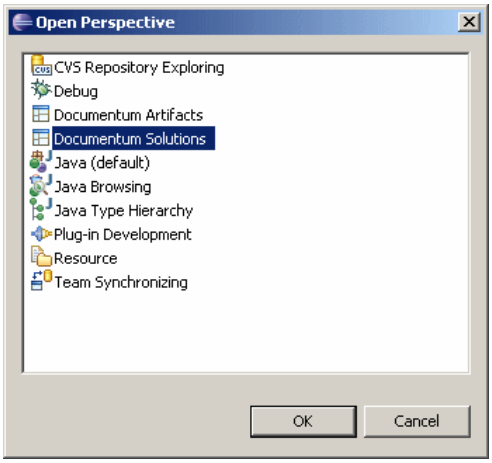
Parameter	Description
Query URL	A string specifying the URL of the server that hosts the service. The URL must have the format http://<domain>:<port>/catalog/inquiry.
Publish URL	A string specifying the URL of the server to which the service is published. The URL must have the format http://<domain>:<port>/catalog/publish.
User Name	The login name for the server hosting the service.
Password	The password for the server hosting the service.

Viewing Web services

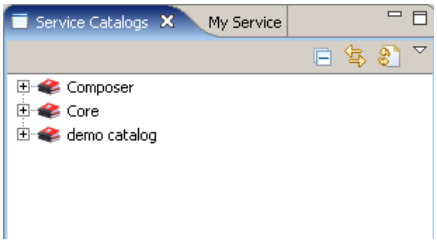
In Composer all Web services and related actions are displayed in a different perspective, the **Documentum Solutions** perspective.

To view available Web services:

1. In the Composer main menu, select **Window > Open Perspective > Other ...**
   The **Open Perspective** dialog appears.



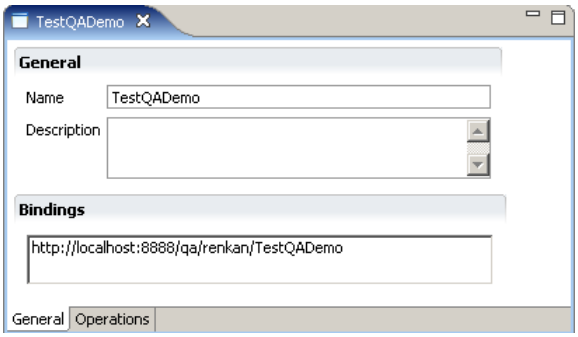
2. Select **Documentum Solutions** and click **OK**.
   The **Service Catalog** tab and the **My Service** tab display.



The **Service Catalog** tab shows all service catalogs and services on the server that you configured. The **MyService** tab shows the services you imported or created. If you installed the Documentum Services Catalog Repository, Documentum services are not automatically published as part of that installation. Services must be published to the catalog before you can view them in Composer. If no services are published, Composer does not display any services.

3. To view the Web services, expand the catalog, then double-click the service to display the service details.

   The service details appear with the **General** tab selected.



4. Click the **Operations** tab to view the service methods.

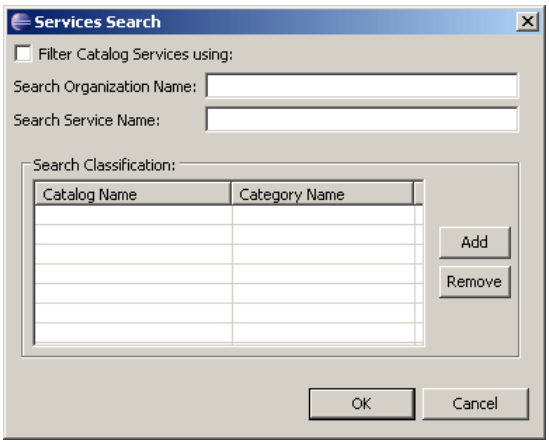
Filtering services

You can specify which services you want to display in the **Service Catalog** tab by using the service filter.

To filter services:

1. In the **Documentum Solutions** perspective, click the search icon ( ) below the **Catalog Services** tab.

   The **Services Search** dialog appears.



2. Select **Filter Catalog Services using:** and enter your search criteria. You can filter by **Organization Name**, **Service Name**, **Catalog Name**, and **Category Name**.

3. Click **OK**.

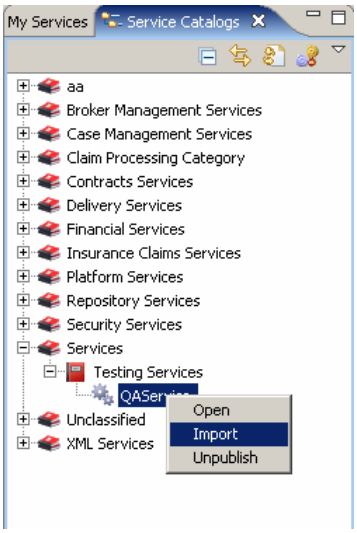
   The services that match the filter criteria are displayed in the **Catalog Services** view.

Generating a client proxy

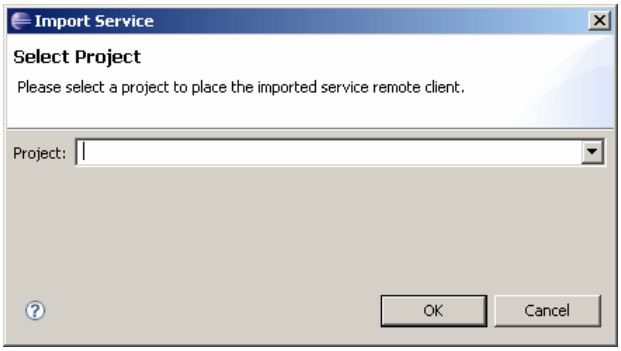
The **Import** option in the **Service Catalog** tab lets you generate the client proxy of a service and make it available in a Composer project.

To generate the client proxy of a service:

1. In the Service Catalog view, right-click the service you want to generate the client proxy and select **Import**.

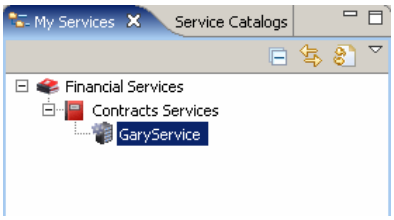


   The **Import Service** dialog appears.

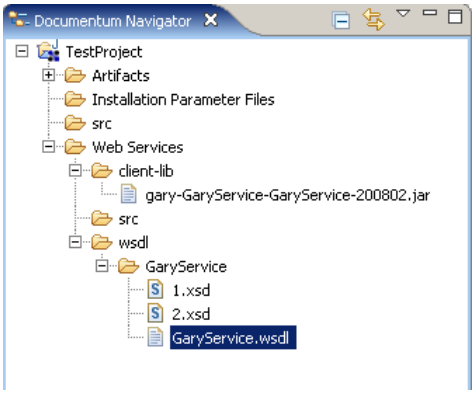


2. Enter the project in which you want to generate the client proxy or select a project from the drop-down list, then click **OK**.

   Composer imports the client proxy into the project. The service name appears in the My Services tab.



The JAR file and the WSDL of the service appear in the **Web Services** folder of the project in the **Documentum Navigator** view.



Consuming a service

Consuming a service requires importing the client proxy of a service, as described in Generating a client proxy , page 65, and creating the code that calls the service.

The following code example describes how to call a service. The only custom code in the is example is the **try** block that is highlighted in bold.

```
package com.acme.loanapp.services;


import com.emc.documentum.fs.datamodel.core.context.RepositoryIdentity;
import com.emc.documentum.fs.rt.context.ContextFactory;
import com.emc.documentum.fs.rt.context.IServiceContext;
import com.emc.documentum.fs.rt.context.ServiceFactory;
import com.emc.services.ws.client.soap.*;

public class AcmeLoanServiceOrchestration
{

    public static void main(String [ ] args)
    {
     RepositoryIdentity m_theId = new RepositoryIdentity();

     m_theId.setRepositoryName("D65Docbase");
     m_theId.setUserName("dfsuser");
     m_theId.setPassword("dfs");

     //completion point 'get context'
     IServiceContext context = ContextFactory.getInstance().newContext();
     ServiceFactory sf = ServiceFactory.getInstance();
```

```
     context.addIdentity(m_theId);

  try {
  //completion point 'instantiate services'
   IWorkflowService qSvc = sf.getRemoteService(IWorkflowService.class,
         context, "core", "http://localhost:9080/services");
   qSvc.start("ProcessLoanApplication");
  }
    catch (Exception e)
  {
   System.out.println("An exception has occured: " + e);
  }
    }
}
```

Creating a service

You can create a service from a Java file or generate a service from the WSDL file of a client proxy.

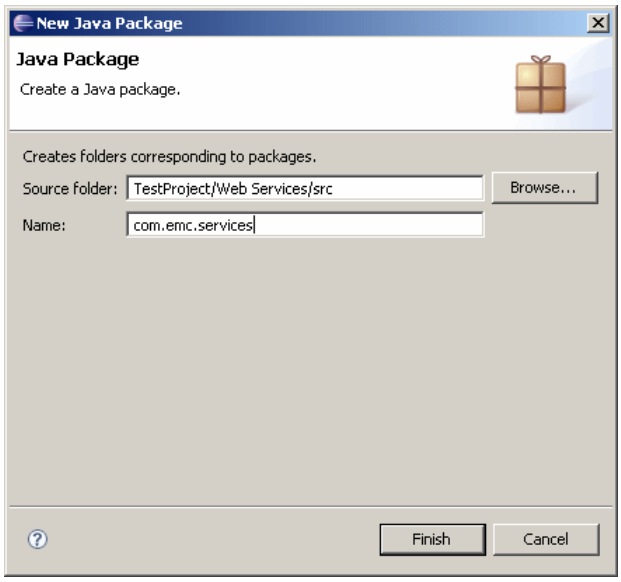
Creating a service from a Java file

You must switch to the **Package Explorer** view to create a new service.

Note: This guide only describes how to create the Java file for the service, not how to develop a service. For more information about developing DFS services, see the *EMC Documentum Documentum Foundation Services Development Guide Version 6.5.*

To create a service from a Java file:

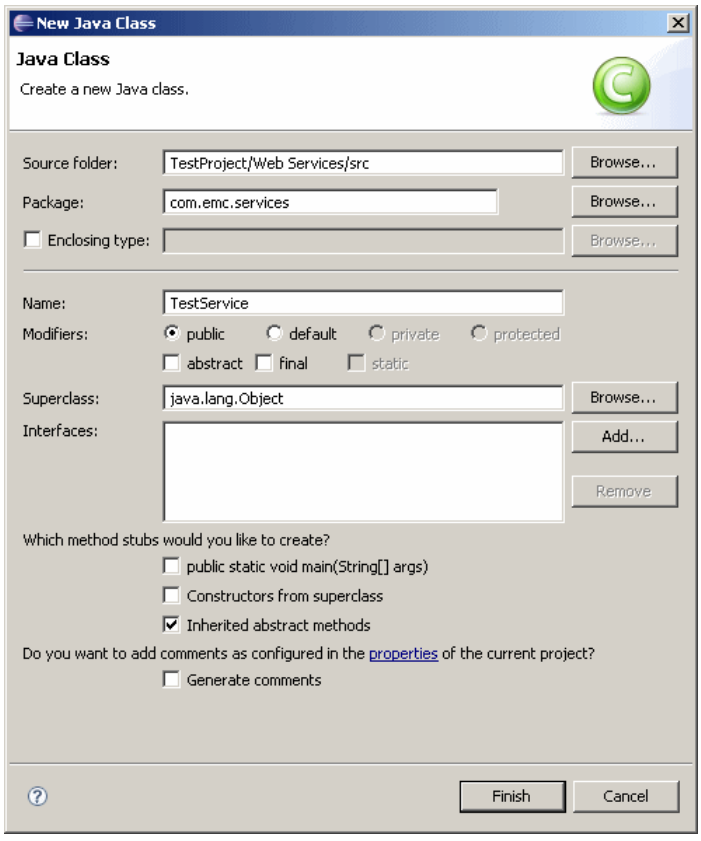
1. Change to the **Package Explorer** view in Composer by selecting **Window** > **Show View** > **Package Explorer**.
2. Create a Java package for your service in the **Web Service/src** directory:
   a. Right-click the **Web Service/src** directory and select **New > Package**.
      The **New Java Package** dialog appears.



b. Enter a name for your Java package, for example **com.emc.services**, then click **Finish**.

3. Create a Java class:

a. Right-click the Java package you just created and select **New > Class**.
The **New Java Class** dialog appears.



b. Enter a name for your Java class, for example **TestService** and select any methods you want to include, then click **Finish**. The Java file appears in the workspace.

c. Write the code that specifies your service. For more information about developing DFS services, see *EMC Documentum Documentum Foundation Services Development Guide Version 6.5*.

d. Save your changes.

You new service appears on the **MyServices** tab in the **Documentum Solutions** perspective under **Unclassified**.

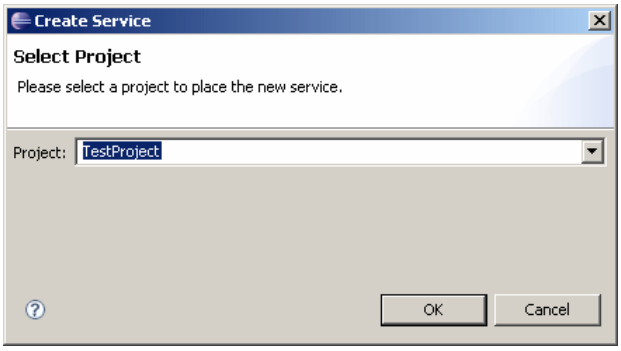
Creating a service from a WSDL

You can create a service directly from a WSDL file.

To create a service from a WSDL:

1. Navigate to the Web Services folder of the project in the **Documentum Navigator** view.
2. Right-click the WSDL from which you want to generate a service and select **Create Service** from the drop-down menu.

   The **Create Service** dialog appears.



3. Enter the name of the project in which you want Composer to create the service or select a project from the drop-down list, then click **OK**.

   The Java file for the service appears in the **/src** directory of the project's **Web Services** folder.

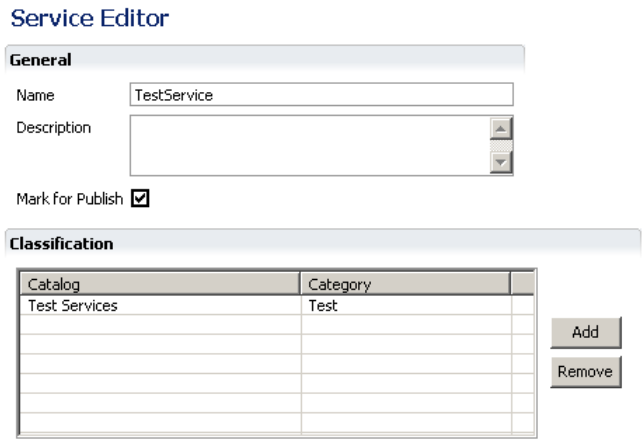
Modifying catalog and category information

When you first create a service, as described in Creating a service, page 67, the services appears under the **Unclassified** catalog and category in the **My Services** tab. You can modify the catalog name and category for your service in the **Service Editor**.

To create a service catalog and category:

1. Right-click the service in the **My Services** tab and select **Open**.

   The **Service Editor** appears.



2. Enter the service information in the **General** and **Classification** sections, as described in Table 10, page 70.

Table 10. Web service information

Parameter	Description
General	
Name	The name of the service.
Description	A description of the service.
Mark for Publish	Specifies whether this service is ready to be published. This option is enabled by default.
Classification	
Catalog	The name of the catalog. Click **Add** to add a new catalog entry, then select the field in the **Catalog** column to modify it.
Category	The name of the catalog category. Click **Add** to add a new category entry, then select the field in the **Category** column to modify it.

3. Save your changes.
   The new or modified catalog name and category appear in the **My Services** tab.

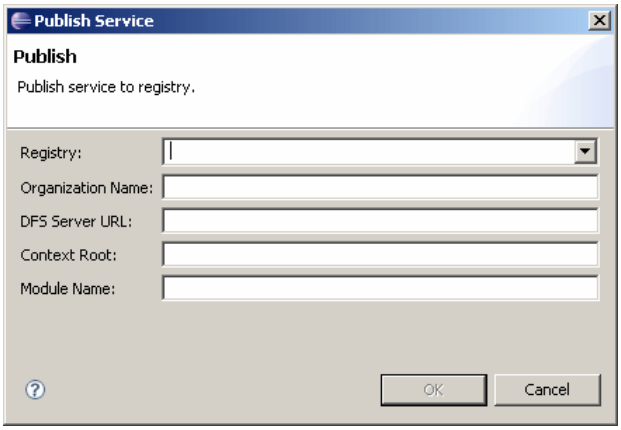
Publishing a service

Use the **Publish Service** dialog to publish a service to a registry.

To publish a service:

1. Switch to the **Documentum Solutions** perspective and locate the service you want to publish in the **My Services** tab.

2. Right-click the service and select **Publish** from the drop-down list.
   The **Publish Service** dialog appears.



3. Enter the publishing information, as described in Table 11, page 71, then click **OK**.

Table 11. Publish service information

Parameter	Description
Registry	The alias of the registry to which the service is published. Select the registry from the drop-down list.
Organization Name	The name of the organization where the service resides.
DFS Server URL	The URL of the DFS server.
Context Root	Specifies the root of the service address. For example, in the URL https://127.0.0.1:7001/services/core, "services" signifies the context root.
Module Name	Specifies the name of the service module. For example, in the URL https://127.0.0.1:7001/services/core, "core" signifies the module name.

4. Click the **Service Catalogs** tab and refresh the view.
   The service appears in the **Service Catalogs** list if the service was published successfully.

Unpublishing a service

When you unpublish a service, it is no longer available on the DFS server and does not show up in registry queries.

To unpublish a service:

1. In the **Service Catalogs** view, right-click the service and select **Unpublish**.
   The **Unpublish Services** dialog appears.
2. Click **OK** to unpublish the service.
   The service does no longer appear in the **Service Catalogs** view.

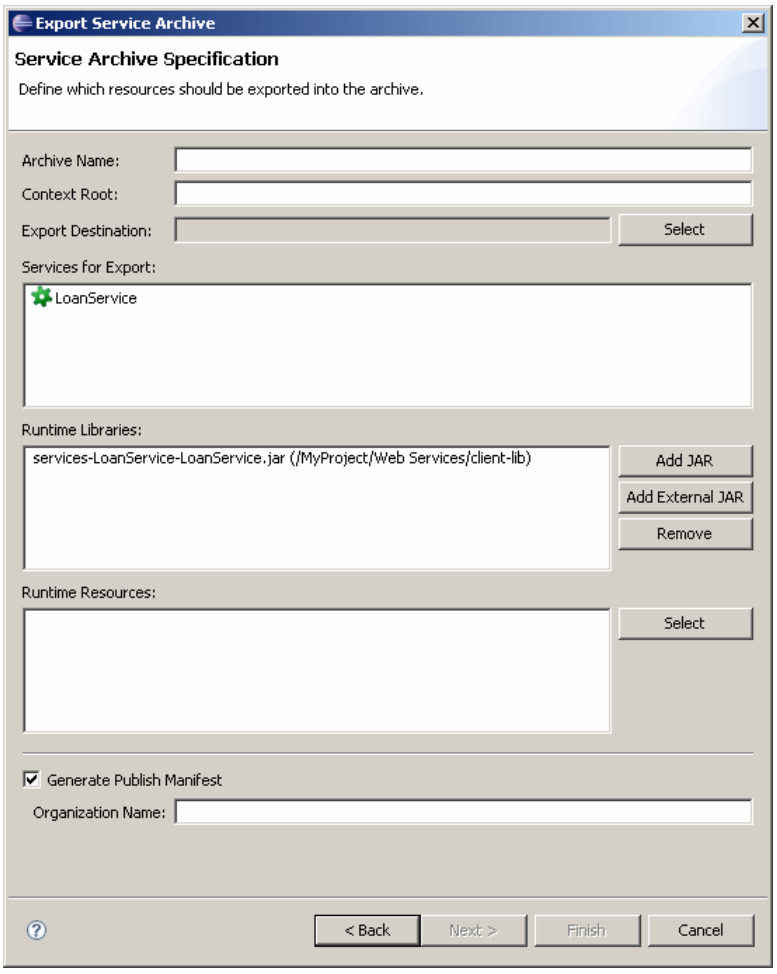
Exporting a service

When you export a service, Composer generates an archive EAR file, a JAR containing runtime resources, and an optional manifest XML file.

To export a service:

1. Right-click the project that contains your service either in the **Documentum Navigator** or **Package Explorer** view.
2. Select **Export** > **Documentum** > **Export Service Archive**, then click Next.

   The **Export Service Archive** dialog appears.



3. Enter the export service information, as described in Table 12, page 72, then click **Finish**.

Table 12. Export service information

Parameter	Description
Archive Name	The name for the archive EAR file. The file has the format *<archive name>.ear*.

Parameter	Description
Context Root	Specifies the root of the service address. For example, in the URL https://127.0.0.1:7001/services/core, "services" signifies the context root.
Export Destination	The location on the local machine or network drive where the EAR file is saved. Click **Select** and browse for a location to store the EAR file.
Services for Export	Select the service to be exported.
Runtime Libraries	The JARs that are included in and exported with the EAR file. Click **Add** to include JAR files from your Composer workspace. Click **Add External JAR** to include JAR files from the local file system.
Runtime Resources	The runtime resources that are included in and exported with the archive EAR file. The runtime resources are packaged in a JAR file. Click **Select** and check the resource to be exported.
Generate Publish Manifest	Select this option to export a manifest XML file with the EAR file. The manifest file has the format *<archive name>-publish-manifest.xml*.
Organization Name	The name of the organization that created the service.

Composer creates the archive EAR file, runtime resources JAR file, and the manifest file in the export destination.

Deploying a service

After you have exported the service and generated an EAR file, you can deploy the service by copying the EAR file to your DFS server.

To deploy a service:

1. Generate an EAR file for the service you want to deploy, as described in Exporting a service, page 72.
2. Copy the EAR file to the services directory on your DFS server.

Chapter 5

Managing Alias Sets

This chapter contains the following topics:

Alias, alias values, and alias sets

Many aspects of Documentum applications involve references to specific users, groups, permission sets, and repository cabinets and folders. Instead of referencing the actual user, group, cabinet or folder name, you can assign an alias, a symbolic name. The symbolic name is called the alias name. The actual value is called the alias value. A collection of aliases is called an alias set.

Creating an alias set

Use the **Alias Set** editor to create a new alias set.

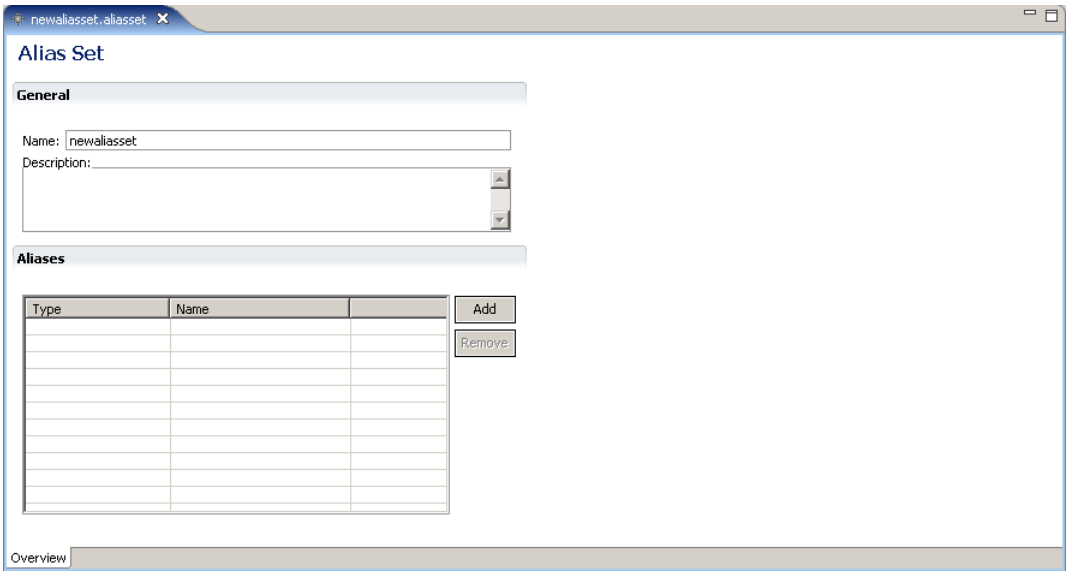
To create an alias set:

1. Open the **Select a wizard** dialog in one of the following ways:
   - From the Composer menu, select **File > New > Other**.
   - In your Composer project, expand the **Artifacts** folder and right-click **Alias Set**. Select **New > Other.**

   The **Select a wizard** dialog appears.
2. Select **Documentum > Alias Set**, then click **Next**.

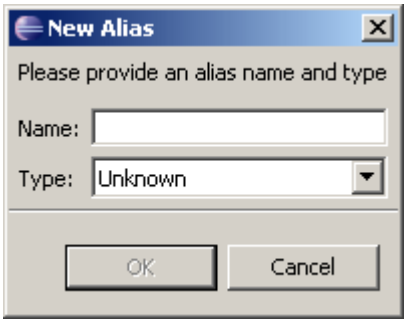
   The **New Documentum Artifact - Name and Location** dialog appears.
3. Enter the folder path and name of the project for which you want to create an alias set in the **Folder:** field, or click **Browse** to select the project from a folder list.
4. Enter a file name for the alias set in the **Artifact name:** field, then click **Finish**.

   The **Alias Set** editor appears.



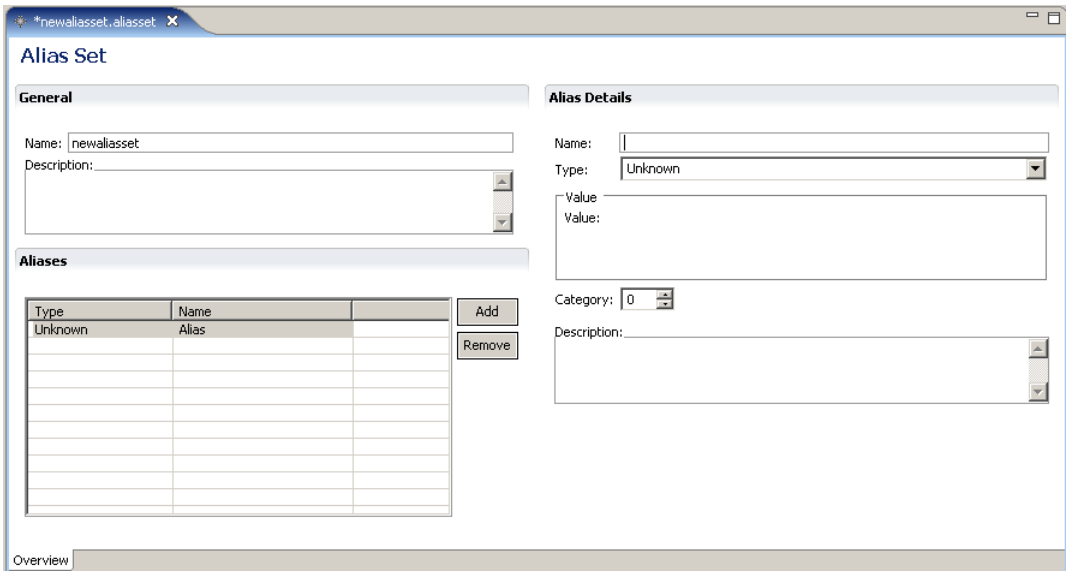
5. Enter a name for the alias set in the **Name** field and an optional description in the **Description** field.
6. Click **Add** in the **Aliases** section to create one or more aliases that make up the alias set.

   The **New Alias** dialog appears.



7. Enter a name for the new alias in the **Name:** field and select a type from the **Type:** drop-down list. You can create an alias for the following types:
   - Unknown
   - User
   - Group
   - User or Group
   - Cabinet Path
   - Cabinet or Folder Path
   - Permission Set
8. Click **OK** when you are finished.

   The **Alias Details** section appears.



9. Enter the details for the alias in the **Alias Details** section, as described in Table 13, page 77.

Table 13. Alias details

Parameter	Description
Name	A string specifying the name of the alias.
Type	Specifies the type for which this alias is used.
Value	Specifies the parameter and value of the alias. Depending on which alias type you specified in the **Type** field of the **New Alias** dialog, different parameter and value options are available in the **Value** section. For more information about alias values, see Configuring alias values, page 77.
Category	Alias category is a tool for developers to use to organize the aliases in their applications. Documentum software does not use this field.
Description	An optional description of the category.

10. Save your changes.

Configuring alias values

An alias can have different parameters and values, depending on the type of alias that is specified in the **Type** field in the **Alias Details** section. Table 14, page 78 describes the parameters associated with a specific alias type.

Table 14. Alias type values

Alias Type	Value Options	Description
Unknown	No value can be assigned to an unknown alias type.	
User	• Leave It Blank • Parameter	To assign a parameter, click **Parameter**, then click **Select**. The **User Installation Parameter** dialog appears. Select a parameter from the listbox or click **New...** to create a new user parameter.
Group	• Leave It Blank • Parameter • Value	To assign a parameter: • Click **Parameter**, then click **Select**. The **Group Installation Parameter** dialog appears. Select a parameter from the listbox or click **New...** to create a new group parameter. To assign a value: • Click **Value**, then click **Select**. The **Documentum Group Artifact** dialog appears. Select an artifact from the listbox, then click **OK**.
User or Group	• Leave It Blank • Parameter	To assign a parameter, click **Parameter**, then click **Select**. The **Principal (User or Group) Installation Parameter** dialog appears. Select a parameter from the listbox or click **New...** to create a new parameter.
Cabinet Path	• Parameter • Value	To assign a parameter: • Click **Parameter**, then click **Select**. The **Folder Installation Parameter** dialog appears. Select a parameter from the listbox or click **New...** to create a new parameter. To assign a value: • Click **Value**, then click **Select**. The **FolderSubtype Artifact** dialog appears. Select an artifact from the listbox, then click **OK**.

Alias Type	Value Options	Description
Cabinet or Folder Path	• Parameter • Value	To assign a parameter: • Click **Parameter**, then click **Select**. The **Folder Installation Parameter** dialog appears. Select a parameter from the listbox or click **New...** to create a new parameter. To assign a value: • Click **Value**, then click **Select**. The **FolderSubtype Artifact** dialog appears. Select an artifact from the listbox, then click **OK**.
Permission Set	• Parameter • Value	To assign a parameter: • Click **Parameter**, then click **Select**. The **ACL Installation Parameter** dialog appears. Select a parameter from the listbox or click **New...** to create a new parameter. To assign a value: • Click **Value**, then click **Select**. The **Permission Set (ACL) Template Artifact** dialog appears. Select an artifact from the listbox or click **New ...** to create a new artifact..

Chapter 6

Managing Aspects

This chapter contains the following topics:

Aspect modules and aspect types

An aspect module consists of executable business logic and supporting material for an aspect, such as third-party software and documentation. An aspect customizes behavior or records metadata or both for an instance of an object type. An aspect module is comprised of the aspect type definition, the JAR files that contain the implementation classes and the interface classes for the behavior the aspect implements, and any interface classes on which the aspect depends. The module may also include Java libraries and documentation.

Creating an aspect type

Use the **Aspect** editor to create an aspect type.

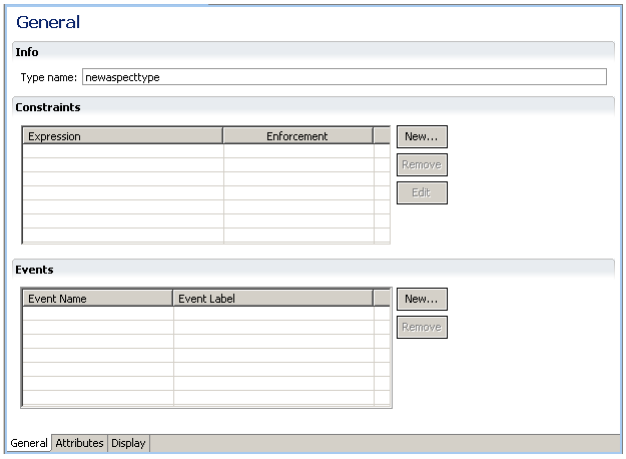
To create an aspect type:

1. Open the **Select a wizard** dialog in one of the following ways:
   - From the Composer menu, select **File > New > Other**.
   - In your Composer project, expand the **Artifacts** folder and right-click **Types**. Select **New > Other.**

   The **Select a wizard** dialog appears.
2. Select **Documentum > Aspect Type**, then click **Next**.
   The **New Documentum Artifact - Name and Location** dialog appears.

3. Enter the folder path and name of the project for which you want to create an aspect type in the **Folder:** field, or click **Browse** to select the project from a folder list.
4. Enter a file name for the aspect type in the **Artifact name:** field, then click **Next**.

   The **Aspect** editor appears with the **General** tab selected.



5. Enter the aspect information in the **Info**, **Constraints**, and **Events** sections, as described in Table 15, page 82.

Table 15. Aspect information on General tab

Property	Description
Info	
Type name	A string specifying the name of the aspect. The aspect type name should be the same as the name of the aspect module that is referencing the aspect. The following rules apply to all aspect names: • A maximum of 27 characters, all lower-case. The Content Server is case-insensitive and stores all type names in lower-case. • The first character must be a letter, the remaining characters can be letters, digits, or underscores • Cannot contain any spaces or punctuation • Cannot end in an underscore (_)
Constraints	Constraints are internal consistency requirements in the form of Docbasic expressions that relate the types attribute values to one another or to constant values.
Expression	The Docbasic expression defining the constraint. Click **New** to create a new expression. For more information about adding constraints, see Configuring constraint expressions, page 83.

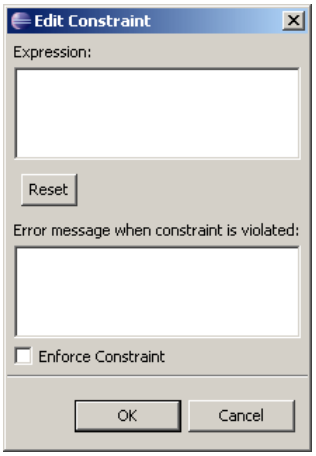
Property	Description
Enforcement	Specifies whether applications should enforce this constraint or not. Click the table cell in the Enforcement column to enable or disable constraint enforcement for the associated expression.The enforcement field can have two values, as follows: • **disabled**: The constraint is disabled • **ApplicationEnforced**: The constraint is enforced by the applications that use this type.
Events	Events are specific actions on objects. You can only create and modify application events, not system events. Click **New** to enter a new event. To edit or remove an event, select the event and click **Edit** or **Remove**, respectively.
Event name	A string specifying the name of the event that is associated with instances of this type.
Event label	A string that specifies the label for the event.

Configuring constraint expressions

Constraints are internal consistency requirements in the form of Docbasic expressions that relate the aspect attribute values to one another or to constant values.

To add a constraint expression for an aspect:

1. Click **Add** in the Constraints section of the Aspect Overview tab in the aspect editor.
   The **Edit Constraint** dialog appears.



2. Type a valid Docbasic constraint expression that resolves to true or false in the **Expression:** text box. The Docbasic expression should resolve to true when the constraint is fulfilled and false when the constraint is violated.

3. Type a message for applications to display when the constraint is violated in the **Error message when constraint is violated:** text box.
4. Check the **Enforce Constraint** checkbox to instruct applications to enforce this constraint or leave it unchecked to not enforce the constraint.
5. Click **OK** to save your changes.

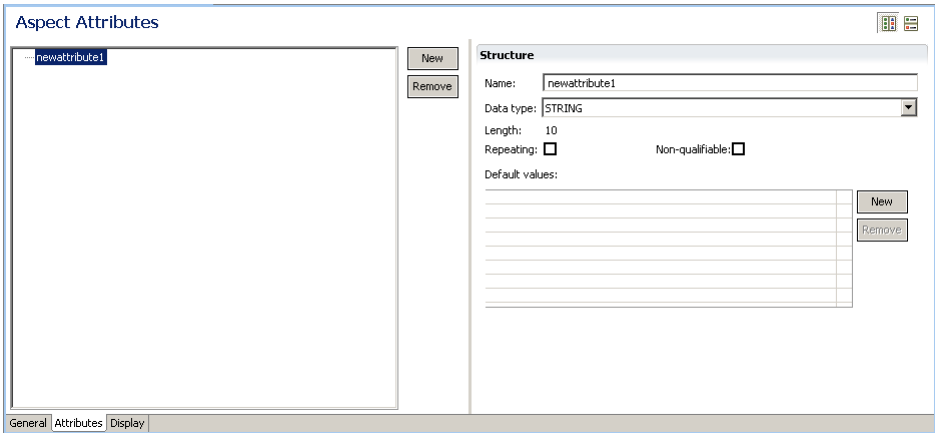
Adding aspect attributes

Aspect attributes are configured in the **Attributes** tab of the aspect editor. An aspect attribute is a property that applies to all aspect of that type. When an aspect is created, its attributes are set to values that describe that particular instance of the aspect type. You must configure attributes for each aspect. If you create an aspect without configuring any attributes, the aspect artifact will not install correctly in the repository and causes the installation of the entire project to fail.

To create an attribute:

1. Click the **Attributes** tab in the aspect editor to display the Attributes view.
2. Click **New** to create a new attribute entry, then select the new attribute entry.

   The **Aspect Attributes** view expands.

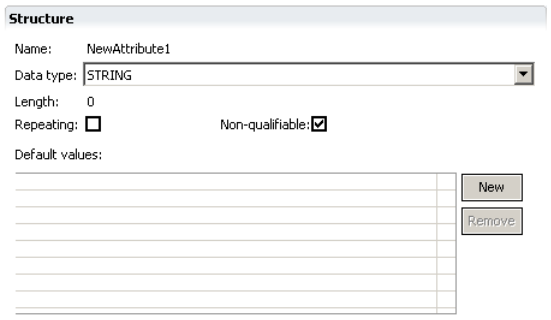


3. Configure the attribute structure, as described in Configuring the aspect attribute structure, page 84.

Configuring the aspect attribute structure

The attribute structure is configured in the **Structure** section of the **Aspect Attributes** view (Figure 3, page 85).

Figure 3. Structure section in Aspect Attributes view



Enter the attribute structure properties, as described in Table 16, page 85.

Table 16. Attribute structure properties

Property	Description
Name	A string specifying the name of the new attribute. The attribute name must use all lowercase letters, cannot begin with dm_, a_, i_, r_, a numeral, space, or single quote, and cannot be named select, from, or where.
Data type	The data type of the new attribute. Select one of the following data types from the drop-down list: • BOOLEAN • INTEGER • STRING • ID • TIME • DOUBLE • UNDEFINED
Length	This parameter only applies to attributes that use the STRING data type. Enter the number of characters for this attribute. The maximum number of characters that you can assign to this attribute depends on the database where you are installing the application.
Repeating	Specifies whether this attribute can have more than one value. Select the checkbox to allow more than one value for this attribute.

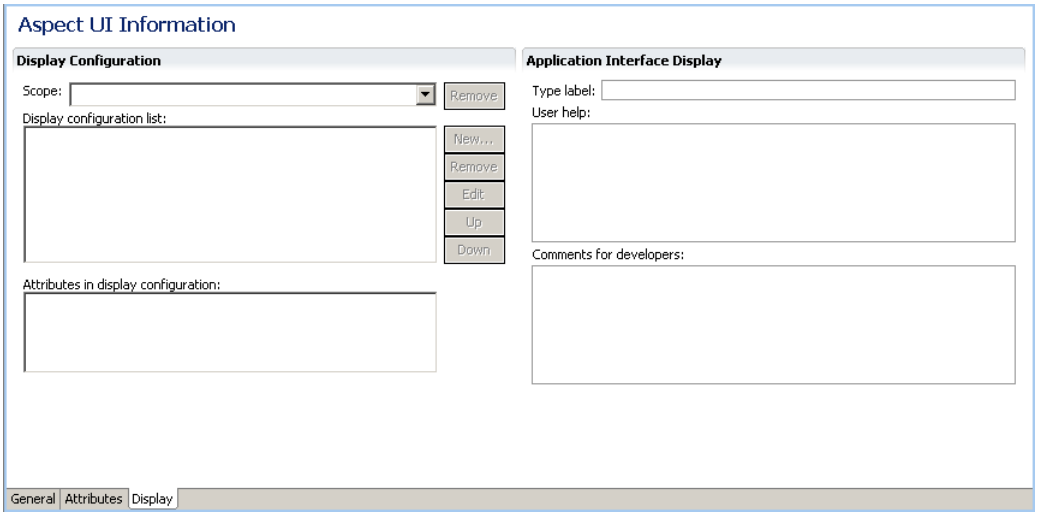
Property	Description
Non-qualifiable	Specifies whether the attribute is qualifiable or non-qualifiable. The properties and values of a non-qualifiable attribute are stored in a serialized format and do not have their own columns in the underlying database tables that represent the object types for which they are defined. Consequently, non-qualifiable attributes cannot be used in queries because they are not exposed in the database.
Default values	Lets you specify one default value for a single-value attribute or multiple default values for a repeating attribute.

Configuring the aspect UI information

The **Aspect UI Information** view lets you specify which aspect attributes are displayed in Documentum clients and custom applications. Ensure that the client application that you are using supports displaying attribute information for aspects.

Note: Webtop does not support displaying attributes for aspects.

Figure 4. Aspect UI information view



To configure one or more attributes to be displayed in clients:

1. Click the **Display** tab in the aspect editor to display the **Aspect UI Information** view (Figure 4, page 86).
2. Enter the aspect UI information as described in Table 17, page 87.

Table 17. Aspect UI information

Property	Description
Display Configuration	
Scope	The name of the application, in which the aspect is displayed. The name of the application must exist in the repository. **Note:** Webtop does not support displaying attributes for aspects.
Display configuration list	Specifies the tab on which the aspect attribute is displayed. You can add, remove, rename, and change the position of a tab, as follows: • Click **New** to add a new tab. The **Display Configuration** dialog appears. For more information about adding a tab to display an attribute in a client application, see Adding a tab , page 87. • To remove a tab, select the tab name in the list, then click **Remove**. • To rename a tab, select the tab name in the list, then click **Rename**. • To change the order in which the tabs are displayed, select the tab name in the list, then click **Up** or **Down** to move the tab to the desired position.
Attributes in display configuration	Lets you modify the attributes that are displayed on a tab.
Application Interface UI	
Type label	A string that the client application displays for this aspect.
User help	Optional description for the aspect that is displayed in the application.
Comments for developers	Optional comments for developers.

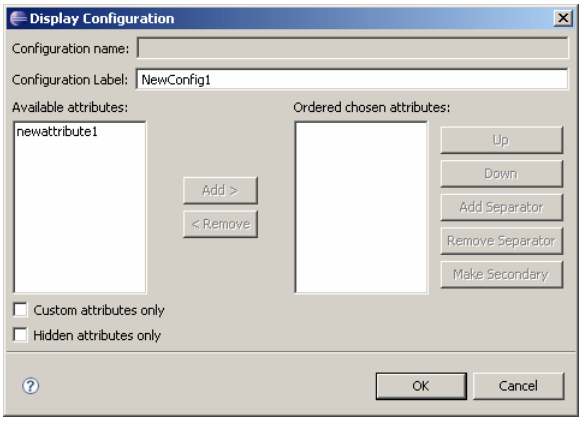
Adding a tab

Use the **Display Configuration** dialog to add a tab to display an attribute.

To add a tab to display an attribute:

1. Click **New** in the **Display** configuration list section of the **Aspect UI Information** view (Figure 13, page 186). A default tab (**NewConfig1)** for the new tab appears in the **Display** configuration list box.

2. Select the default tab and then click **Edit**.
   The **Display Configuration** dialog appears.



3. Configure the tab properties, as described in Table 18, page 88.

Table 18. Tab configuration properties

Tab properties	Description
Configuration name	A string that specifies the internal object name of the tab. You cannot change this name in this dialog.
Configuration label	A string that specifies the name that is displayed on the tab in the client application.
Available attributes	Shows a list of the attributes that can be displayed on the tab. Select the attribute that you want to display on the tab and click **Add**. The attribute appears in the Ordered chosen attributes list. If the available attributes list is empty, no attributes have been configured yet. For more information about configuring attributes, see Adding aspect attributes, page 84.
Ordered chosen attributes	Specifies which attributes are displayed on the tab and how they are displayed. You can arrange how the attributes are displayed on the tab by selecting the attribute and using the following buttons: • **Up**: Moves the attribute up in the display order. • **Down**: Move the attribute down in the display order. • **Add Separator**: Adds a separator between the selected and the following attribute. • **Remove Separator**: Removes the separator. • **Make Secondary**: Force attributes to be displayed on a secondary page, if not all attributes can fit on one tab.

Tab properties	Description
Custom attributes only	Select this option to display only custom attributes in the Available attributes list.
Hidden attributes only	Select this option to display only hidden attributes in the Available attributes list.

4. Click **OK** to save your changes.

Creating an aspect module

Before you can create an aspect module, you must create a Documentum project. For more information about creating a Documentum project, see Creating a new project, page 25.

To create an aspect module:

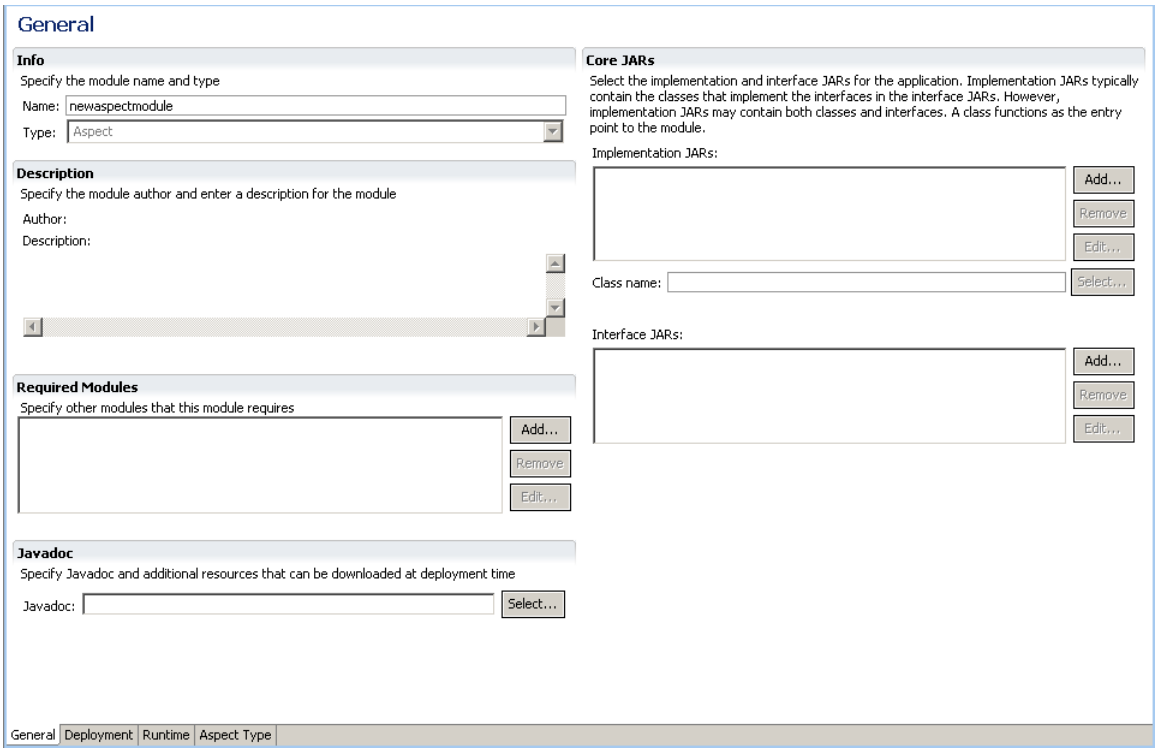
1. Open the Select a wizard dialog in one of the following ways:
   - From the Composer menu, select **File > New > Other**.
   - In your Composer project, expand the **Artifacts** folder and right-click **module**. Select **New > Other.**

   The **Select a wizard** dialog appears.
2. Double-click the Documentum folder to expand it, then select **Aspect Module**.

   The **New Documentum Artifact Name and Location** dialog appears.
3. Enter a name for the new aspect module, then click **Finish**.

   Note: It is recommended to use the same name for the aspect module and the aspect type associated with the module.

   The **Aspect Module** editor appears with the **General** tab selected.



4. Enter the required and optional properties in the Info, Description, Required Modules, Javadoc, and Core JARs sections, as described in Table 19, page 90.

Table 19. Properties in module editor General tab

Property	Description
General	
Name	A string specifying the name of the module. Required parameter. The name can have up to 255 characters.
Type	A string specifying the type of the module. Required parameter. An aspect module can only be of the type Aspect.
Description	
Author	Contact information for the module author. Optional parameter
Description	Description for the module, not exceeding 255 characters. Optional parameter.
Required Modules	Specifies modules that this module requires to function properly. Click **Add** to open the Module Artifact dialog. Select a module from the listbox and click **OK**, or click **New** to create a new module.

Property	Description
Javadoc	Specifies Javadocs and other resources that can be downloaded with the aspect module at runtime. Click **Select** to open the SysObject Subtype Artifact dialog. Select a SysObject that contains the Javadoc or resource content from the list or click **New** to create a SysObject containing the content to be downloaded.
Core JARs	
Implementation JARs	Implementation of the module. Required parameter. Click **Add** to add implementation JARs from your local machine.
Class name	Primary Java implementation class for the module. Required parameter. TBOs must implement the IDfBusinessObject interface; SBOs must implement the IDfService interface; and all modules must implement the IDfModule interface.
Interface JARs	Java interfaces that this module implements. Optional parameter. Click **Add** to add interface JARs from your local machine.

5. Click the **Deployment** tab and configure the module dependencies as described in Configuring aspect module deployment , page 91.
6. Click the **Runtime** tab to configure the runtime environment for this module, as described in Configuring the aspect module runtime environment, page 93.
7. Click the **Aspect Type** tab to configure an aspect type for this module, as described in Configuring the aspect type, page 94.

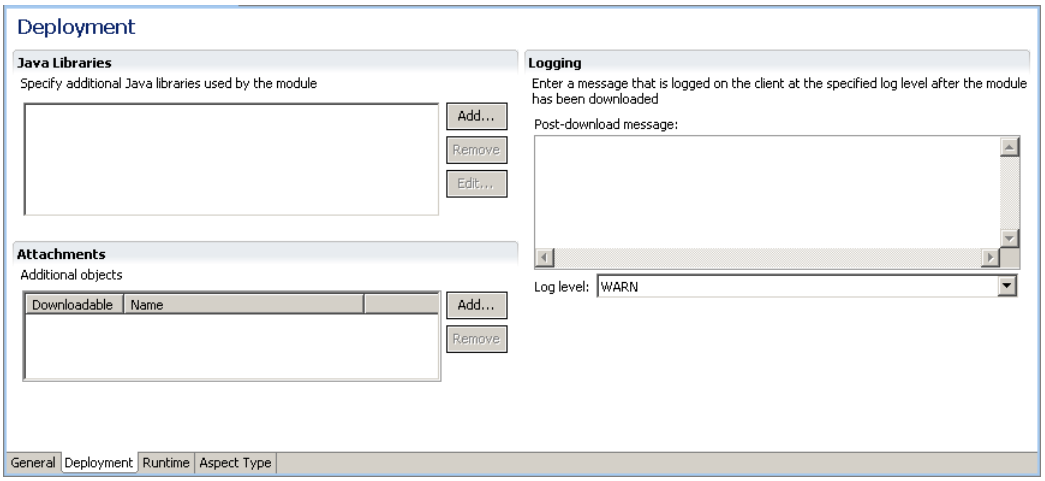
Configuring aspect module deployment

The **Deployment** tab lets you link Java libraries and other modules to the module you are creating or editing.

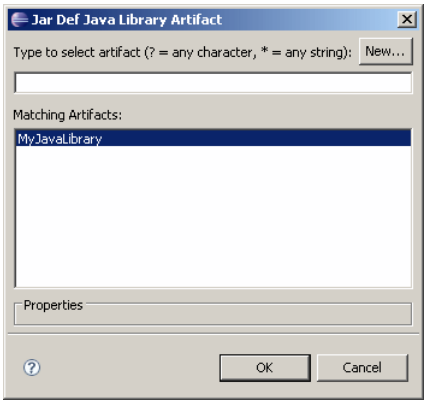
To configure module deployment:

1. Click the Deployment tab in the aspect module editor.

   The **Deployment** view appears.



2. In the **Java Libraries** section, click **Add** to add Java libraries for this module.
   The **Jar Def Java Library Artifact** dialog appears.



   Select a Java library from the listbox and click **OK** or click **New** to create a new Java Library. You can only link existing Java libraries into this module. You cannot modify an existing library that is shared by multiple modules. For more information about creating a new Java Library, see Linking and configuring a Java Library, page 103.

3. In the **Attachments** section, specify additional objects that should be available for download when the module is deployed.
4. In the **Logging** section, specify a post-download message and select a log level for the message.
   The log level can have the following values:
   - **WARN**: The post-download message is logged as a warning.
   - **NONE**: The post-download message is not logged.
   - **INFO**: The post-download message is logged as an informational message.
   - **DEBUG**: The post-download message is logged at debug level.
5. Save your changes.

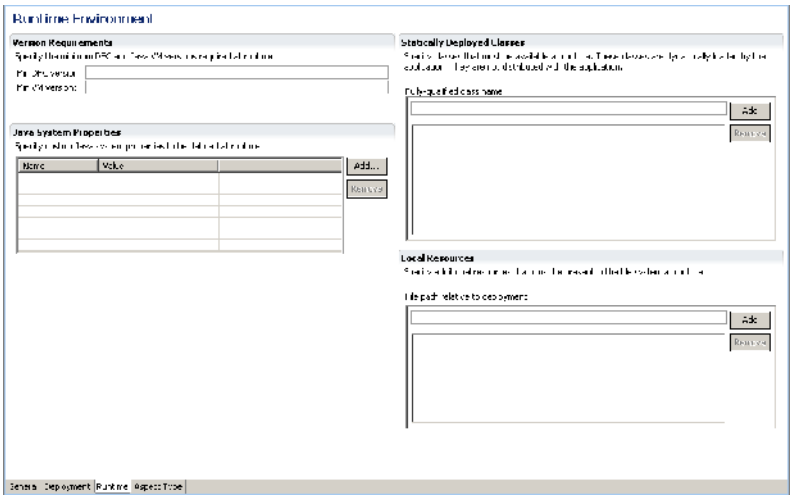
Configuring the aspect module runtime environment

The runtime environment lets you configure optional properties that are executed at runtime, such as version requirements, Java system properties, statically deployed classes, and local resources.

To configure the runtime environment:

1. Click the **Runtime** tab in the aspect module editor.

   The **Runtime** view appears.



2. Specify the version requirements, Java system properties, statically deployed classes, and local resources, as described in Table 20, page 93.

Table 20. Module runtime environment properties

Property	Description
Version Requirements	This section lets you specify the DFC and Java VM versions required on the client for the module to function properly.
Min DFC version	The minimum DFC version on the client machine for this module to work properly.
Min VM version	The minimum Java VM version on the client machine for this module to work properly.
Java System Properties	This section lets you specify Java system properties as name-value pairs. When the module is downloaded, the client machine is checked to see if all the specified Java properties match the properties on the client machine. Click **Add** to enter placeholders for the name and value of the Java system property, then click the **Name** and the **Value** fields to modify the property name and value.
Name	Name of the Java system property.

Property	Description
Value	Corresponding value for the Java system property name.
Statically Deployed Classes	This section lets you specify static Java classes that are required for the module to function properly. When the module is downloaded, the class path is checked for the specified Java classes.
Fully qualified class name	Fully qualified Java class names. Enter the class name and click **Add**.
Local Resources	This section lets you specify files that are required on the local machine for the module to function properly. When the module is downloaded, the client machine is checked for the specified files specified.
File path relative to deployment	Full file path. Enter the file name and path and click **Add**.

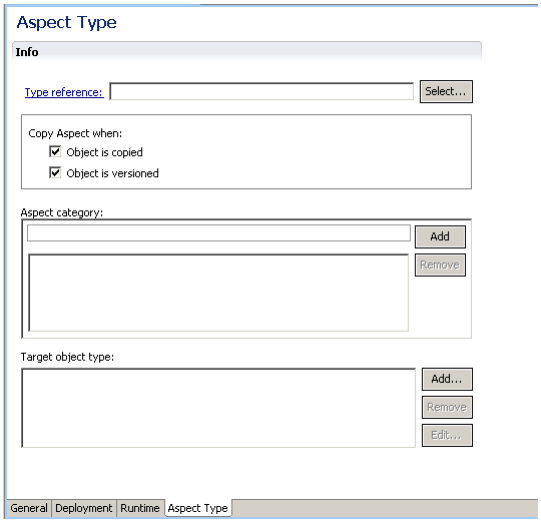
3. Save your changes.

Configuring the aspect type

Use the **Aspect Type** tab to configure the aspect type.

To configure the aspect type:

1. Click the **Aspect Type** tab in the aspect module editor.
   The **Aspect Type** view appears.



2. Configure the aspect type properties in the **Info** section, as described in Table 21, page 95.

Table 21. Aspect type properties

Property	Description
Info	
Type reference	Specifies the aspect type that is associated with this aspect module. Click **Select** to add a type reference. The **Select Aspect Artifact** dialog displays. Select an aspect type from the list or click **New** to create a new aspect type. For more information about creating an aspect type, see Creating an aspect type, page 81.
Copy aspect	Specifies whether the aspect is copied with the associated object if the object is copied. By default, the aspect is copied with the object.
Version aspect	Specifies whether the aspect is copied with the associated object if the object is versioned. By default, the aspect is copied with the object.
Aspect category	Specifies the aspect category that is associated with this aspect module. To add an aspect category, select and aspect category from the list and click **Add**.
Target object type	Specifies to which object types the aspect can be attached. Click **Add** to add a target object type. The **Select Type Artifact** dialog displays. Select a type from the list or click **New** to create a new type. For information about creating types, see Chapter 17, Managing Types .

Chapter 7

Managing Formats

This chapter contains the following topics:

Formats

A format object contains information about a file format recognized by Content Server. A predefined set of file formats is installed by default when a repository is configured.

Creating a format artifact

Use the format editor to create a format artifact.

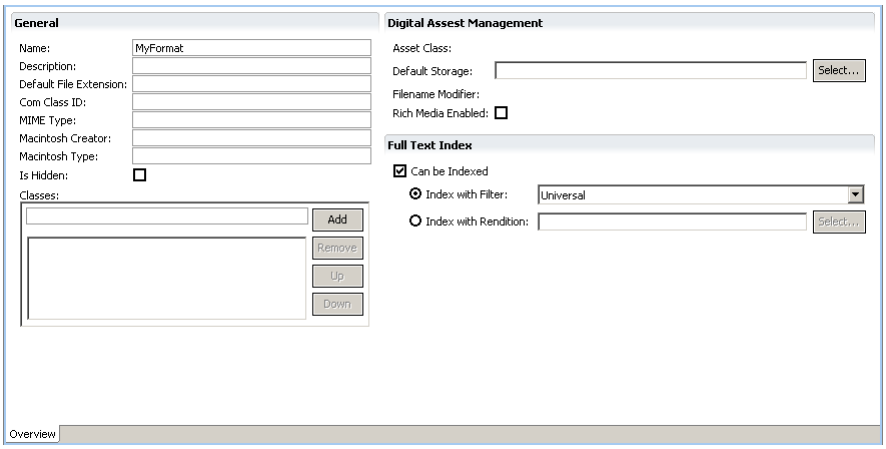
To create a format artifact:

1. Open the **Select a wizard** dialog in one of the following ways:
   - From the Composer menu, select **File > New > Other**.
   - In your Composer project, expand the **Artifacts** folder and right-click **Formats**. Select **New > Other.**

   The **Select a wizard** dialog appears.
2. Select **Documentum > Format**, then click **Next**.

   The **New Documentum Artifact - Name and Location** dialog appears.
3. Enter a folder path and a name for the format artifact, or accept the default path and name, then click **Finish**.

   The **Format** editor appears.



4. Enter the format properties in the **General**, **Digital Asset Management**, and **Full Text Index** sections, as described in Table 22, page 98.

Table 22. Format artifact properties

Parameter	Description
General	
Name	An ASCII string specifying the name of the format. The string cannot be longer than 64 characters.
Description	A string describing the format. The string cannot be longer than 64 characters.
Default file extension	A string specifying the DOS extension that is used when a file with this format is copied into the common area, client local area, or storage. The string cannot be longer than 10 characters.
COM class ID	A string specifying the class ID recognized by the Windows registry for a content type. The string cannot be longer than 38 characters.
MIME type	A string specifying the Multimedia Internet Mail Extension (MIME) for the content type. The string cannot be longer than 64 characters.
Macintosh creator	A string with up to 4 characters used internally for managing Macintosh resource files.
Macintosh type	A string with up to 4 characters used internally for managing Macintosh resource files.
Is hidden	Used by client applications to determine whether to display this format object in the client application's user interface. If selected, the format is not displayed in the client's user interface.
Classes	Specifies the class or classes of formats to which the format belongs. For example, the xml, xsd, and xsl formats belong to the XML and MSOffice classes.
Digital Asset Management	

Parameter	Description
Asset class	A string with up to 32 characters that applications use to classify the asset type (for example, audio, video, image) of the contents of objects with this format.
Default storage	Specifies the default storage area (identified by its object_id) where the contents of the objects with this format are stored. If a storage type is not specified for a SysObject, the default storage for the associated format is used. If no default storage is specified, then the storage type specified for the object type is used. If none of these are specified, then turbo storage is used as the default storage.
File name modifier	Specifies a string that a client application can append to a file name when multiple renditions (of an object) having the same extension are exported. For example, if you specify "_th" as the filename_modifier for the jpeg_th format, then when a rendition, my_picture.jpeg with a jpeg_th format, is exported, the rendition's file name is my_picture_th.jpeg.
Rich media enabled	Indicates whether Content Server automatically generates thumbnails, auto proxy and metadata for its contents.
Full Text Index	
Can be indexed	Indicates whether an object's content with the format can be full-text indexed.
Index with filter	Name of the Verity topic filter to use for full-text indexing. The topic filter can have the following values: • Universal • None
Index with rendition	A string specifying the format to which this format must be transformed for full-text indexing. Click **Select** to select a format from the listbox.

Chapter 8

Managing JARs and Java Libraries

This chapter contains the following topics:

JAR definitions, JARs and Java libraries

A Java ARchive or JAR file is an archive file that aggregates many files into one. It is generally used to distribute Java classes and associated metadata, and can serve as building block for applications and extensions. The JAR files themselves can be bundled in a Java library.

There are two types of JAR files, interface JARs and implementation JARs. Interface JARs contain Java interface classes and the implementation JARs contain the classes that implement the interface classes. Generally, the interface classes and the implementation classes are aggregated in separate JAR files, however in some cases a JAR file can contain both, interface and implementation classes.

Composer lets you create a definition that points to the JAR files and Java libraries. The definition encapsulates the JAR files and Java libraries and links them to artifacts, such as modules.

Creating a JAR Definition

Composer lets you create a JAR definition that points to JAR files containing Java interface and implementation classes. You can create a JAR definition either from the module editor or from the Composer main menu.

To create a JAR definition:

1. Open the Select a wizard dialog in one of the following ways:
   - From the Composer menu, select **File > New > Other**.
   - In your Composer project, expand the **Artifacts** folder and right-click **JAR Definitions**. Select **New > Other.**

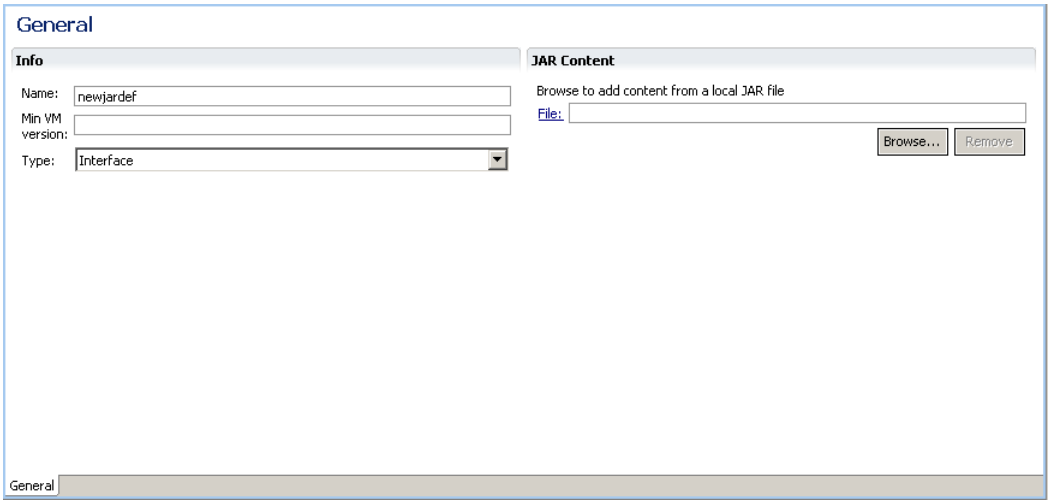
   The **Select a wizard** dialog appears.

2. Select **Documentum > JAR Definition**, then click **Next**.

   The **New Documentum Artifact - Name and Location** dialog appears.

3. Enter a folder path and a name for the JAR definition, or accept the default path and name, then click **Finish**.

   The **JAR** definition editor appears.



4. Enter the JAR definition properties and add file content, as described in Table 23, page 102.

Table 23. JAR definition properties

Property	Description
Info	
Name	A string specifying the name of the JAR definition.
Min VM version	A string with a maximum of 32 characters specifying the Java VM version level required by this JAR definition.
Type	Specifies the type of JAR definition. The type can have the following values: • **Implementation**: The JAR file definition contains implementation classes or implementation and interface classes. • **Interface**: The JAR file definition contains only interface classes. • • **Interface and Implementation**: This JAR file definition points to both, interface and implementation classes.

Property	Description
JAR Content	Specifies the local files that are aggregated in the JAR file. Click **Browse** to select a JAR file on the local machine. The **Select content from a JAR file** dialog appears. Select the JAR file from the list and click **Open**. To view the content of a selected JAR file, click the **File** link, then select the editor you want to use to view the content.

5. Save your changes.

Linking and configuring a Java Library

Composer lets you link Java libraries to a module. A Java library contains interface and implementation JARs for the module. You can link a Java library either from the module editor or from the Composer main menu.

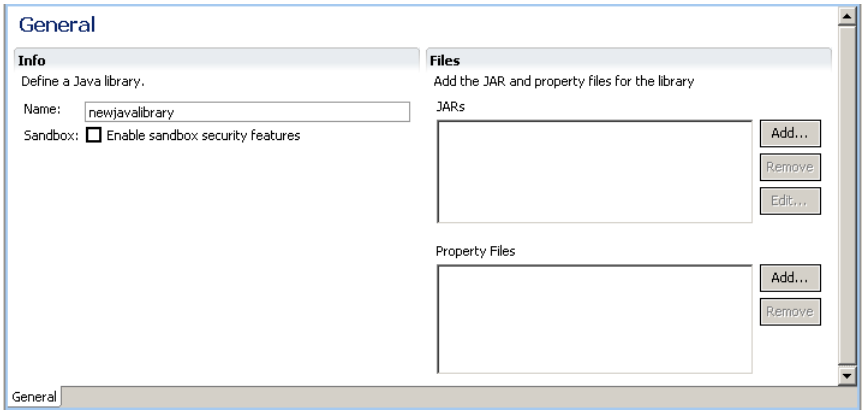
To link a Java library:

1. Open the **Select a wizard** dialog in one of the following ways:
   - From the Composer menu, select **File > New > Other**.
   - In your Composer project, expand the **Artifacts** folder and right-click **Java Library**. Select **New > Other.**

   The **Select a wizard** dialog appears.
2. Select **Documentum > Java Library**, then click **Next**.

   The **New Documentum Artifact - Name and Location** dialog appears.
3. Enter a folder path and a name for the Java library, or accept the default path and name, then click **Finish**.

   The **Java Library** editor appears.



4. Configure the Java library, as described in Table 24, page 104.

Table 24. Java library properties

Property	Description
Info	
Name	A string specifying the name of the Java library.
Sandbox	Specifies whether this Java library uses a sandbox. If checked, the Java library uses a sandbox. By default, the sandbox is disabled.
Files	The JARs and property files to be included in the Java library. Click **Add** and select the JARs and properties files from the listbox.

5. Save your changes.

Chapter 9

Managing Lifecycles

This chapter contains the following topics:

Lifecycles

A lifecycle specifies business rules for changes in the properties of an object, such as a document. In other words, a lifecycle defines the different stages of an attached document. For example, a document could be in a draft state, a review stage, and a finalized state. A lifecycle does not define what activities happen to the document while it resides in a state. Activities are defined by workflows.

From a high-level view, a lifecycle consists of an attached object and various states that define the properties of the object. Planning a lifecycle includes determining the following:

- Object type(s) that can be attached to the lifecycle.
- Normal states, including entry and exit criteria, state actions, and procedures.
- Exception states, including entry and exit criteria, state actions, and procedures.
- Validation procedures.
- Alias sets.

Lifecycle object types

In general, any content object type can be attached to a lifecycle. SysObjects are the supertype, directly or indirectly, of all object types that can have content, and any SysObject and SysObject subtype can be attached to a lifecycle. The SysObject subtype most commonly associated with content is dm_document.

A lifecycle requires a primary object type and can include secondary object types. The primary object type specifies the type of document that can be attached to the lifecycle. A document can only be attached to the lifecycle, if the document type matches the primary object type. The primary object type can only be dm_sysobject or one of its subtypes. Secondary object types are subtypes of the primary object type.

By default, the Composer provides the dm_sysobject and its subtypes in the lifecycle editor.

You can also create a new object type. For more information about object types and creating a new object type, see Chapter 17, Managing Types .

Creating a lifecycle

You can create a lifecycle using the **Lifecycle** editor in Composer.

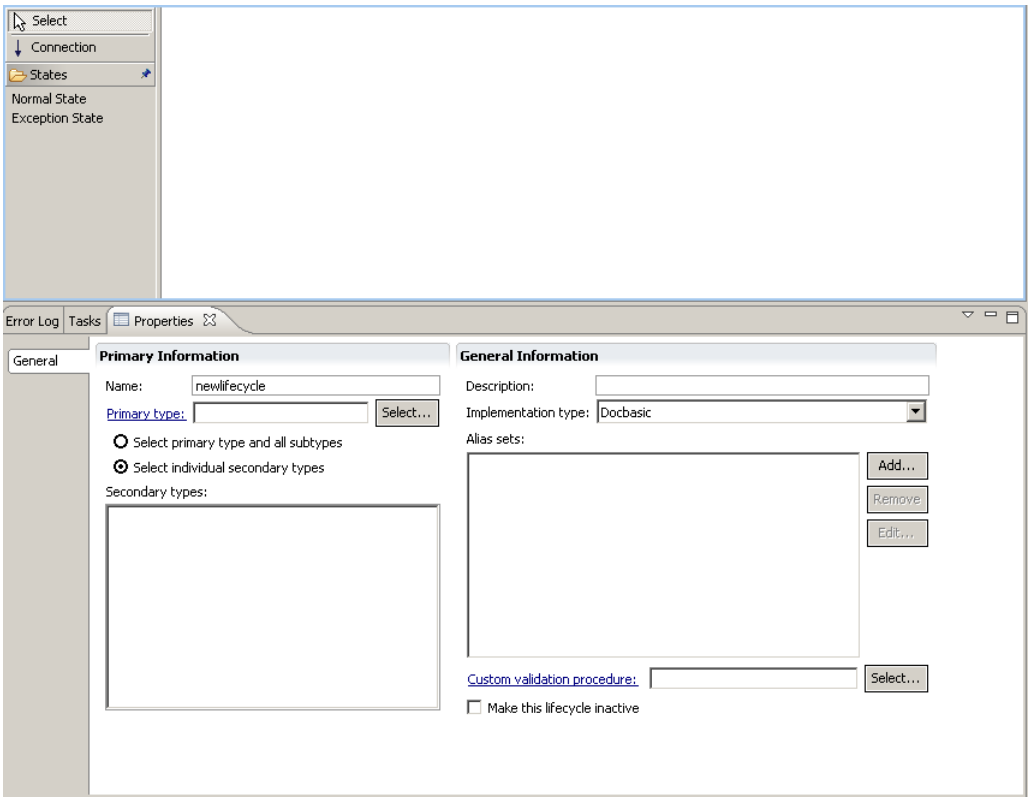
To create a lifecycle:

1. Open the lifecycle editor in one of the following two ways:
   - From the Composer main menu, select **File > New > Other**.
   - In your Composer project, expand the **Artifacts** folder and right-click **Lifecycles**. Select **New > Other.**

   The **Select a wizard** dialog appears.
2. Select **Documentum Artifact** > **Lifecycle**, then click **Next**.

   The **New Documentum Artifact – Name and Location** dialog appears.
3. Enter the full path name of the folder in which you want to create the new lifecycle in the **Folder:** field or click **Browse** to select a folder from drop-down list.
4. Enter the name of the lifecycle in the **Artifact name:** field, or accept the default name.
5. Click **Finish**.

   The **Lifecycle** editor appears.

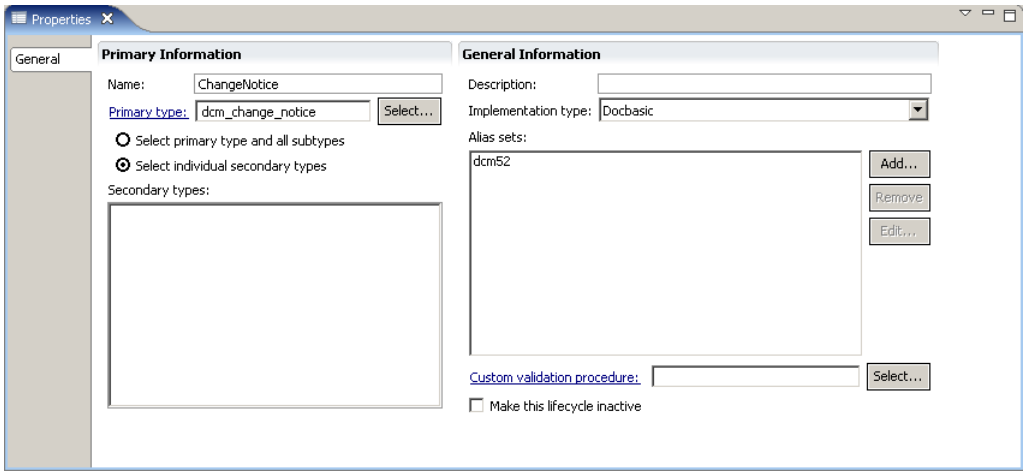


6. Configure the properties of the lifecycle, as described in Configuring lifecycle properties, page 107.
7. Configure lifecycle states, as described in Adding and configuring lifecycle states, page 109.
8. Save your changes.

Configuring lifecycle properties

The **General** tab of the lifecycle properties page (Figure 5, page 108) lets you configure the properties of a lifecycle, such as the primary object type, secondary object types, a validation procedure, implementation type, and alias sets.

Figure 5. Lifecycle properties tab



Configure the **Primary Information** and **General Information** for the lifecycle, as described in Table 25, page 108.

Table 25. Lifecycle properties tab parameters

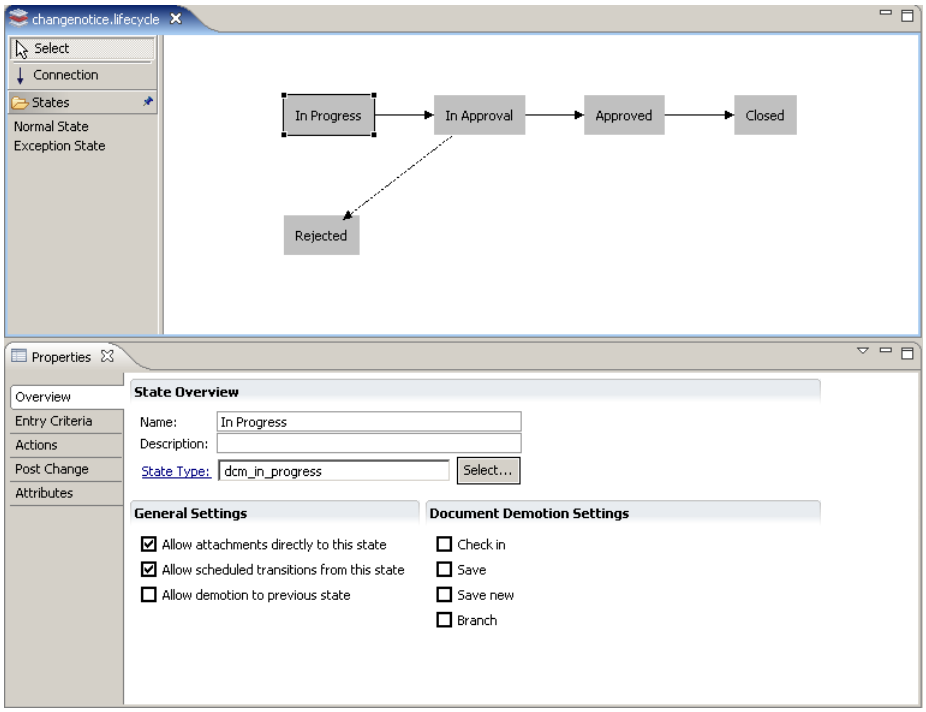
Parameter	Description
Primary Information	
Name	A string specifying the name of the lifecycle.
Primary type	Assign a primary object type, using one of the following methods: • Click **Select**. The **Select Type Artifact** dialog appears. Select the primary object type from the listbox. • Click the Primary type link to create a new object type. The **New Object Type Artifact** wizard appears. For more information about creating a custom object type, see Creating a standard object type, page 170.
Select primary type and all subtypes	If selected the lifecycle applies to the primary type and all its subtypes.
Select individual secondary types	Select this option if you want to assign individual secondary types. Choose the secondary types from the Secondary types listbox.

Parameter	Description
Secondary types	Displays the secondary types that can the assigned to this lifecycle. Secondary object types are subtypes of the primary object type that is specified in the **Primary type** field. If you specify secondary object types, only objects of the secondary object type can be attached to the lifecycle. If you do not select any secondary object types, the attached type must match the primary object type.
General Information	
Implementation type	Specifies the implementation type. Select an implementation type from the drop-down list. There are two implementation types, as follows: • Docbasic • Java
Alias sets	Specifies the alias set associated with this lifecycle. Add one or more alias sets by clicking **Add** next to the Alias sets field in the **General Information** section. The aliases are resolved to real user names or group names or folder paths when the lifecycle executes.For more information about alias sets, see Chapter 5, Managing Alias Sets.
Custom validation procedure	Specifies the validation procedure associated with this lifecycle. Assign a validation procedure using the **Validation Procedure** field. Click **Browse** to select a validation procedure.
Make this lifecycle inactive	Select this option to deactivate this lifecycle.

Adding and configuring lifecycle states

Once you have created the lifecycle, you can add states. Lifecycle states are added in the main window of the lifecycle editor in form of a state diagram (Figure 6, page 110). Each state appears as a rectangle, arrows designate transitions from one state to the next.

Figure 6. Lifecycle editor with state diagram



There are two types of lifecycle states, as follows:

- Normal state

  Normal states follow a linear progression, from the first (base) state to the last (terminal) state.

- Exception state

  Exception states handle any deviation from the linear progression. Each normal state has either zero or one exception state into which documents can transition from a normal state. An exception state can serve more than one normal state.

To add a lifecycle state:

1. In the lifecycle editor palette, click **Normal State** or **Exception State**, depending on the type of state you want to add.
2. Move your mouse cursor over the editor window and left-click inside the editor window to draw the lifecycle state.
3. Enter the state properties in State Overview, General Settings, and Document Demotion Settings in the Overview tab, as described in Table 26, page 110.

Table 26. State properties in Overview tab

Properties	Description
State Overview	
Name	A string that specifies the name of the lifecycle state.

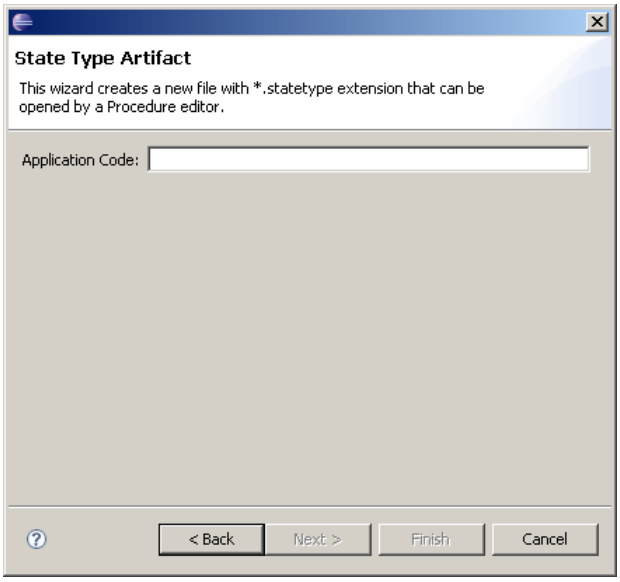
Properties	Description
Description	A string that describes the purpose or function of the lifecycle state. Optional parameter.
State Type	A string that specifies the state by type. A state type identifies a state that a client application uses. Certain client applications identify a state by type instead of the name. Type an existing state type in the **State type** field, click **Select** and select a state type from the list, or click **State Type** to create a new state type. For more information about creating state types, see Creating a state type , page 112.
General Settings	
Allow attachments directly to this state	If this setting is checked, users can attach a document to this state. The document type must match the primary and secondary object types that were specified for the lifecycle.
Allow scheduled transitions from this state	If this setting is checked, a document can transition from this state to the next state as the result of a time schedule, and no explicit user action is required.
Allow demotions to previous state	If this setting is checked, a document attached to this state can be moved back to the previous state or the base (first) state.
Document Demotion Settings	
Check in	If this setting is checked, a document is automatically returned to the base state when the document is checked in. The document must pass the base state's entry criteria for the check in to succeed.
Save	If this setting is checked, a document is automatically returned to the base state when the document is saved. The document must pass the base state's entry criteria to be saved successfully.
Save new	If this setting is checked, a document is automatically returned to the base state when the document is saved as a new document. The document must pass the base state's entry criteria to be saved successfully.
Branch	If this setting is checked, a document is automatically returned to the base state when the document is branched.

Creating a state type

A state type identifies a state that a client application uses. Certain client applications identify a state by type instead of the name. You can create a state type artifact from the lifecycle state editor.

To create a state type:

1. In the **Lifecycle** editor, select the lifecycle state for which you want to create a state type.
2. In the **State Overview** section, click the **State Type** link.
   The **New Documentum Artifact – Name and Location** dialog appears.
3. Enter the full path name of the folder in which you want to create the new state type in the **Folder:** field or click **Browse** to select a folder from drop-down list.
4. Enter a name for the state type in the **Artifact name:** field, or accept the default name. The state type name must be unique.
5. Click **Next**. The **State Type Artifact** dialog appears.



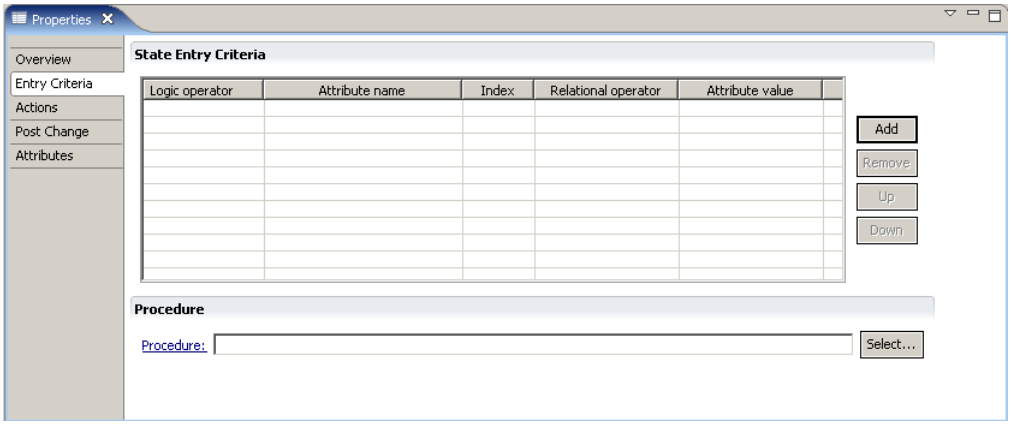
6. Enter the application code. Application code is a string that the client application recognizes, for example, “dm_dcm” for Documentum Compliance Manager (DCM). See the client application’s documentation for information about its application code and valid state types.
7. Click **Finish**.

Configuring state entry criteria

State entry criteria specify the conditions a document must meet to enter a stage and a procedure that is executed when the document enters the stage. State entry criteria are configured in the **Entry Criteria** tab on the **Properties** page.

To configure state entry criteria:

1. Click the **Entry Criteria** tab on the Properties pane to display the State Entry Criteria page.



2. In the **State Entry Criteria** section click **Add** to add one or more state entry criteria rules. Click on any field in a row to enter or edit a value, as described in Table 27, page 113.

Table 27. State entry criteria

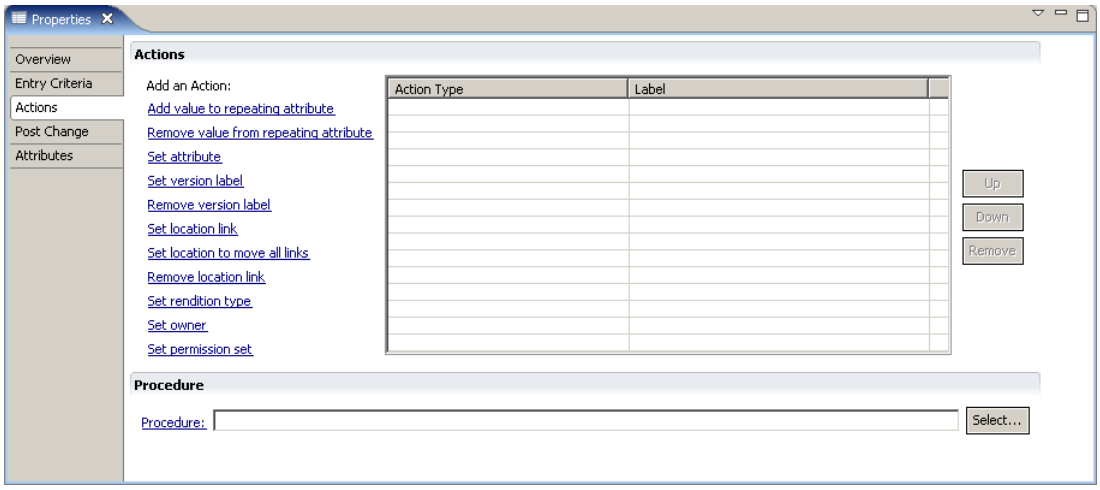
Property	Description
State Entry Criteria	
Logic operator	A logic operator that can have the values AND and OR.
Attribute name	The name of the attribute that specifies the entry criteria for the state. Click on the field and use the drop-down list to select an attribute name. The list only shows the attributes that are valid for the primary object type associated with the lifecycle.
Index	Specifies the position of the entry criteria in the entry criteria list. By default, the index is set to NONE.
Relational operator	A relational operator that can have the following values: • > (greater than) • >= (greater or equal) • < (less than) • <= (less or equal) • = (equal)
Attribute value	A string that specifies the value of the attribute.

Property	Description
Procedure	
Procedure	The procedure that is executed when a document enters this stage of the lifecycle. Enter a procedure name in one of the following ways: • Click **Select**. The **Procedure Artifact** dialog appears. Select a procedure from the listbox or click **New ...** to create a new procedure. • Click the **Procedure:** link to create a new procedure. For more information about creating a new procedure, see Creating a procedure, page 151.

Configuring state actions

For each state, you can define actions to be performed on an object entering the state. The actions on entry are performed after the entry criteria are evaluated. The actions must complete successfully before the object can enter the state. All state actions can be configured in the **Actions** tab of the **Properties** page.

Figure 7. Lifecycle state actions



Configuring repeating attributes

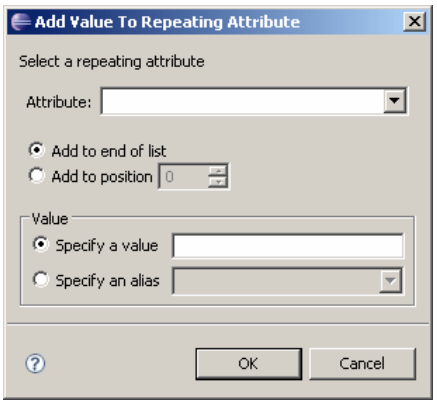
Repeating attributes are attributes that can have more than one value. For example, the document object's authors attribute is a repeating attribute since a document can have more than one author.

To configure repeating attributes:

1. In the lifecycle state diagram, click on the state for which you want to configure one or more repeating attributes.
2. Click the **Actions** tab in the **Properties** pane.

   The **Actions** page appears.
3. Click the **Add value to repeating attribute** link.

   The **Add Value to Repeating Attribute** dialog appears.



4. Enter the name of the repeating attribute, the position, and the value, as described in Table 28, page 115.

Table 28. Add repeating attribute properties

Property	Description
Attribute	The name of the repeating attribute. You can either type the name in the **Attribute** field or select a name from the drop-down list. The list only displays the attributes that are valid for the primary object you assigned to the lifecycle.
Add to end of list	Stores the position of the attribute at the end of the repeating attribute list. All repeating attributes are stored in a list, therefore an index value must be stored with the attribute's name.
Add to position	Stores the position of the attribute at the specified index position in the repeating attribute list. All repeating attributes are stored in a list, therefore an index value must be stored with the attribute's name.
Value	The value of the repeating attribute.

Property	Description
Specify a value	Select this option if you want the repeating attribute to be stored as a value, and enter the value.
Specify an alias	Select this option if you want the repeating attribute to be stored as an alias, then select the alias from the drop-down list. If the drop-down list does not show any aliases, no aliases have been configured for the project yet. For more information about configuring aliases, see Chapter 5, Managing Alias Sets.

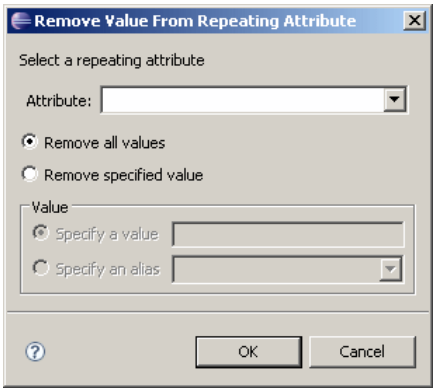
5. Click **OK** when you are finished.

Removing repeating attributes values

Repeating attributes are attributes that can have more than one value. For example, the document object's authors attribute is a repeating attribute since a document can have more than one author.

To remove one or more values from a repeating attribute:

1. In the lifecycle state diagram, click on the state for which you want to remove one or more repeating attributes values.
2. Click the **Actions** tab in the **Properties** pane.
   The **Actions** page appears.
3. Click the **Remove value from repeating attribute(s)** link.
   The **Remove Value from Repeating Attribute** dialog appears.



4. Select the attribute and specify the value or alias you want to remove, as described in Table 29, page 117.

Table 29. Remove repeating attribute properties

Property	Description
Attribute	The name of the repeating attribute. You can either type the name in the **Attribute** field or select a name from the drop-down list. The list only displays the attributes that are valid for the primary object you assigned to the lifecycle.
Remove all values	Select this option to remove all values from the repeating attribute.
Remove specified value	Select this option if you want to remove a specific value or alias to be removed, then enter the value or select the alias set in the Value section.
Value	The value of the repeating attribute.
Specify a value	Select this option if you want a specific value to be removed from the attribute, and enter the value.
Specify an alias	Select this option if you want a specific alias to be removed from the attribute, then select the alias from the drop-down list.

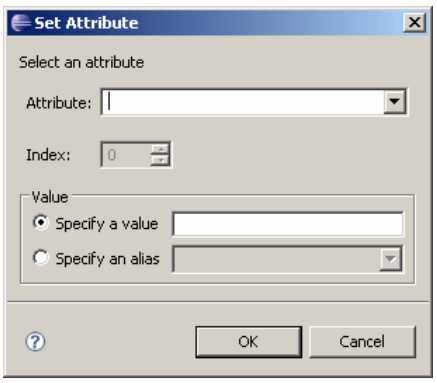
5. Click **OK** when you are finished.

Setting attributes

The **Set Attribute** action lets you specify attributes for a state. For example, the title or version of a document.

To set an attribute for a lifecycle state:

1. In the lifecycle state diagram, click on the state for which you want to configure an attribute.
2. Click the **Actions** tab in the **Properties** pane.
   The **Actions** page appears.
3. Click the **Set attribute** link.
   The **Set Attribute** dialog appears.



4. Select the attribute, and enter an index and value for the attribute, as described in Table 30, page 118.

Table 30. Set attribute properties

Property	Description
Attribute	The name of the attribute. You can either type the name in the **Attribute** field or select a name from the drop-down list. The list only displays the attributes that are valid for the primary object you assigned to the lifecycle.
Index	Stores the position of the attribute according to the index value that is entered. All attributes are stored in a list, therefore an index value must be stored with the attribute's name.
Value	The value of the attribute.
Specify a value	Select this option if you want the attribute to be stored as a value, and enter the value.
Specify an alias	Select this option if you want the attribute to be stored as an alias, then select the alias from the drop-down list. If the drop-down list does not show any aliases, no aliases have been configured for the project yet. For more information about configuring aliases, see Chapter 5, Managing Alias Sets.

5. Click **OK** when you are finished.

Setting version labels

Version labels let you specify which version of a document can be attached to a lifecycle state.

To specify a version label:

1. In the lifecycle state diagram, click on the state for which you want to specify a document version.
2. Click the **Actions** tab in the **Properties** pane.
   The **Actions** page appears.
3. Click the **Set version label** link.
   The **Set Version Label** dialog appears.
4. Enter the version label in the **Version label** field, then click **OK** to save your changes.

Removing version labels

Use the **Remove version label** link to remove version labels from a lifecycle state.

To remove a version label:

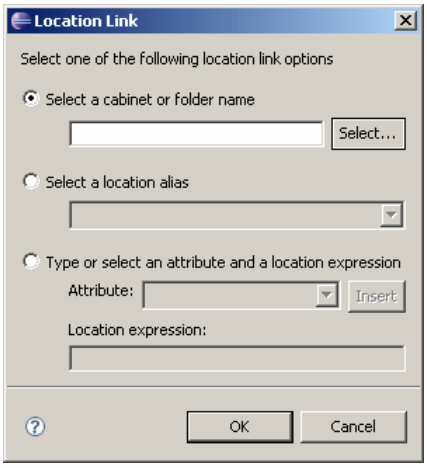
1. In the lifecycle state diagram, click on the state for which you want to remove a document version.
2. Click the **Actions** tab in the **Properties** pane.
   The **Actions** page appears.
3. Click the **Remove version label** link.
   The **Remove Version Label** dialog appears.
4. Enter the version label in the **Version Label** field, then click **OK** to save your changes.

Setting location links

Location links let you link a document to a specific location, such as a folder, a cabinet, an alias, or a local expression. The link is created when the document enters the state.

To link a document to a specific location:

1. In the lifecycle state diagram, click on the state for which you want to configure a location link.
2. Click the **Actions** tab in the **Properties** pane.
   The **Actions** page appears.
3. Click the **Set location link** link.
   The **Location Link** dialog appears.



4. Select one of the location link options and enter the location to which you want to link the document when it enters the state, as described in Table 31, page 120.

Table 31. Location link properties

Property	Description
Select a cabinet or folder	Select this option to link the state to a cabinet or folder. Click **Select.** The **Folder Subtype Artifact** dialog appears. Select a cabinet or folder from the listbox.
Select a location alias	Select this option to link the state to a location alias. Enter the location alias or select an alias from the drop-down list.
Type or select an attribute and a location expression	Select this option to link the state to an attribute and a location expression.
Attribute	A string specifying the name of the attribute for the location expression. Enter the attribute name or select an attribute from the drop-down list. The list only displays the attributes that are valid for the primary object you assigned to the lifecycle.
Location expression	The dynamic location path associated with the attribute selected from the **Attribute** drop-down list. Enter the location expression. The location path is resolved at runtime.

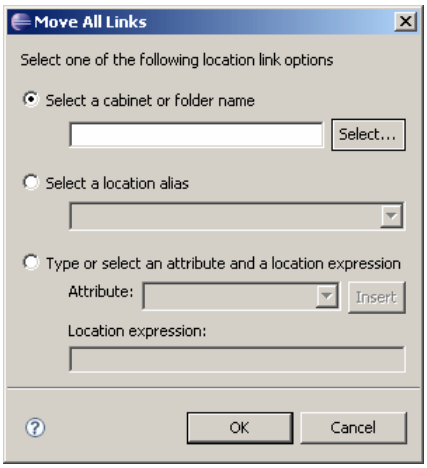
5. Click **OK** when you are finished.

Moving all links

The **Set location to move all links** link lets you move all links to a specific location.

To move all location links:

1. In the lifecycle state diagram, click on the state for which you want to move all links.
2. Click the **Actions** tab in the **Properties** pane.
   The **Actions** page appears.
3. Click **Set location to move all links**.
   The **Move All Links** dialog appears.



4. Select one of the location link options and enter the location to which you want to move all links when the document enters the state, as described in Table 32, page 121.

Table 32. Move all links properties

Property	Description
Select a cabinet or folder	Select this option to move all links to a cabinet or folder. Click **Select.** The **Folder Subtype Artifact** dialog appears. Select a cabinet or folder from the listbox.
Select a location alias	Select this option to move all links to a location alias. Enter the location alias or select an alias from the drop-down list.
Type or select an attribute and a location expression	Select this option to move all links to an attribute and a location expression.
Attribute	A string specifying the name of the attribute for the location expression. Enter the attribute name or select an attribute from the drop-down list. The list only displays the attributes that are valid for the primary object you assigned to the lifecycle.
Location expression	The dynamic location path associated with the attribute selected from the **Attribute** drop-down list. Enter the location expression. The location path is resolved at runtime.

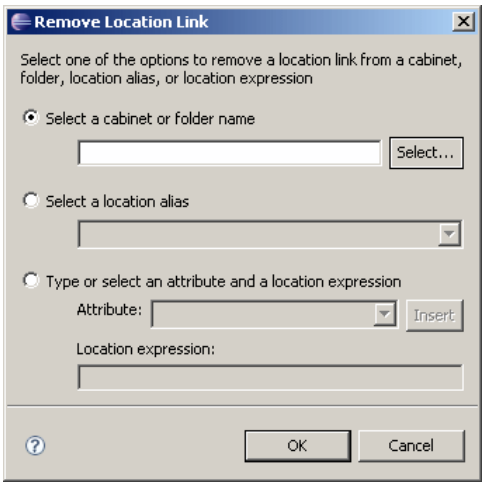
5. Click **OK** when you are finished.

Removing location links

Use the **Remove Location Link** dialog to remove location links.

To remove location links:

1. In the lifecycle state diagram, click on the state from which you want to remove a location link.
2. Click the **Actions** tab in the **Properties** pane.
   The **Actions** page appears.
3. Click the **Remove location link** link.
   The **Remove Location Link** dialog appears.



4. Select one of the location link options and enter the location from which you want to remove the document link when the document enters the state, as described in Table 33, page 122.

Table 33. Remove location link properties

Property	Description
Select a cabinet or folder	Select this option to remove the link to a cabinet or folder. Click **Select.** The **Folder Subtype Artifact** dialog appears. Select a cabinet or folder from the listbox.
Select a location alias	Select this option to remove the link to a location alias. Enter the location alias or select an alias from the drop-down list.
Type or select an attribute and a location expression	Select this option to remove the link to an attribute and a location expression.
Attribute	A string specifying the name of the attribute for the location expression. Enter the attribute name or select an attribute from the drop-down list. The list only displays the attributes that are valid for the primary object you assigned to the lifecycle.
Location expression	The dynamic location path associated with the attribute selected from the **Attribute** drop-down list. Enter the location expression. The location path is resolved at runtime.

5. Click **OK** when you are finished.

Assigning a document renderer

Composer uses Auto Render Pro for Windows or Macintosh to display a document attached to a lifecycle.

To assign the rendering application for an attached document:

1. In the lifecycle state diagram, click on the state for which you want to configure the rendering application.
2. Click the **Actions** tab in the **Properties** pane.
   The **Actions** page appears.
3. Click **Set rendition type**.
   The **Rendition Type** dialog appears.
4. Select one of the rendering options, as follows:
   - **Auto Render Pro for Windows**, if you are running Composer on a machine with a Windows operating system.
   - **Auto Render Pro for Macintosh**, if you are running Composer on a machine with a Macintosh operating system.
5. Click **OK** when you are finished.

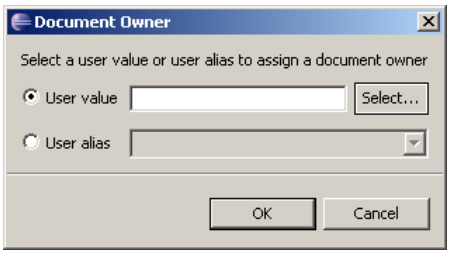
Assigning document owners

By default, the owner of a document is the user who creates it. However, you can assign ownership to another user or a group. To assign another user as the owner, you must be a superuser. To assign a group as the owner of an object, you must either be a superuser or you must own the document and be a member of the group to which you are assigning ownership.

Note: You must have an alias set assigned to the lifecycle to assign an owner to a document. You can assign an alias set in the **General** tab of the lifecycle properties page (Figure 5, page 108).

To assign an owner to a document:

1. In the lifecycle state diagram, click on the state for which you want to assign a document owner. The Properties pane appears below the lifecycle editor.
2. Click the Actions tab in the Properties pane. The Actions page appears.
3. Click the **Set owner** link. The Document Owner dialog appears.



4. Select one of the user options, as described in Table 34, page 124.

Table 34. Document owner properties

Property	Description
User value	Select this option assign a user value. Click **Select.** The **Principal (User or Group) Installation Parameter** dialog appears. Select a user value from the listbox or click **New** to create a new user value.
User alias	Use this option to assign a user alias. Select an alias from the drop-down list.

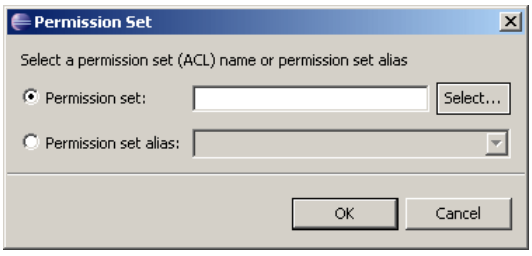
5. Click **OK** to save your changes.

Setting permission sets

Permission sets (also known as ACLs, or access control lists) specify the operations (such as read, edit, create a new version, or delete) users can perform on a document attached to a lifecycle.

To assign a permission set to a lifecycle state:

1. In the lifecycle state diagram, click on the state for which you want to assign a permission set.
2. Click the **Actions** tab in the **Properties** pane.
   The **Actions** page appears.
3. Click the **Set permission set** link.
   The **Permission Set** dialog appears.



4. Select one of the permission set options, as described in Table 35, page 125.

Table 35. Permission set properties

Property	Description
Permission set	Select this option to assign a permission set to the lifecycle state. Click **Select** to open the **Permission Set (ACL) Template** artifact dialog. Select a permission set from the list or click **New** to create a new permission set. For information about permission sets, see Chapter 12, Managing Permission Sets (ACLs).
Permission set alias	Select this option to assign a permission set alias and select a permission set alias from the drop-down list. For information about permission sets, see Chapter 12, Managing Permission Sets (ACLs).

Configuring post-change information

A post-change procedure executes after the state transition is complete. When the part of a transition that occurs within the transaction is complete, the system executes the post-change procedure. Failure of any part of the post-change procedure does not prevent the transition from succeeding.

To assign a post-change procedure to a lifecycle state:

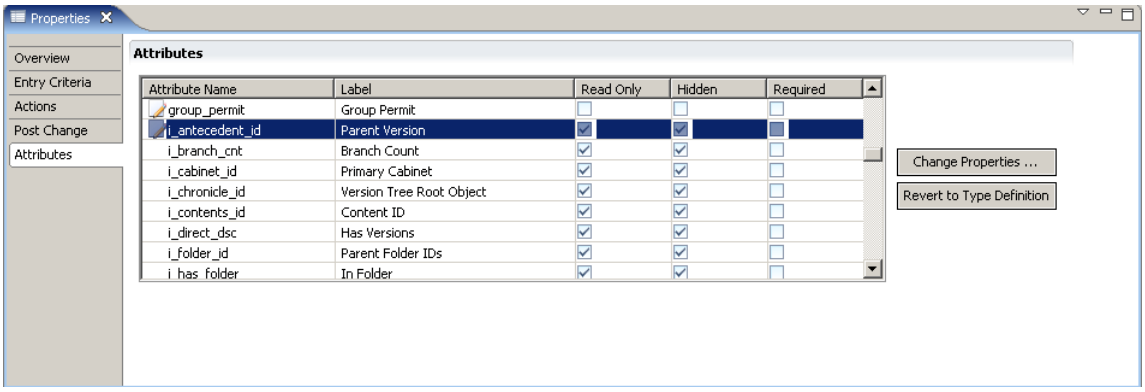
1. In the lifecycle state diagram, click on the state to which you want to assign a post-change procedure.
2. Click the **Post Change** tab in the **Properties** pane.
   The **Procedure** page appears.
3. Click **Select**.
   The **Procedure Artifact** dialog appears.
4. Select a post-change procedure from the list or click **New** to create a post-change procedure.
   For more information about creating procedures, see Chapter 13, Managing Procedures.

Configuring state attributes

State attributes include labels, help text, comment, and attribute properties for this state. The required and cannot be blank properties are checked when the client application validates an object (which typically occurs on saving or checking in an object), not when it enters the state.

To configure state attributes:

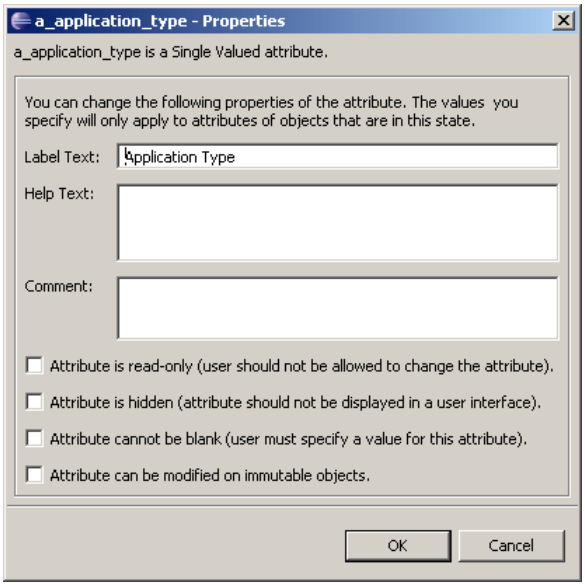
1. In the lifecycle state diagram, click on the state for which you want to configure attributes.
2. Click the **Attributes** tab in the **Properties** pane.
   The **Attributes** page appears.



The page only displays the attributes that are valid for the primary object you assign to the lifecycle.

3. Select an attribute from the **Attributes** table and click **Change Properties...** to change any of the attribute's properties.

   The **Properties** dialog for that attribute appears.



4. Configure the attribute's properties by selecting any of the available options, then click **OK**.

   You can revert back to the original type definition for the attribute by clicking **Revert to Type Definition**.

Deleting a state

To delete a lifecycle state:

1. In the lifecycle state diagram, select the state you want to delete.

2. Delete the state using one of the following methods:
   - Right-click the state and select **Delete** from the drop-down menu.
   - Select **File > Delete** from the Composer menu.

Deleting a lifecycle

To delete a lifecycle:

1. Locate the lifecycle in the **Documentum Navigator** view.
2. Right-click the lifecycle and select **Delete** from the pop-up menu.

Chapter 10

Managing Methods and Jobs

This chapter contains the following topics:

Methods and jobs

Methods are executable programs that are represented by method objects in the repository. The program can be a Docbasic script, a Java method, or a program written in another programming language such as C++.

Jobs automate the execution of a method, for example how to transfer content from one storage place to another. The attributes of a job define the execution schedule and turn execution on or off.

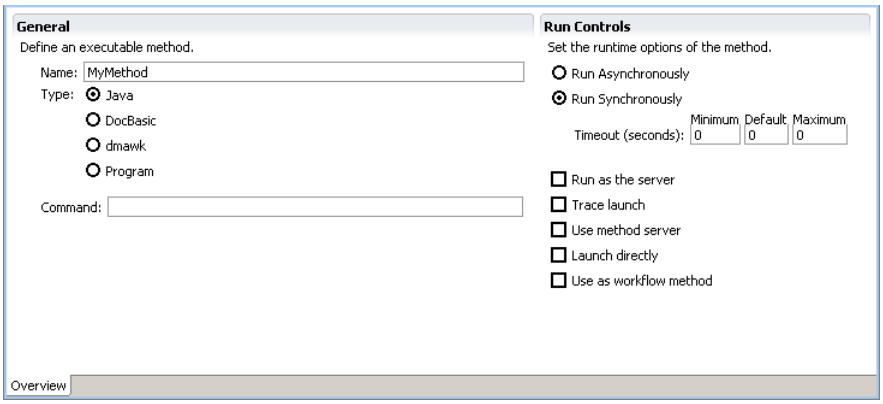
Creating a method

Use the **Method** editor to create a method.

To create a method:

1. Open the **Select a wizard** dialog in one of the following ways:
   - From the Composer menu, select **File > New > Other**.
   - In your Composer project, expand the **Artifacts** folder and right-click **Method**. Select **New > Other.**

   The **Select a wizard** dialog appears.
2. Double-click the Documentum folder to expand it, then select **Method**.
   The **New Documentum Artifact – Name and Location** dialog appears.
3. Enter a name for the new method or accept the default name, then click **Finish**.
   The **Method** editor appears.



4. Enter the properties for the method, as described in Table 36, page 130.

Table 36. Method artifact properties

Parameter	Description
General	
Name	A string specifying the name of the method. Do not use the format dm_methodname to name the method. This naming convention is reserved for default Documentum objects.
Type	Specifies the language or program used for this method. Select one of the following types: • **Java** — The method is written in Java and the Java method server is executing the method. • **Docbasic** — The method is written in Docbasic. • **dmawk** — The method is written in dmawk. • **Program** — The method is a program.
Command	The command that launches the method.
Run Controls	
Run Asynchronously	Specifies whether the procedure is run asynchronously or not. This parameter is ignored if the method is launched on the Method Server or Content Server and SAVE_RESULTS is set to TRUE on the command line.
Run Synchronously	Specifies whether the method is run synchronously.
Timeout	Specifies the minimum, default, and maximum amount of time before the methods times out.
Run as the server	Specifies whether the method is run as the server account. If selected, the method is run as the server account.

Parameter	Description
Trace launch	Specifies whether internal trace messages generated by the executing program are logged. If selected, the messages are stored in the session log.
Use method server	Specifies whether to use the Method Server or Application Server to execute Dmbasic or Java methods. If selected, the Method Server or application server is used. If not selected, the Content Server is used.
Launch directly	Specifies whether the program is executed by the operating system or exec API call. If selected, the server uses the exec call to execute the procedure. If the checkbox is cleared, the server uses the system call to execute the procedure.
Use as workflow method	Specifies whether this method is used in a workflow.

5. Save your changes.

Creating a job

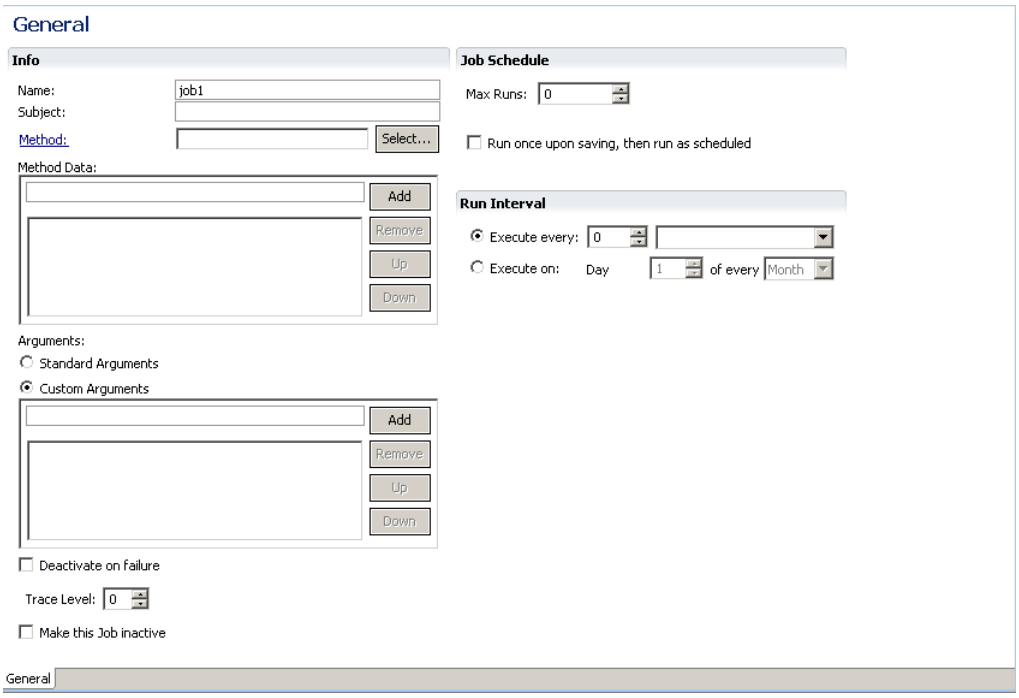
A job automates the execution of a method.

To create a job:

1. Open the **Select a wizard** dialog in one of the following ways:
   - From the Composer menu, select **File > New > Other**.
   - In your Composer project, expand the **Artifacts** folder and right-click **Job**. Select **New > Other.**

   The **Select a wizard** dialog appears.
2. Double-click the Documentum folder to expand it, then select **Job**. The New Documentum Artifact – Name and Location dialog appears.
3. Enter a name for the new job or accept the default name, then click **Finish**.

   The **Job** editor appears.



4. Enter the properties for the job in the Info, Job Schedule, and Run Interval sections, as described in Table 37, page 132.

Table 37. Job properties

Parameter	Description
Info	
Name	A string specifying the name of the job.
Subject	A comment or description of the job.
Method	The method that this job is automating. Click **Select** and select a method from the **Documentum Method Artifact** dialog or click **Method** to create a new method for this job.
Method data	Data that is used by method associated with job. This property is available for the method to write/read as needed during its execution. Enter the data in the method data field and click **Add**.
Standard arguments	Select to pass the standard arguments to the method. The standard arguments are: • **Repository owner** • **Repository name** • **Job ID** • **Trace level**

Parameter	Description
Custom arguments	Select to pass one or more custom arguments to the method. Enter the argument in the custom arguments field and click **Add**.
Deactivate on failure	Select to direct the application to stop running the job if the underlying method fails.
Trace level	Controls tracing for the method. Any value other than 0 turns on tracing. By default the trace level is set to 0.
Make this job active	Select to activate the job.
Job Schedule	
Max runs	Specifies the maximum number of times the job is run.
Run once upon saving, then run as scheduled	Select to run the job immediately after you save it, then return to the configured schedule.
Run Interval	
Execute every	Specifies the number of times this job is run every minute, hour, day, or week.
Execute on	Specifies a day on which this job is run. The job can be run on the same day once every week, once every month, or once every year.

5. Save your changes.

Chapter 11

Managing Modules

This chapter contains the following topics:

Modules

A module consists of executable business logic and supporting material, such as third-party software and documentation. A module is comprised of the JAR files that contain the implementation classes and the interface classes for the behavior the module implements, and any interface classes on which the module depends. The module may also include Java libraries and documentation.

There are three types of modules, as follows:

- Service-based modules (SBOs)

  An SBO provides functionality that is not specific to a particular object type or repository. For example, an SBO can be used to customize a user's inbox.

- Type-based modules (TBOs)

  A TBO provides functionality that is specific to an object type. For example, a TBO can be used to validate the title, subject, and keywords properties of a custom document subtype.

- Aspect

  An aspect provides functionality that is applicable to specific objects. For example, an aspect can be used to set the value of a one property based on the value of another property. An aspect module is created using a different editor, the **Aspect Module** editor, as described in Creating an aspect type, page 81.

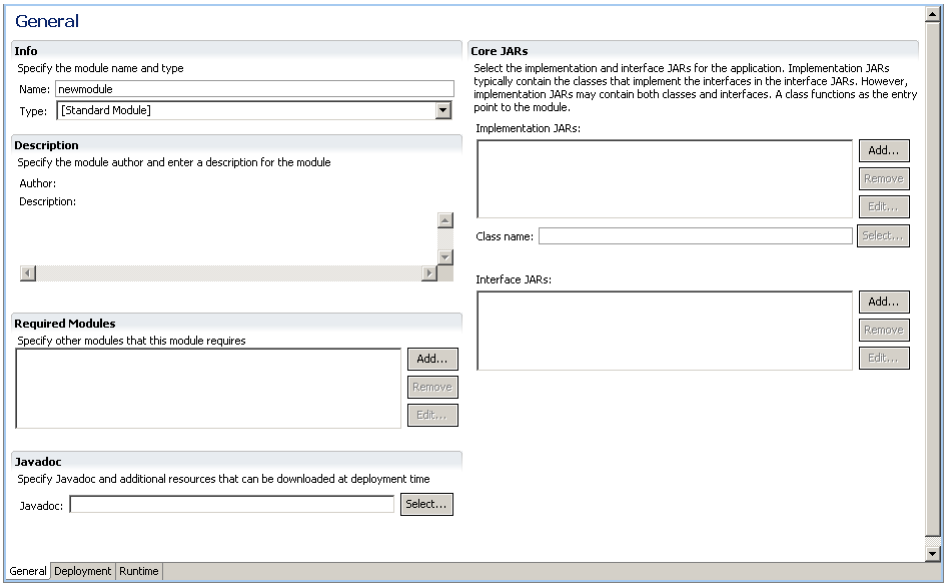
Creating a module

Use the **Module** editor to create a new module.

To create a module:

1. Open the **Select a wizard** dialog in one of the following ways:
   - From the Composer menu, select **File > New > Other**.
   - In your Composer project, expand the **Artifacts** folder and right-click **Module**. Select **New > Other.**

   The **Select a wizard** dialog appears.
2. Double-click the Documentum folder to expand it, then select **Module**.
   The **New Documentum Artifact – Name and Location** dialog appears.
3. Enter a name for the new module or accept the default name, then click **Finish**.
   The **Module** editor appears with the **General** tab selected.



4. Enter the required and optional properties in the **Info**, **Description**, **Required Modules**, **Deployment**, **Javadoc**, and **Core JARs** sections, as described in Table 38, page 136.

Table 38. Properties in General tab

Property	Description
Info	
Name	A string specifying the name of the module. Required parameter. Enter the module name associated with the module's type, as follows: • **SBO** module — Enter the fully qualified primary interface name of the SBO. • **TBO** module — Enter the name of the corresponding object type. The name can have up to 255 characters.

Property	Description
Type	A string specifying the type of the module. Required parameter. Enter the module type or select the type from the drop-down list. Composer provides the following standard module types: • **Standard Module** — Provides a generic module. • **TBO** — Provides functionality that is specific to an object type. For example, a TBO can be used to validate the title, subject, and keywords properties of a custom document subtype. • **SBO** — Provides functionality that is not specific to a particular object type or repository. For example, an SBO can be used to customize a user's inbox.
Description	
Author	Contact information for the module author. Optional parameter.
Description	Description for the module, not exceeding 255 characters. Optional parameter.
Required Modules	Specifies modules that this module requires to function properly. Click **Add** to open the **Module Artifact** dialog. Select a module from the listbox and click **OK**, or click **New** to create a new module.
Javadoc	Specifies Javadocs and other resources that can be downloaded with the module at runtime. Click **Select** to open the **SysObject Subtype Artifact** dialog. Select a SysObject that contains the Javadoc or resource content from the list or click **New** to create a SysObject that contains the content to be downloaded. The Java doc must be a zip file with content.
Core JARs	
Implementation JARs	Implementation of the module. Required parameter. Click **Add** to add implementation JARs from your local machine.

Property	Description
Class name	Primary Java implementation class for the module. Required parameter. TBOs must implement the IDfBusinessObject interface, SBOs must implement the IDfService interface, and all modules must implement the IDfModule interface.
Interface JARs	Java interfaces that this module implements. Optional parameter. Click **Add** to add interface JARs from your local machine.

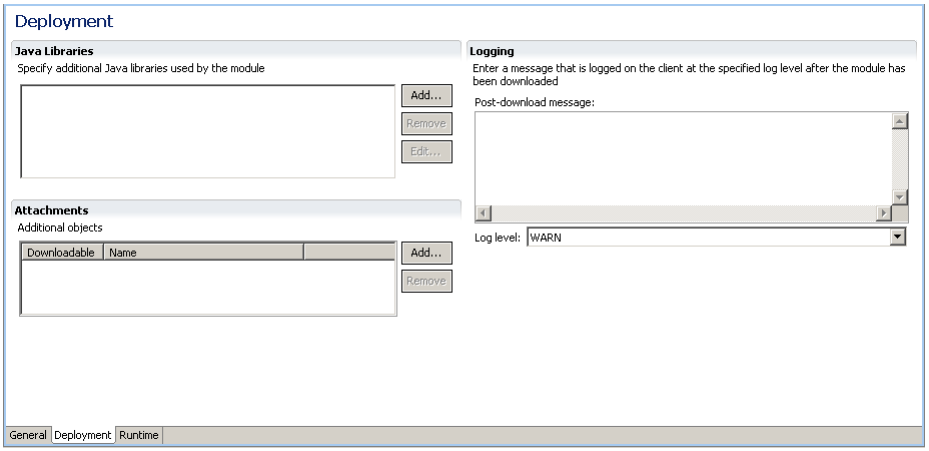
5. Click the Deployment tab and configure the module deployment options as described in Configuring module deployment , page 138.
6. Click the Runtime tab to configure the runtime environment for this module, as described in Configuring the module runtime environment, page 139.

Configuring module deployment

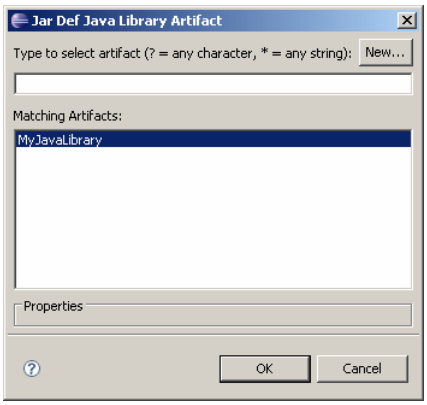
The **Deployment** tab lets you link Java libraries and other modules to the module you are creating or editing.

To configure module deployment:

1. Click the **Deployment** tab in the **Module** editor.
   The **Deployment** view appears.



2. In the **Java Libraries** section, click **Add** to add Java libraries for this module.
   The **Jar Def Java Library Artifact** dialog appears.



Select a Java library from the listbox and click **OK** or click **New** to create a new Java Library. You can only link existing Java libraries into this module. You cannot modify an existing library that is shared by multiple modules. For more information about creating a new Java Library, see Linking and configuring a Java Library, page 103.

3. In the **Attachments** section, specify Javadocs and other resources that should be available for download when the module is deployed.
4. In the **Logging** section, specify a post-download message and select a log level for the message.

   The log level can have the following values:

   - **WARN** — The post-download message is logged as a warning.
   - **NONE** — The post-download message is not logged.
   - **INFO** — The post-download message is logged as an informational message.
   - **DEBUG** — The post-download message is logged at debug level.
5. Save your changes.

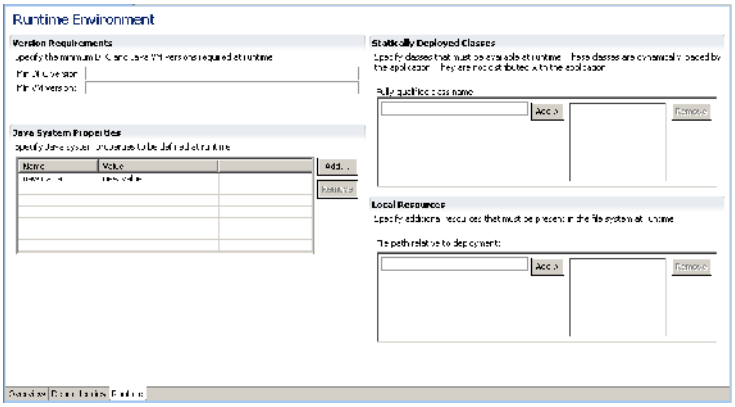
Configuring the module runtime environment

The runtime environment lets you configure optional properties that are executed at runtime, such as version requirements, Java system properties, statically deployed classes, and local resources.

To configure the runtime environment:

1. Click the **Runtime** tab in the **Module** editor.

   The **Runtime** view appears.



2. Specify the version requirements, Java system properties, statically deployed classes, and local resources, as described in Table 39, page 140.

Table 39. Module runtime environment properties

Property	Description
Version Requirements	This section lets you specify the DFC and Java VM versions required on the client for the module to function properly.
Min DFC version	The minimum DFC version on the client machine for this module to work properly.
Min VM version	The minimum Java VM version on the client machine for this module to work properly.
Java System Properties	This section lets you specify Java system properties as name-value pairs. When the module is downloaded, the client machine is checked to see if all the specified Java properties match the properties on the client machine. Click **Add** to enter placeholders for the name and value of the Java system property, then click the **Name** and the **Value** fields to modify the property name and value.
Name	Name of the Java system property.
Value	Corresponding value for the Java system property name.
Statically Deployed Classes	This section lets you specify static Java classes that are required for the module to function properly. When the module is downloaded, the class path is checked for the specified Java classes.
Fully qualified class name	Fully qualified Java class names. Enter the class name and click **Add**.

Property	Description
Local Resources	This section lets you specify files that are required on the local machine for the module to function properly. When the module is downloaded, the client machine is checked for the specified files specified.
File path relative to deployment	Full file path. Enter the file name and path and click **Add**.

3. Save your changes.

Chapter 12

Managing Permission Sets (ACLs)

This chapter contains the following topics:

Permissions, permission sets, and permission set templates

Access to folders and documents in a repository is subject to an organization's security restrictions. All content in the repository is associated with object permissions, which determine the access users have to each object in the repository such as a file, folder, or cabinet and governs their ability to perform specific actions. There are two categories of object permissions:

- **Basic**—Required for each object in the repository.
- **Extended**—Optional.

Permission sets (also known as access control lists, or ACLs) are configurations of basic and extended permissions assigned to objects in the repository that lists users and user groups and the actions they can perform. Each repository object has a permission set that defines the object-level permissions applied to it, including who can access the object. Depending on the permissions, users can create new objects, perform file-management actions such as importing, copying, or linking files, and start processes, such as sending files to workflows.

ACLs are the mechanism that Content Server uses to impose object-level permissions on SysObjects. A permission set has one or more entries that identify a user or group and the object-level permissions assigned to user or group. Composer lets you create permission set templates, regular, and public ACLs, as follows:

- **Template**—Creates a permission set template. Template ACLs are used to make applications, workflows, and lifecycles portable. For example, an application that uses a template ACL could be used by a variety of departments within an enterprise because the users or groups within the ACL entries are not defined until the ACL is assigned to an actual document.
- **Public**—Creates a public ACL that can be used by anyone in a repository. Public ACLs are available for use by any user in the repository.
- **Regular**—Creates a regular ACL that can only be used by the user or group that creates it.

Basic permissions

Basic permissions grant the ability to access and manipulate an object's content. The seven basic permission levels are hierarchical and each higher access level includes the capabilities of the preceding access levels. For example, a user with Relate permission also has Read and Browse. The basic permission are described in Table 40, page 144.

Table 40. Basic permissions

Basic Permission	Description
None	No access to the object is permitted.
Browse	Users can view the object's properties but not the object's content.
Read	Users can view both the properties and content of the object.
Relate	Users can do the above and add annotations to the object.
Version	Users can do the above and modify the object's content and check in a new version of the item (with a new version number). Users cannot overwrite an existing version or edit the item's properties.
Write	Users can do the above and edit object properties and check in the object as the same version.
Delete	Users can do all the above and delete objects.

Extended permissions

Extended permissions are optional, grant the ability to perform specific actions against an object, and are assigned in addition to basic permissions. The six levels of extended permissions are not hierarchical, so each must be assigned explicitly. The extended permissions are described in Table 41, page 144.

Table 41. Extended permissions

Extended Permission	Description
Execute Procedure	Superusers can change the owner of an item and use Run Procedure to run external procedures on certain object types. A procedure is a Docbasic program stored in the repository as a dm_procedure object.
Change Location	Users can move an object from one folder to another in the repository. A user also must have Write permission to move the object. To link an object, a user also must have Browse permission.
Change State	Users can change the state of an item with a lifecycle applied to it.
Change Permissions	Users can modify the basic permissions of an object.

Extended Permission	Description
Change Ownership	Users can change the owner of the object. If the user is not the object owner or a Superuser, they also must have Write permission.
Extended Delete	Users can only delete the object. For example, you may want a user to delete documents but not read them. This is useful for Records Management applications where discrete permissions are common.

Creating a permission set template

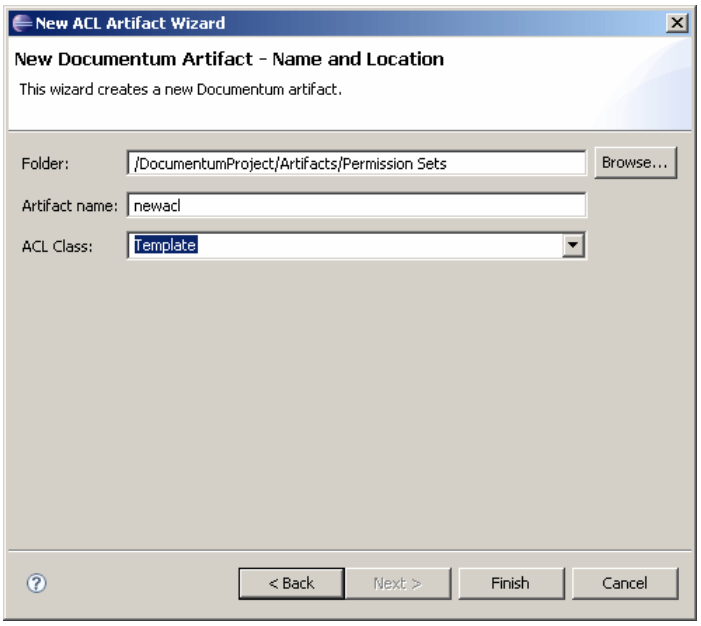
Template ACLs are used to make applications, workflows, and lifecycles portable. For example, an application that uses a template ACL could be used by a variety of departments within an enterprise because the users or groups within the ACL entries are not defined until the ACL is assigned to an actual document.

To create a permission set template:

1. Open the **Select a wizard** dialog in one of the following ways:
   - From the Composer menu, select **File > New > Other**.
   - In your Composer project, expand the **Artifacts** folder and right-click **Permission Sets**. Select **New > Other.**

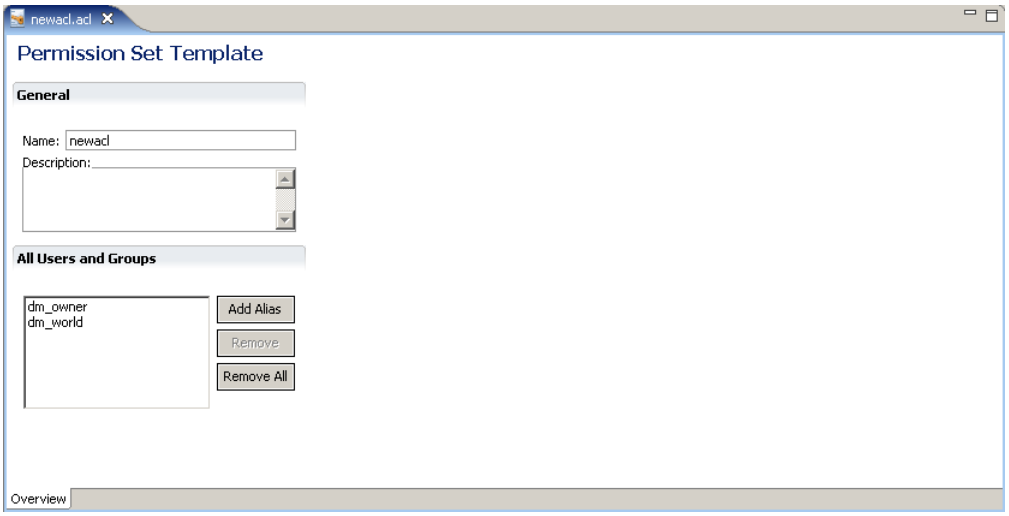
   The **Select a wizard** dialog appears.
2. Select **Documentum > Permission Set**, then click **Next**.

   The **New ACL Artifact** wizard appears.



3. Enter the folder path and name of the project for which you want to create a permission set in the **Folder:** field, or click **Browse** to select the project from a folder list.

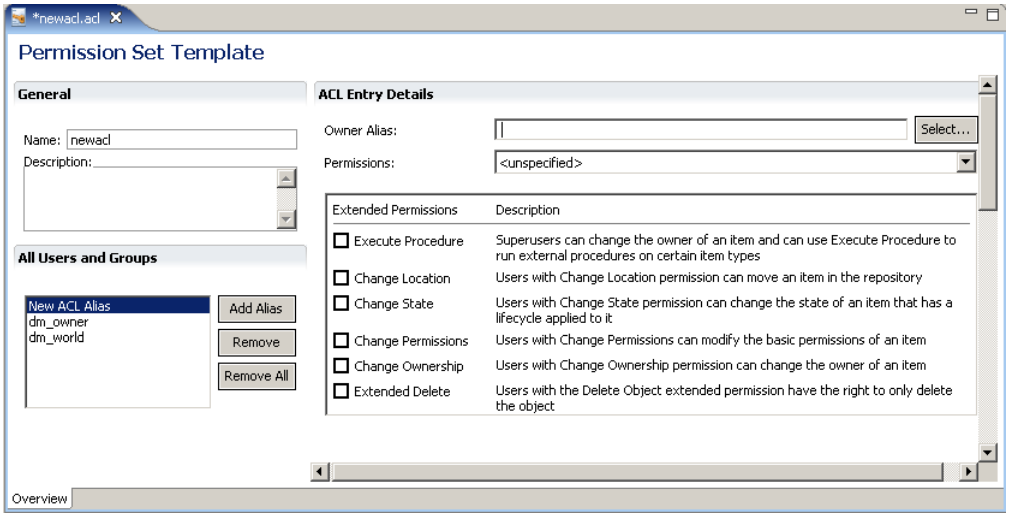
4. Enter a name for the permission set in the **Artifact name:** field.
5. Verify that **Template** is selected in the **ACL Class** field, then click **Finish**.
   The **Permission Set Template** editor appears.



   The new permission set contains two default alias entries in the **All Users and Groups** section, as follows:

   - **dm_owner**—The owner of the permission set template.
   - **dm_world**—All repository users.

   You cannot delete these default entries from a permission set template.
6. Enter a name and an optional description for the permission set template in the **General** section.
7. Select the **dm_user** or **dm_world** alias in the **All Users and Groups** section or click **Add Alias** to add a new alias.
   The **ACL Entry Details** section appears.



8. Specify the name and permissions for the alias you selected, as described in Table 42, page 147.

Table 42. ACL entry details – Permission Set Template

Parameter	Description
Owner Alias	A string specifying the owner for the alias. Click **Select** to select an owner for the ACL entry. The **Documentum AliasSet Artifact** dialog appears. Select an alias owner from the list. If the list is empty, you need to create an alias first. For more information about creating an alias, see Creating an alias set, page 75.
Permissions	Specifies the permissions for the alias. Select a permission from the drop-down list and optionally assign extended permission by checking the associated option in the **Extended Permissions** column. For more information about permissions, see Basic permissions, page 144 and Extended permissions, page 144.

9. Save your changes.

Creating a regular or a public permission set

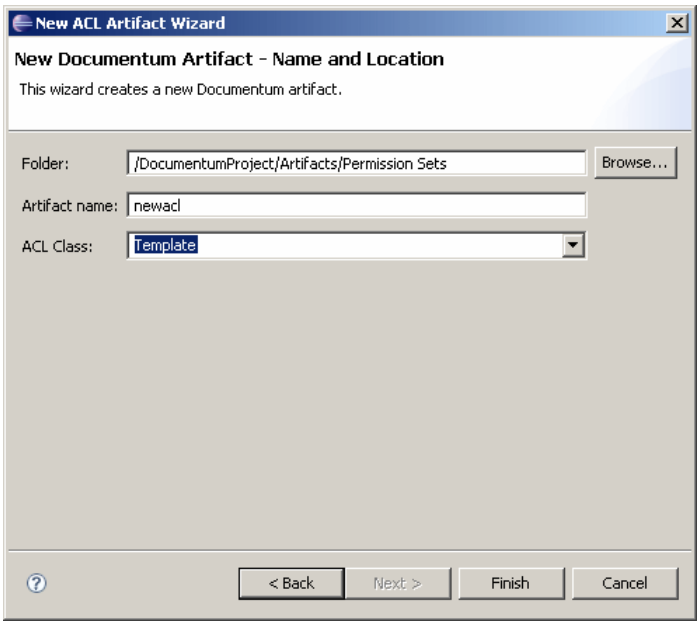
Regular ACLs can only be used by the user or group that creates it, while public ACLs can be used by any user or group in the repository.

To create a regular or a public permission set:

1. Open the **Select a wizard** dialog in one of the following ways:
   - From the Composer menu, select **File > New > Other**.
   - In your Documentum project, expand the **Artifacts** folder and right-click **Permission Sets**. Select **New > Other.**

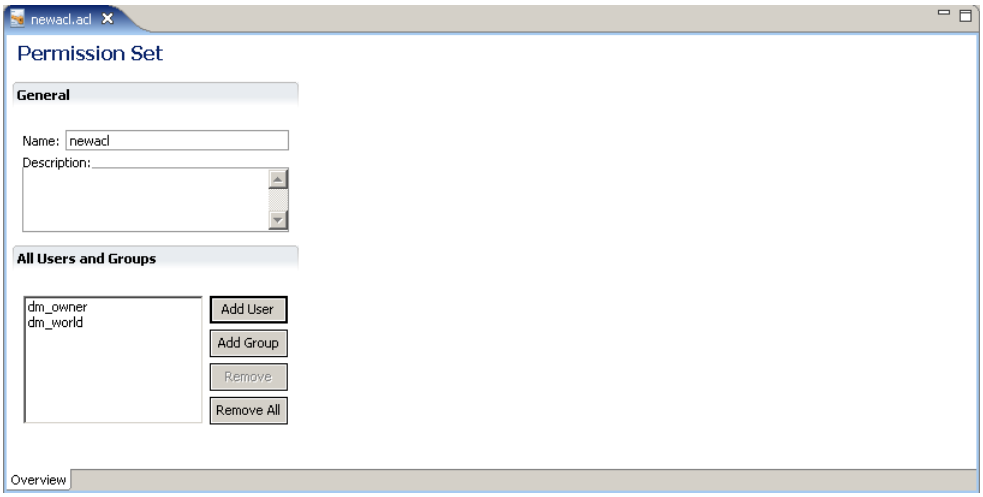
   The **Select a wizard** dialog appears.
2. Select **Documentum > Permission Set**, then click **Next**.

   The **New ACL Artifact Wizard** appears.



3. Enter the folder path and name of the project for which you want to create a permission set in the **Folder:** field, or click **Browse** to select the project from a folder list.
4. Enter a name for the permission set in the **Artifact name:** field.
5. Select **Regular** or **Public** from the **ACL class** drop-down list, depending on what type of ACL you want to create, then click **Finish**.

   The **Permission Set** editor appears.



   The new permission set contains two default ACL entries in the **All Users and Groups** section, as follows:

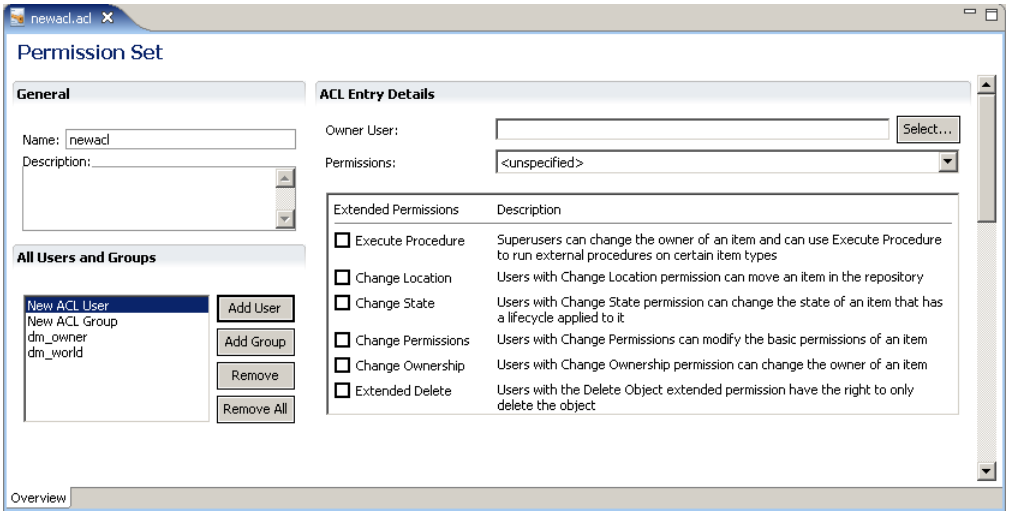
   - **dm_owner**—The owner of the permission set.
   - **dm_world**—All repository users.

   You cannot delete these default entries from a permission set.

6. Enter a name and an optional description for the permission set in the **General** section.

7. Select the dm_user or dm_world ACL entry in the **All Users and Groups** section or click
   - **Add User** to add a new user ACL
   - **Add Group** to add a new group ACL

   The **ACL Entry Details** section appears.



8. Specify the name and permissions for the ACL entry you selected, as described in Table 43, page 149.

Table 43. ACL entry details – Permission Set

Parameter	Description
Owner UserOwner Group	A string specifying the owner for the ACL entry. Click **Select** to select an owner for the ACL entry. The **User Installation Parameter** or **Group Installation Parameter** dialog appears, depending on whether you are adding a user ACL or a group ACL. Select an owner from the list. If the listbox in the **Installation Parameter** dialog is empty or does not contain the desired user or group, you need to create an owner in the form of a user or group installation parameter. You cannot add users or groups directly to the ACL. You must create an installation parameter for each group or user that you want to add to the ACL and specify the value of the group or user in that installation parameter. You can then specify the installation parameter in the ACL. For more information about creating an ACL entry owner, see Creating an ACL entry owner, page 150.
Permissions	Specifies the permissions for the ACL entry. Select a permission from the drop-down list and optionally assign extended permission by checking the associated option in the **Extended Permissions** column. For more information about permissions, see Basic permissions, page 144 and Extended permissions, page 144.

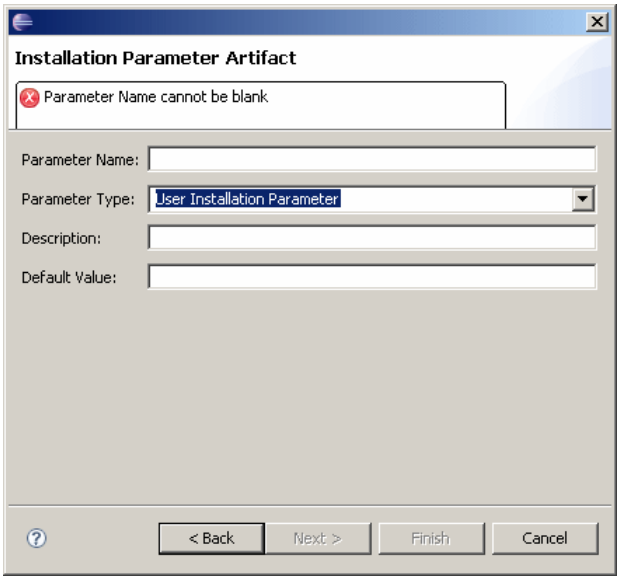
9. Save your changes.

Creating an ACL entry owner

Every ACL entry requires an owner, which can be either a user or a group of users. Composer lets you create an owner parameter in form of a user or a group installation parameter. The parameter can be mapped to the associated owner in a repository when the project is installed into the repository.

To create a user or group owner for an ACL entry:

1. Click **Select** in the **ACL Entry Details** section of the **Permission Set** editor.
   The **User or Group Installation Parameter** dialog appears.
2. Click **New**.
   The **New Documentum Artifact** dialog appears.
3. Accept the default folder location and artifact name, and click **Next**.
   The **Installation Artifact** dialog appears.



4. Enter a name for the ACL entry owner in the **Parameter name** field. You may also enter an optional description and a default value.
5. Click **Next**. The new owner name appears in the **Matching artifacts** list.
6. Click **OK** to save your changes.

Chapter 13

Managing Procedures

This chapter contains the following topics:

Procedures

Procedures are applications that extend or customize the behavior of Documentum clients or Content Server. Depending on where they are stored in the repository, procedures can be executed automatically when a user connects to a repository, or on demand when users select a menu item. Procedures are written in a proprietary Documentum language called Docbasic, which is based on Visual Basic.

For more information about Docbasic, see the *Docbasic Reference Manual*.

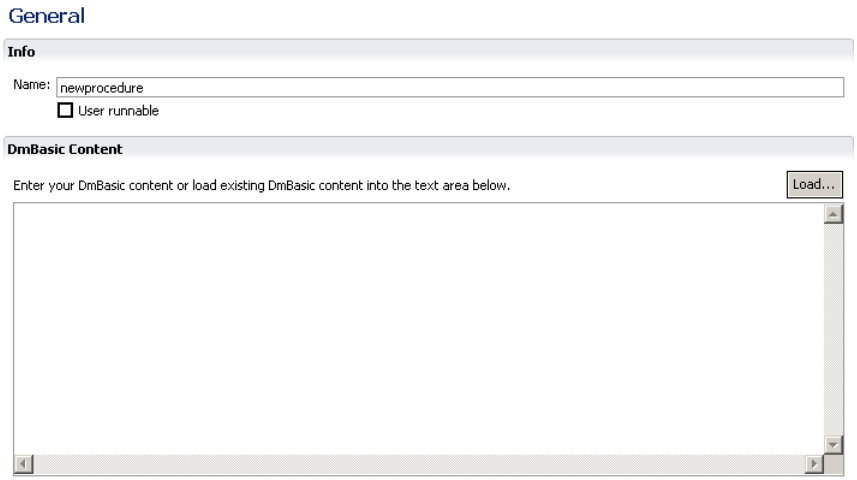
Creating a procedure

Use the **Procedure** editor to create a procedure.

To create a new procedure:

1. Open the **Select a wizard** dialog in one of the following ways:
   - From the Composer menu, select **File > New > Other**.
   - In your Composer project, expand the **Artifacts** folder and right-click **Procedures**. Select **New > Other.**

   The **Select a wizard** dialog appears.
2. Select **Documentum > Procedure**, then click **Next**.
   The **New Documentum Resource - Name and Location** dialog appears.
3. Enter the folder path and name of the project for which you want to create a procedure in the **Folder:** field, or click **Browse** to select the project from a folder list.
4. Enter a file name for the procedure in the **Artifact name:** field, then click **Finish**.
   The **Procedure** editor appears.



5. Enter a name for the procedure in the **Name** field or accept the default name.
6. Check the **User runnable** checkbox if a user is allowed to execute the procedure in the associated client application.
7. Enter the Docbasic code for the procedure in the **Docbasic Content** section or click **Load** to load the procedure code from a local file.

Chapter 14

Managing Relation Types

This chapter contains the following topics:

Relation types

A relation type defines the relationship between two objects in a repository. In general, when two objects are connected by a relationship, one is considered the parent objects and the other is considered the child.

A relation type describes how one item is related to another. There are two relation types, as follows:

- **Ad hoc**

  This relation type can be added, modified and deleted by users.

- **System**

  This relation type cannot be manipulated by users. For example, a relationship between a file and its thumbnail is a system relation type.

Creating a relation type

Use the **Relation Type** editor to create a new relation type or modify an existing relation type.

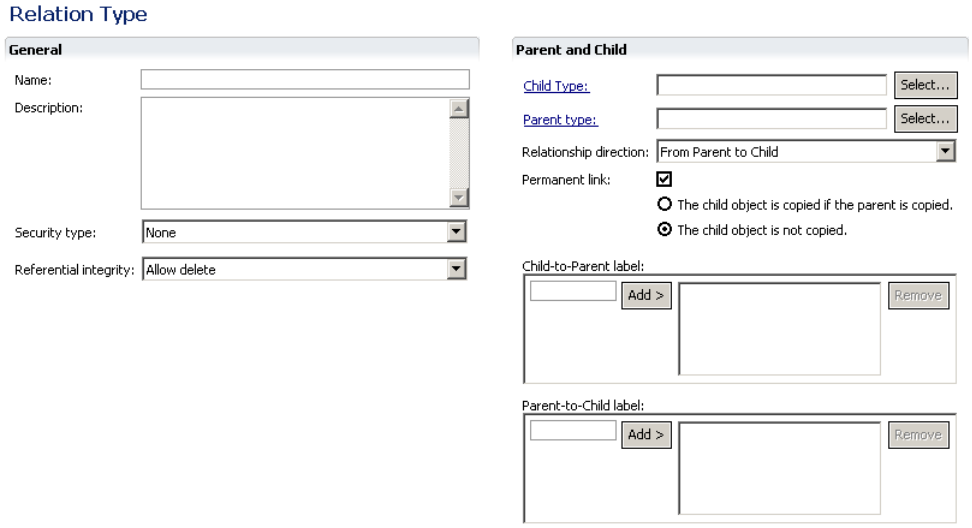
To create a relation type:

1. Open the **Select a wizard** dialog in one of the following ways:
   - From the Composer menu, select **File > New > Other**.
   - In your Composer project, expand the **Artifacts** folder and right-click **Relation Types**. Select **New > Other.**

   The **Select a wizard** dialog appears.
2. Select **Documentum > Relation Type**, then click **Next**.

   The **New Documentum Resource - Name and Location** dialog appears.

3. Enter the folder path and name of the project for which you want to create a relation type in the **Folder:** field, or click **Browse** to select the project from a folder list.
4. Enter a file name for the relation type in the **Artifact name:** field, then click **Next**.
5. Enter a name for the relation type in the **Relation type name:** field, then click **Finish**.
   The **Relation Type** editor appears.



6. Enter the relation type properties in the **General** and **Parent and Child** sections, as described in Table 44, page 154

Table 44. Relation type properties

Property	Description
General	
Name	A string specifying the name of the relation type. The name can be up to 255 characters long.
Description	A string describing the relation type. The description can be up to 250 characters long.
Security type	A string specifying the security level for the relation type. The security type can have the following values: • **System** — Only users with superuser or system administrator privileges can create, modify, or drop this relation type. • **Parent** — This relation type inherits the security level from its parent. • **Child** — This relation type inherits the security level from its child. • **None** — This relation type has no security level.

Property	Description
Referential integrity	Specifies how the referential integrity is enforced. Select one of the following values from the drop-down list: • **Allow delete** • **Restrict delete** • **Cascade delete** The default referential integrity value is **Allow delete**.
Parent and Child	
Child type	A string with a maximum of 32 characters specifying the object type for a child object. The child type is an optional relation type property. You can specify a child type in one of the following ways: • Type the name of the child type in the **Child type:** field. • Click **Select** to select a child type. The **Select Type Artifact** dialog appears. Select a child type from the listbox. • Click the **Child type:** link to create a new child type. The **Documentum Artifact - Name and Location** wizard appears. For more information about creating a new artifact, see Creating an artifact , page 33.
Parent type	A string with a maximum of 32 characters specifying the object type of a valid parent object. The parent type is an optional relation type property. You can specify a child type in one of the following ways: • Type the name of the parent type in the **Parent type:** field. • Click **Select** to select a parent type. The **Artifact Selector** dialog appears. Select a parent type from the listbox. • Click the **Parent type:** link to create a new parent type. The **New Documentum Artifact - Name and Location** wizard appears. For more information about creating a new artifact, see Creating an artifact , page 33.
Relationship direction	An integer specifying the relationship direction. The relationship direction can have the following values: • **From Parent to Child** • **From Child to Parent** • **Bidirectional** The default relationship direction value is **From Parent to Child**.

Property	Description
Permanent link	Determines whether the relationship is maintained when the parent is copied or versioned. Select this option to maintain the relationship between parent and child, in one of the following ways: • The child object is copied if the parent is copied. • The child object is not copied.
Child-to-parent label	A string with up to 255 characters specifying a label for the child-to-parent relationship. Type the string in the text field and click **Add**.
Parent-to-child label	A string with up to 255 characters specifying a label for the parent-to-child relationship. Type the string in the text field and click **Add**.

Chapter 15

Managing Smart Containers

This chapter contains the following topics:

Smart containers

Smart containers define objects and relationships in a template that is used to instantiate instances at runtime. Composer provides a smart container editor that lets developers construct smart containers declaratively instead of programmatically, thus greatly reducing the time to write DFC applications. After a smart container has been constructed the objects are similar to Documentum persistent objects.

Because a smart container template is intended to be used for repeated construction of a modeled composite object, each new instance of the composite object should be different. This is accomplished by parameterizing the smart container at design time.

Note: When configuring the installation options for a smart container artifact, be sure to set the **Upgrade option** to **Create New Version Of Matching Objects**. Do not select **Overwrite Matching Objects** for smart container artifacts because overwriting smart container objects invalidates the model-instance association for existing instances.

Constructing a smart container

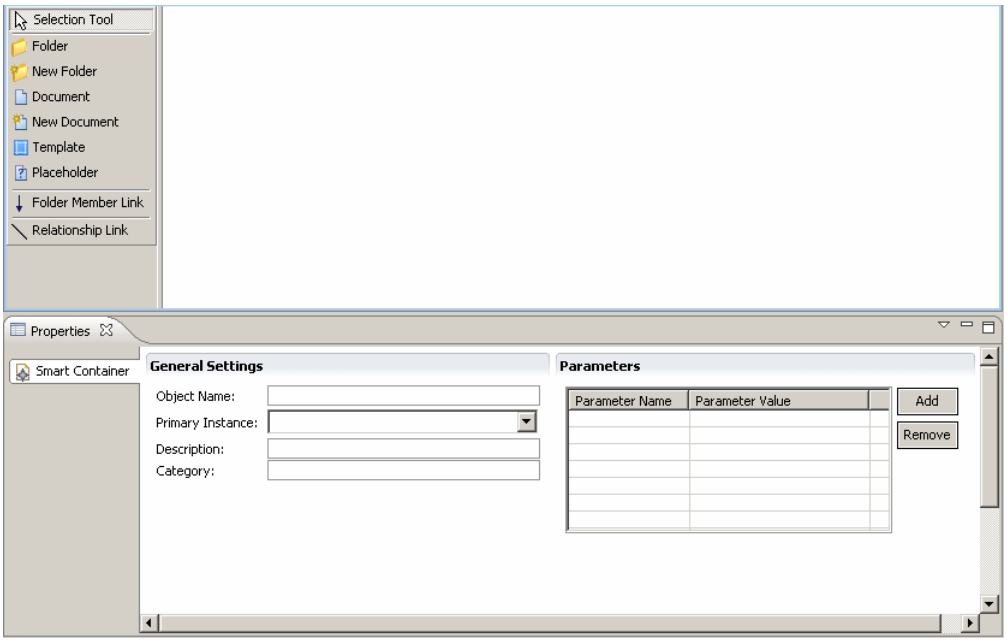
Use the **Smart Container** editor to construct or modify a smart container.

To construct a smart container:

1. Open the **Select a wizard** dialog in one of the following ways:
   - From the Composer menu, select **File > New > Other**.
   - In your Composer project, expand the **Artifacts** folder and right-click **Types**. Select **New > Other.**
2. Select **Documentum > Smart Container**, then click **Next**.

   The **New Documentum Resource - Name and Location** dialog appears.
3. Enter the folder path and name of the project for which you want to construct a smart container in the **Folder:** field, or click **Browse** to select the project and folder path from a folder list.
4. Enter a file name for smart container in the **Artifact name:** field, then click **Next**.

   The **Smart Container** editor appears.



5. Configure the smart container properties, as described in Table 45, page 158.

Table 45. Smart container properties

Parameter	Description
General Settings	
Object Name	A string specifying the name of the smart container. You can either accept the default name that is assigned by Composer or enter a new name.
Primary Instance	Specifies the primary object of the smart container. Select a primary instance from the drop-down list. Every smart container must have exactly one primary instance. The primary instance can be a new folder, existing folder, new document, existing document, or a template, but not a placeholder.

Parameter	Description
Description	A description of the smart container.
Category	A string specifying a discriminator that can be used in a filter, for example in drop-down lists.
Parameters	
Parameter Name	The name of a parameter that is used by the smart container at runtime.
Default Value	The value of the runtime parameter.

6. Add one or more artifacts to your smart container, as follows:
   - **Folder**, as described in Adding a folder , page 159
   - **New folder**, as described in Adding a new folder , page 160
   - **Document**, as described in Adding a document , page 161
   - **New document**, as described in Adding a new document, page 162
   - **Template**, as described in Adding a template , page 162
   - **Placeholder**, as described in Adding a placeholder, page 163
7. Add relationships to your artifacts, as described in Adding smart container relationships, page 164.
8. Save your changes.
9. Configure the artifact installation parameters for the smart container, as described in Configuring artifact install options, page 197.

   Note: Be sure to set the upgrade option in the installation parameters to **Create New Version Of Matching Objects**. Do not select **Overwrite Matching Objects** for smart container artifacts because overwriting smart container objects invalidates the model-instance association for existing instances.

Adding smart container elements

A smart container can contain various different elements, such as folders, documents, templates, and placeholders.

Adding a folder

Use the **Folder** option in the smart container editor to add an instance of an existing folder to the smart container. Since you are adding a folder that already exists in a repository, you must define the folder by adding either the folder's Documentum object id or path.

To add a folder:

1. Open the smart container editor, as described in Constructing a smart container, page 157.
2. Select the  **Folder** icon and click the workspace. The folder appears in the smart container workspace.
3. Click the **Folder Info** tab in the **Properties** view and define the folder properties, as described in Table 46, page 160.

Table 46. Folder properties

Parameter	Description
Display Name	The name of the folder that appears in the workspace. This name for display purposes only, so folders can be distinguished in the workspace. The display name is not used in a repository or any other application.
Object Id	The 16-character Documentum object id of the folder.
Path	The relative path for a location to which the folder is linked.

Adding a new folder

A new smart container folder is similar to a regular new folder with the exception that a new smart container folder does not get instantiated until runtime.

To add a new folder:

1. Open the **Smart Container** editor, as described in Constructing a smart container, page 157.
2. Select the  **New Folder** icon and click the workspace. A new folder appears in the smart container workspace.
3. Click the **New Folder Info** tab in the **Properties** view and define the folder properties, as described in Table 47, page 160.

Table 47. New folder properties

Parameter	Description
Display Name	The name of the folder that appears in the workspace. This name for display purposes only, so folders can be distinguished in the workspace. The display name is not used in a repository or any other application.
Object Name	The object name of the new folder.

Parameter	Description
Type	The object type of the new folder. Click **Select** and select an object type from the list, or click the **Type** link to create a new object type. For more information about types, see Chapter 17, Managing Types .
Permission Set	The permission set assigned to the folder. Click **Select** and select a permission set from the list, or click the **Permission Set** link to create a new permission set. For more information about permission sets, see Chapter 12, Managing Permission Sets (ACLs).

4. Click the **Aspects Tab** to attach one or more aspects to the new folder. Click **Add** and select an aspect from the list or create a new aspect. For more information about aspects, see Chapter 6, Managing Aspects.
5. Click the **Attributes Tab** to add one or more attributes to the new folder. Click **Add** and select an attribute from the list.
6. Save your changes.

Adding a document

Use the **Document** option in the smart container editor to add an instance of an existing document to the smart container. Since you are adding a document that already exists in a repository, you must define the document by adding either the document's Documentum object id or path.

To add a document:

1. Open the **Smart Container** editor, as described in Constructing a smart container, page 157.
2. Select the  **Document** icon and click the workspace. The document appears in the smart container workspace.
3. Click the **Instance Info** tab in the **Properties** view and define the document properties, as described in Table 48, page 161.

Table 48. Document instance properties

Parameter	Description
Display Name	The name of the document instance that appears in the workspace. This name for display purposes only, so document instances can be distinguished in the workspace. The display name is not used in a repository or any other application.
Object Id	The 16-character Documentum object id of the document instance.
Path	The relative path for a location to which the document instance is linked.

Adding a new document

A new smart container document is similar to a regular new document with the exception that a new smart container document does not get instantiated until runtime.

To add a new document:

1. Open the **Smart Container** editor, as described in Constructing a smart container, page 157.
2. Select the  **New Document** icon and click the workspace. A new document appears in the smart container workspace.
3. Click the **New Instance Info** tab in the **Properties** view and define the new document properties, as described in Table 49, page 162.

Table 49. New document instance properties

Parameter	Description
Display Name	The name of the new document instance that appears in the workspace. This name for display purposes only, so new document instances can be distinguished in the workspace. The display name is not used in a repository or any other application.
Object Name	The object name of the new document instance.
Type	The object type of the new document instance. Click **Select** and select an object type from the list, or click the **Type** link to create a new object type. For more information about types, see Chapter 17, Managing Types .
Permission Set	The permission set assigned to the document instance. Click **Select** and select a permission set from the list, or click the **Permission Set** link to create a new permission set. For more information about permission sets, see Chapter 12, Managing Permission Sets (ACLs).

4. Click the **Aspects** tab to attach one or more aspects to the new document instance. Click **Add** and select an aspect from the list or create a new aspect. For more information about aspects, see Chapter 6, Managing Aspects.
5. Click the **Attributes** tab to add one or more attributes to the new document instance. Click **Add** and select an attribute from the list.
6. Save your changes.

Adding a template

A smart container template is an existing document that you want to have copied into your smart container at construction.

To add a template:

1. Open the **Smart Container** editor, as described in Constructing a smart container, page 157.

2. Select the  **Template** icon and click the workspace. The template appears in the smart container workspace.
3. Click the **Template Info** tab in the **Properties** view and define the template properties, as described in Table 50, page 163.

Table 50. Template properties

Parameter	Description
Display Name	The name of the template that appears in the workspace. This name for display purposes only, so new document instances can be distinguished in the workspace. The display name is not used in a repository or any other application.
Object Name	The object name of the template.
Permission Set	The permission set assigned to the template. Click **Select** and select a permission set from the list, or click the **Permission Set** link to create a new permission set. For more information about permission sets, see Chapter 12, Managing Permission Sets (ACLs).
Based on	
Object Id	The 16-character Documentum object id of the template.
Path	The relative path for a location to which the template is linked.

4. Click the **Aspects** tab to attach one or more aspects to the template. Click **Add** and select an aspect from the list or create a new aspect. For more information about aspects, see Chapter 6, Managing Aspects.
5. Click the **Attributes** tab to add one or more attributes to the template. Click **Add** and select an attribute from the list.
6. Save your changes.

Adding a placeholder

A placeholder object is similar to a template object, but a placeholder object is not created at construction time. A placeholder object lets a modeler indicate that other objects must be added later in order for the composite object to be considered “complete”. A placeholder must be a “leaf” node.

To add a placeholder:

1. Open the **Smart Container** editor, as described in Constructing a smart container, page 157.
2. Select the  **Placeholder** icon and click the workspace. The placeholder appears in the smart container workspace.
3. Click the **Placeholder Info** tab in the **Properties** view and define the placeholder properties, as described in Table 51, page 164.

Table 51. Placeholder properties

Parameter	Description
Display Name	The name of the placeholder that appears in the workspace. This name for display purposes only, so new placeholders can be distinguished in the workspace. The display name is not used in a repository or any other application.
Object Name	The object name of the placeholder.
Type	The object type of the placeholder. Click **Select** and select an object type from the list, or click the Type link to create a new object type. For more information about types, see Chapter 17, Managing Types .
Required	Specifies whether the placeholder must be assigned an object type.

4. Save your changes.

Adding smart container relationships

Relationships between smart container objects can take the form of folder links and generic relationships. The smart container editor provides two options to visually distinguish a relationship:

- **Folder Member Link**

  You can use the **Folder Member Link** relation when you are modeling folders and their members.

- **Relation**

  You can use the **Relation** option for all other generic relationships. For example, a relationship between an insurance claim and a customer.

To add a relationship:

1. Click the **Folder Member Link** tab or **Relation** tab to activate it.
2. Click the first smart container object.
3. Click the second smart container object.

   An arrow appears indicating that the two objects are now connected.

Chapter 16

Managing SysObjects

This chapter contains the following topics:

SysObjects

The SysObject type is the parent type of the most commonly used objects in the Documentum system. SysObjects are the supertype, directly or indirectly, of all object types in the hierarchy that can have content. The SysObject type's defined attributes store information about the object's version, the content file associated with the object, the security permissions on the object and other information important for managing content. The SysObject subtype most commonly associated with content is dm_document.

Creating a SysObject

Use the **SysObject** editor to create or modify a SysObject.

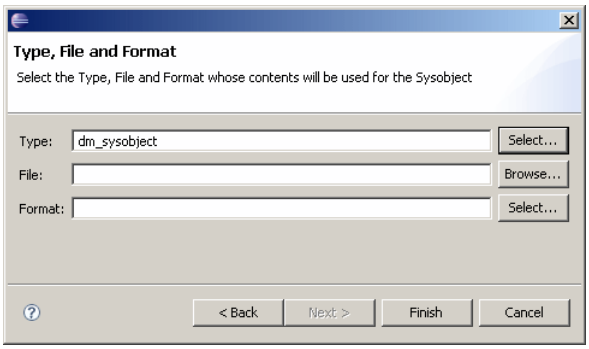
To create a SysObject:

1. Open the **Select a wizard** dialog in one of the following ways:
   - From the Composer menu, select **File > New > Other**.
   - In your Composer project, expand the **Artifacts** folder and right-click **SysObjects**. Select **New > Other.**

   The **Select a wizard** dialog appears.
2. Select **Documentum > SysObject**, then click **Next**.

   The **New Documentum Resource - Name and Location** dialog appears.
3. Enter the folder path and name of the project for which you want to create a SysObject in the **Folder:** field, or click **Browse** to select the project from a folder list.
4. Enter a file name for the SysObject in the **Artifact name:** field, then click **Next**.

The **Type, File, and Format** dialog appears.



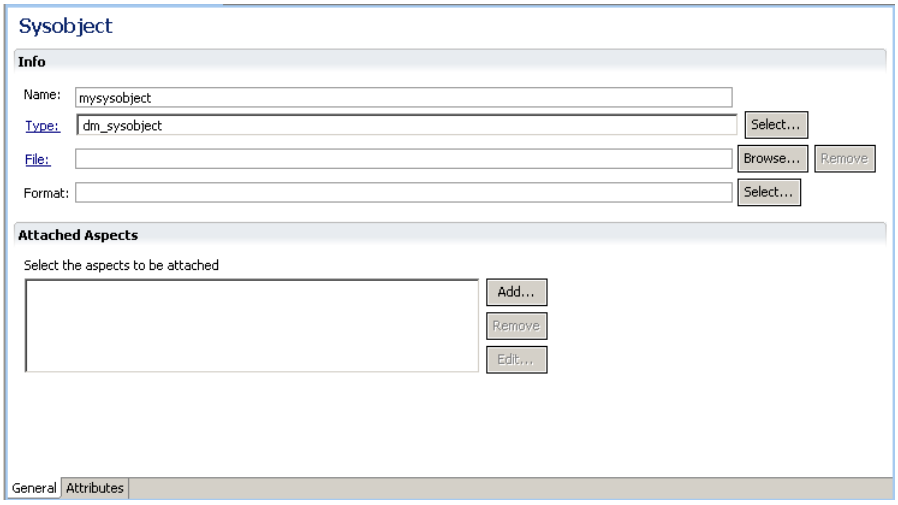
5. Enter the type, file, and format information for the SysObject, if applicable, as described in Table 52, page 166, then click **Next**.

   Note: If you do not want to enter a file name and format for the SysObject, click **Finish**.

Table 52. SysObject properties

Property	Description
Type	The object type that contains the content. By default, the object type is set to dm_sysobject. Click **Select** to select a different object type from the drop-down list.
File	The name of the file that contains the content, if applicable. Click **Browse** to select the file from your local machine or network drive.
Format	The format of the content file. Click **Select** to select a file format from the drop-down list.

The **SysObject** editor appears.



6. Click **Add** in the **Attached Aspects** section to attach one or more aspects to the SysObject.

   The **Aspect Module Artifact** dialog appears.

7. Select an aspect from the list or click **New** to create a new aspect. For more information about aspects, see Chapter 6, Managing Aspects.

8. View or modify the SysObject attributes, as described in Viewing and modifying SysObject attributes, page 167.
9. Save your changes.

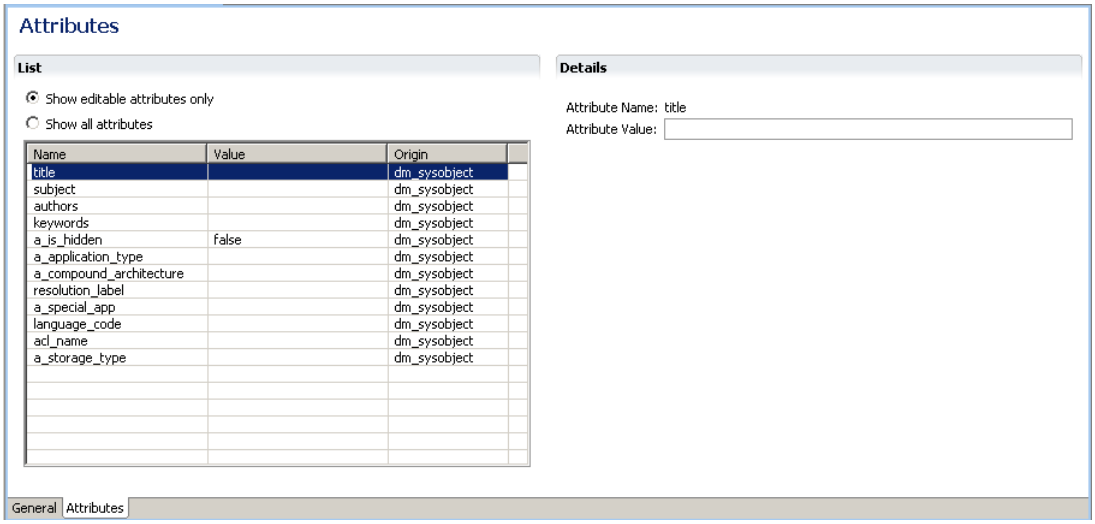
Viewing and modifying SysObject attributes

The **Attributes** tab in the **SysObject** editor lets you view attributes and modify the attribute values associated with the specified SysObject.

To view or modify SysObject attributes:

1. Click the **Attributes** tab in the **SysObject** editor.

   The **Attributes** view appears.
2. Select an attributes from the list to view the attribute details.



   In the **Attributes** view, you can do the following:
   - Select the **Show editable attributes only** radio button to list only attributes that can be modified.
   - Select the **Show all attributes** radio button to list all attributes for the SysObject.
   - Enter or modify the value of an editable attribute in **Attribute Value** field of the **Details** section.
3. Safe your changes.

Chapter 17

Managing Types

This chapter contains the following topics:

Object types

An object type is similar to a template and represents a class of objects. Composer lets you create two types:

- Standard objects
- Lightweight objects

Every object type is defined by a set of attributes. When an object is created, its attributes are set to values that describe that particular instance of the object type. For example, two attributes of the document object type are title and subject. When users create a document, they provide values for the title and subject attributes that are specific to that document. For more information about objects and object types, see *Documentum Content Server Fundamentals*.

Lightweight objects are part of an object model enhancement introduced to share system managed metadata among objects which only hold application specific data. For example, policies for security, retention, and storage, are stored in a regular system object that is shared among all the lightweight objects. Because the system managed metadata is stored only once, it significantly reduces the disk storage requirements and improves the ingestion performance.

Note: Currently only applications designed for Documentum High-Volume Server can make proper use of lightweight objects. Documentum High-Volume Server is an extension of Documentum Content Server that supports features implemented to solve common problems with large content stores, such as email archiving. It requires an additional license key specified when Content Server

is installed. For more information about lightweight object types and Documentum High-Volume Server, see *EMC Documentum High-Volume Server Developer Guide*.

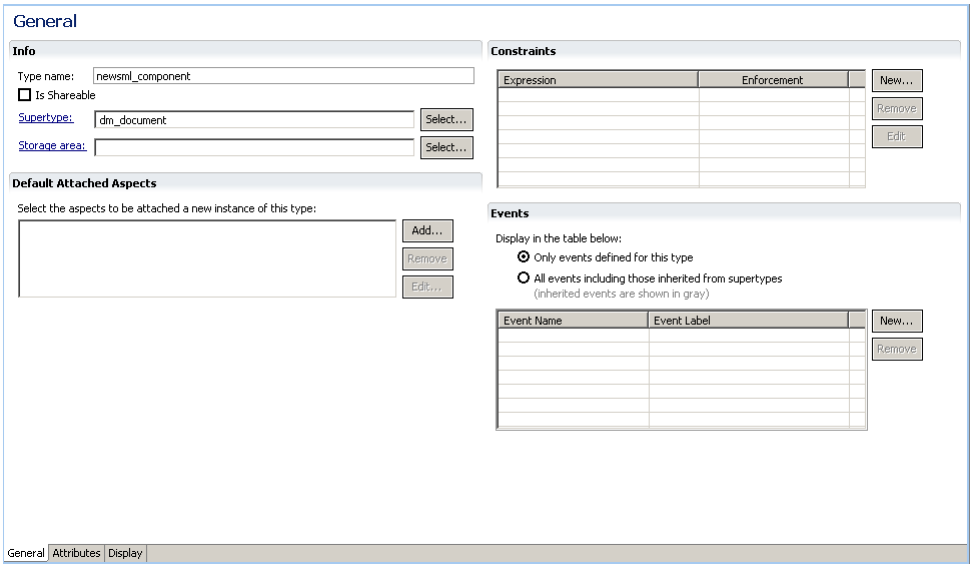
Creating a standard object type

Use the **Type** editor to create or modify a standard object type.

To create a standard object type:

1. Open the **Select a wizard** dialog in one of the following ways:
   - From the Composer menu, select **File > New > Other**.
   - In your Composer project, expand the **Artifacts** folder and right-click **Types**. Select **New > Other.**

   The **Select a wizard** dialog appears.
2. Select **Documentum > Documentum Artifact > Type**, then click **Next**.
   The **New Documentum Artifact - Name and Location** dialog appears.
3. Enter the folder path and name of the project for which you want to create an object type in the **Folder:** field, or click **Browse** to select the project from a folder list.
4. Enter a file name for the object type in the **Artifact name:** field
5. Select **Standard object type**, then click **Next**.
   The **Type** editor appears with the **General** tab selected.



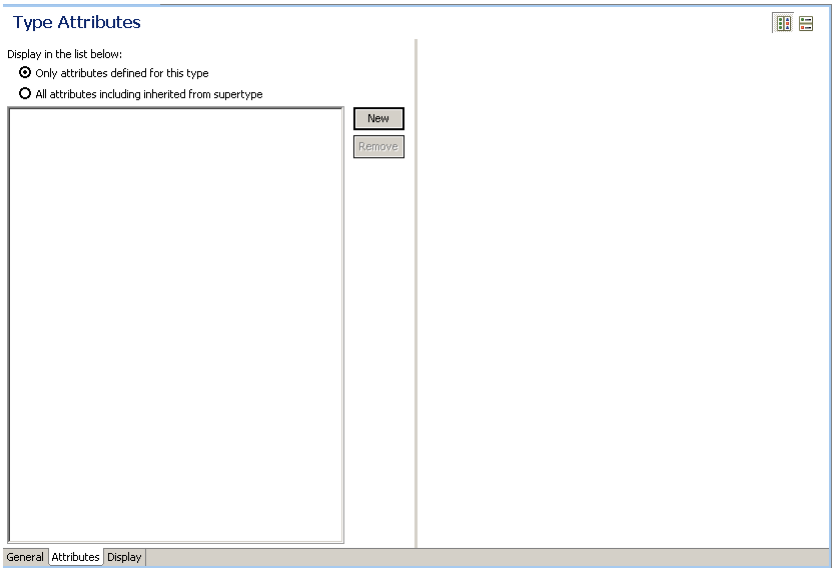
6. Enter the object type information in the **Info**, **Default Attached Aspects**, **Constraints**, and **Events** sections, as described in Table 53, page 171.

Table 53. Type information on General tab

Property	Description
General	
Type name	A string with specifying the name of the type. The following rules apply to all type names: • A maximum of 27 characters, all lower-case. The Content Server is case-insensitive and stores all type names in lower-case. • The first character must be a letter, the remaining characters can be letters, digits, or underscores • Cannot contain any spaces or punctuation • Cannot end in an underscore (_)
Is Shareable	Select this option if you want the properties of this type to be shareable with other object types.
Supertype	The supertype of the new type. A supertype is a type that is the basis for another type. The new type inherits all the properties of the specified supertype. Click **Select ...** and select a supertype from the listbox.
Storage area	Specifies the default storage location for instances of this type. If you do not assign a custom default storage location, Composer automatically assigns the system default storage location.
Default Attached Aspects	Click **Select ...** to specify one or more aspects that are attached to instances of this type. Aspects let you modify the behavior of type instances. For more information about attaching aspects, see Attaching aspects, page 173.
Constraints	Constraints are internal consistency requirements in the form of Docbasic expressions that relate the types attribute values to one another or to constant values.
Expression	The Docbasic expression defining the constraint. Click **New** to create a new expression. For more information about creating or modifying an expression, see Configuring constraint expressions, page 177.

Property	Description
Enforcement	Specifies whether applications should enforce this constraint or not. Click the table cell in the Enforcement column to enable or disable constraint enforcement for the associated expression.The enforcement field can have two values, as follows: • **disabled**: The constraint is disabled • **ApplicationEnforced**: The constraint is enforced by the applications that use this type.
Events	Events are specific actions on objects. You can only create and modify application events, not system events. Click **New** to enter a new event. To edit or remove an event, select the event and click **Edit** or **Remove**, respectively. For more information about creating a new event, see Adding, deleting or modifying events, page 178.
Only events defined for this type	Select this option to display allow events that are defined for this type in the events table.
All events including inherited from supertype	Select this option to display all events for this type in the events table, including events that are inherited from the supertype.
Event name	A string specifying the name of the event that is associated with instances of this type.
Event label	A string that specifies the label for the event.

7. Click the **Attributes** tab.
   The **Type Attributes** view appears.



8. Click **Add** to add a new type attribute. For more information about adding type attributes, see Adding type attributes, page 178.
9. Click the **Display** tab.

   The **Type UI Information** view appears (Figure 13, page 186).
10. Enter the type UI information in the **Display Configuration** and **Application Interface Display** sections, as described in Configuring the type UI information , page 186.

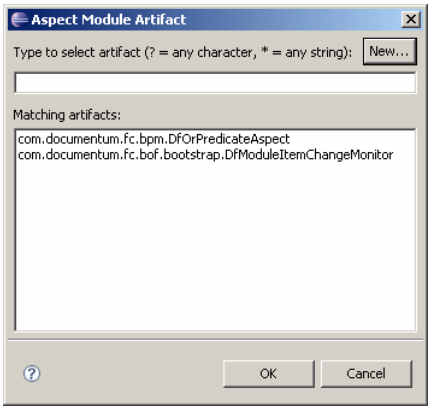
Attaching aspects

When you create a new object type, you can also assign aspects that are attached to the object type by default.

To attach an aspect:

1. Click **Add** in the **Default Attached Aspects** section of the **General** tab in the type editor.

   The **Aspect Module Artifact** dialog appears.



2. Select an aspect from the **Matching Artifacts** listbox, then click **OK**.

   Note: If there are no aspects listed in the **Matching Artifacts** listbox, no aspects have been created yet. For more information about creating an aspect, see Creating an aspect type, page 81.

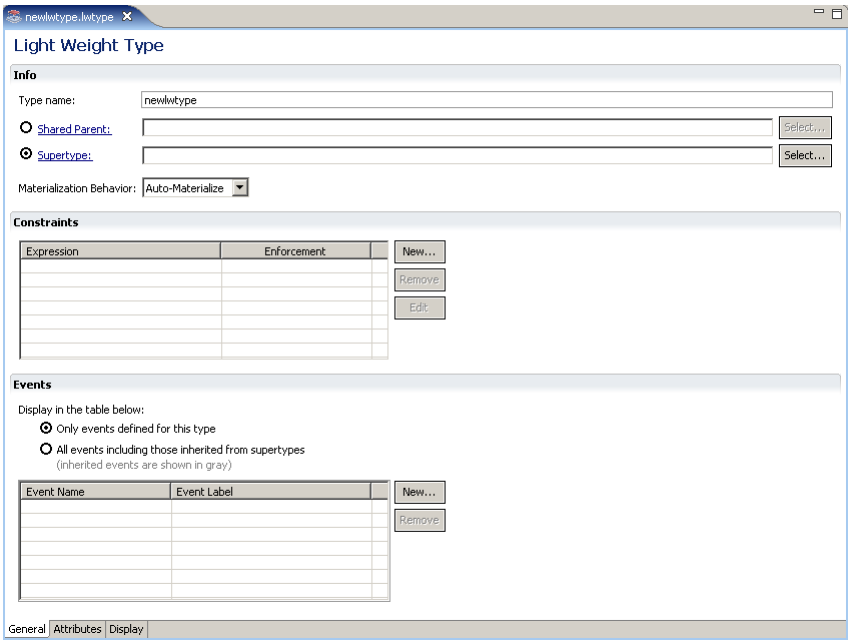
Creating a lightweight object type

Before you create a lightweight object type, verify that the application you are building can actually utilize this type of object. Currently only archiving applications designed for Documentum High-Volume Server have a use for lightweight object types. Documentum High-Volume Server is an extension of Documentum Content Server that supports features implemented to solve common problems with large content stores, such as email archiving. It requires an additional license key specified when Content Server is installed.

To create a lightweight object type:

1. Open the **Select a wizard** dialog in one of the following ways:
   - From the Composer menu, select **File > New > Other**.
   - In your Composer project, expand the **Artifacts** folder and right-click **Types**. Select **New > Other.**

   The **Select a wizard** dialog appears.
2. Select **Documentum > Documentum Artifact > Type**, then click **Next**.
   The **New Documentum Artifact - Name and Location** dialog appears.
3. Enter the folder path and name of the project for which you want to create an object type in the **Folder:** field, or click **Browse** to select the project from a folder list.
4. Enter a file name for the object type in the **Artifact name:** field
5. Select **Lightweight object type**, then click **Next**.
   The **Lightweight Type** editor appears with the **General** tab selected.



6. Enter the object type information in the **Info**, **Constraints**, and **Events** sections, as described in Table 54, page 174.

Table 54. Lightweight type information on General tab

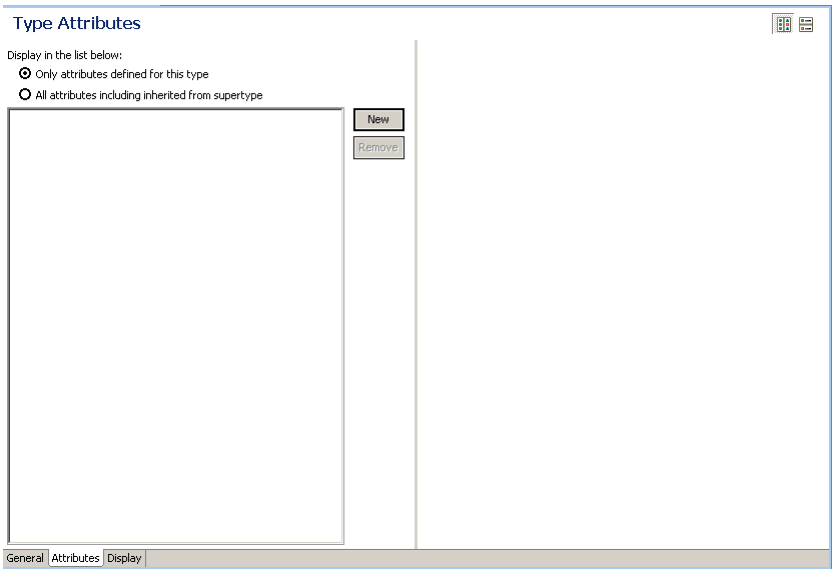
Property	Description
General	

Property	Description
Type name	A string with specifying the name of this lightweight object type. The following rules apply to all type names: • A maximum of 27 characters, all lower-case. The Content Server is case-insensitive and stores all type names in lower-case. • The first character must be a letter, the remaining characters can be letters, digits, or underscores • Cannot contain any spaces or punctuation • Cannot end in an underscore (_)
Shared Parent	The parent object type that is sharing its properties with this lightweight object.
Supertype	The supertype of this lightweight object type. A supertype is a type that is the basis for another type. The new type inherits all the properties of the specified supertype. Click **Select ...** and select a supertype from the listbox.
Materialization Behavior	Specifies when this lightweight object type shares a parent object or has its own private copy of a parent. A lightweight object is classified as unmaterialized when it shares a parent object with other lightweight objects. A lightweight object is classified as materialized, when a lightweight object has its own private copy of a parent. Materializing a lightweight object might drastically increase the size of database tables. The materialization behavior be set to the following values: • **Auto-Materialize**: The lightweight object is materialized automatically when certain operations occur, for example a checkout/checkin operation or a branch operation. • **On-Request**: The lightweight object is materialized only when requested by an explicit API call. For more information on the APIs that can be used to materialize a lightweight object, see the *EMC Documentum Archive Developer Guide.* • **Disallow**: The lightweight object is never materialized.

Property	Description
	By default the materialization behavior is set to **Auto-Materialize**.
Constraints	Constraints are internal consistency requirements in the form of Docbasic expressions that relate the types attribute values to one another or to constant values.
Expression	The Docbasic expression defining the constraint. Click **New** to create a new expression. For more information about creating or modifying an expression, see Configuring constraint expressions, page 177.
Enforcement	Specifies whether applications should enforce this constraint or not. Click the table cell in the Enforcement column to enable or disable constraint enforcement for the associated expression.The enforcement field can have two values, as follows: • **disabled**: The constraint is disabled • **ApplicationEnforced**: The constraint is enforced by the applications that use this type.
Events	Events are specific actions on objects. You can only create and modify application events, not system events. Click **New** to enter a new event. To edit or remove an event, select the event and click **Edit** or **Remove**, respectively. For more information about creating a new event, see Adding, deleting or modifying events, page 178.
Only events defined for this type	Select this option to display allow events that are defined for this type in the events table.
All events including inherited from supertype	Select this option to display all events for this type in the events table, including events that are inherited from the supertype.
Event name	A string specifying the name of the event that is associated with instances of this type.
Event label	A string that specifies the label for the event.

7. Click the **Attributes** tab.

   The **Type Attributes** view appears.



8. Click **Add** to add a new type attribute. For more information about adding type attributes, see Adding type attributes, page 178.
9. Click the **Display** tab.

   The **Type UI Information** view appears (Figure 13, page 186).
10. Enter the type UI information in the Display Configuration and Application Interface Display sections, as described in Configuring the type UI information , page 186.

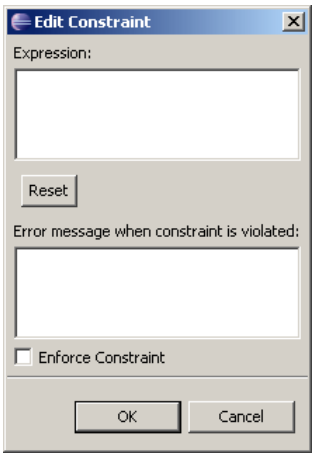
Configuring constraint expressions

Constraints are internal consistency requirements in the form of Docbasic expressions that relate the types attribute values to one another or to constant values.

To add a constraint expression for a type:

1. Click **Add** in the **Constraints** section of the **Type Overview** tab in the type editor.

   The **Edit Constraint** dialog appears.



2. Type a valid Docbasic constraint expression that resolves to true or false in the **Expression:** text box. The Docbasic expression should resolve to true when the constraint is fulfilled and false when the constraint is violated.

   The **Reset** button returns the contents of the **Constraint Expression** text box to the last value that passed a syntax test.

3. Type a message for applications to display when the constraint is violated in the **Error message when constraint is violated:** text box.
4. Check the **Enforce Constraint** checkbox to instruct applications to enforce this constraint or leave it unchecked to not enforce the constraint.
5. Click **OK** to save your changes.

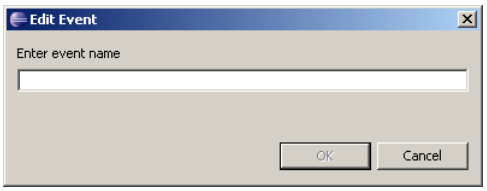
Adding, deleting or modifying events

Events are specific actions on objects. You can only create and modify application events, not system events.

To create a new event:

1. Click **New** in the **Events** section of the **Type Overview** tab in the type editor.

   The **Edit Event** dialog appears.



2. Enter a name for the event, then click **OK**. The event appears in the event table.

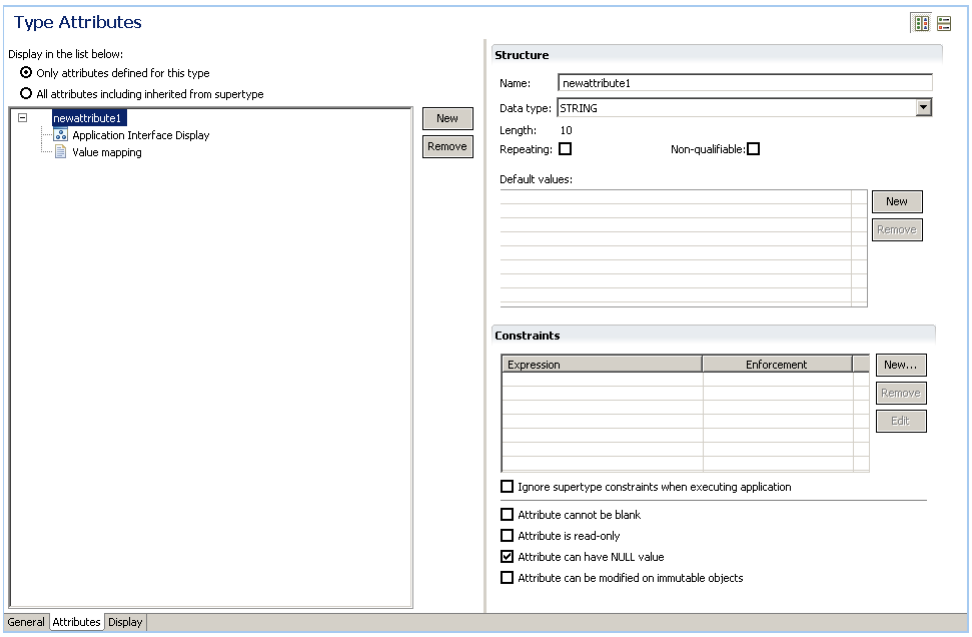
Adding type attributes

Type attributes are configures in the **Attributes** tab of the type editor. A type attribute is a property that applies to all objects of that type. When an object is created, its attributes are set to values that describe that particular instance of the object type.

To create an attribute:

1. Click the **Attributes** tab in the type editor to display the **Attributes** view.
2. Click **New** to create a new attribute entry, then click the + sign to display the configuration options.

   The **Type Attributes** view expands and appears the **Structure** and **Constraints** section.

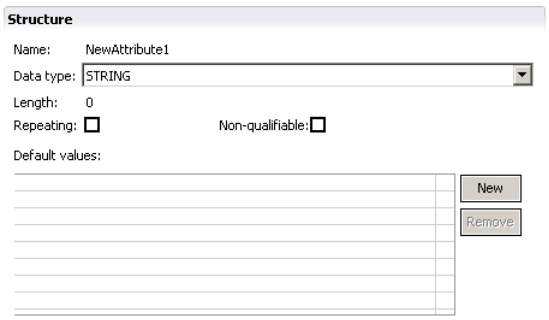


3. Configure the attribute structure, as described in Configuring the attribute structure, page 179.
4. Configure the attribute constraints, as described in Configuring attribute constraints, page 180.
5. Click the UI icon in the **Attribute** pane to display the attribute's UI configuration options.
6. Configure the attribute's UI options, as described in Configuring the type attribute UI, page 182.
7. Click the Value mapping icon to display the attribute's value mapping options.
8. Configure the attribute conditions, as described in Configuring conditional attribute values, page 184.
9. Configure the attribute value mapping, as described in Configuring attribute value mapping, page 185.

Configuring the attribute structure

The attribute structure is configured in the **Structure** section of the **Type Attributes** view (Figure 8, page 179).

Figure 8. Structure section in Type Attributes view



Enter the attribute structure properties, as described in Table 55, page 180.

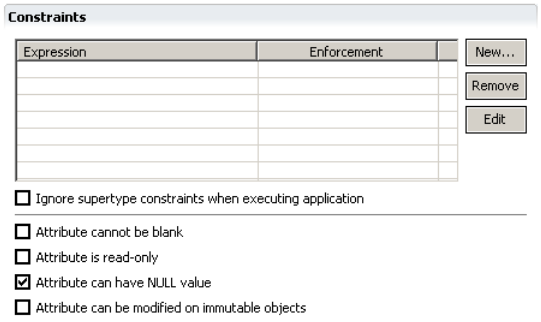
Table 55. Attribute structure properties

Property	Description
Name	A string specifying the name of the new attribute. The attribute name must use all lowercase letters, cannot begin with dm_, a_, i_, r_, a numeral, space, or single quote, and cannot be named select, from, or where.
Data type	The data type of the new attribute. Select one of the following data types from the drop-down list: • **BOOLEAN** • **INTEGER** • **STRING** • **ID** • **TIME** • **DOUBLE** • **UNDEFINED**
Length	This parameter only applies to attributes that use the **STRING** data type. Enter the number of characters for this attribute. The maximum number of characters that you can assign to this attribute depends on the database where you are installing the application.
Repeating	Specifies whether this attribute can have more than one value. Select the checkbox to allow more than one value for this attribute.
Non-qualified	Specifies whether this attribute is qualifiable or non-qualifiable. The properties and values of a non-qualifiable attribute are stored in a serialized format and do not have their own columns in the underlying database tables that represent the object types for which they are defined. Consequently, non-qualifiable attributes cannot be used in queries because they are not exposed in the database.
Default values	Lets you specify one default value for a single-value attribute or multiple default values for a repeating attribute. Click **New** to enter a default value.

Configuring attribute constraints

Attribute constraints are configured in the **Constraint** section of the **Type Attributes** view (Figure 9, page 181). Constraints are internal consistency requirements in the form of Docbasic expressions that relate the types attribute values to one another or to constant values.

Figure 9. Attribute constraints



Enter or specify the attribute constraint properties, as described in Table 56, page 181.

Table 56. Attribute constraint properties

Property	Description
Expression	The Docbasic expression defining the constraint. Click **New** to create an expression. For more information about creating or modifying an expression, see Configuring constraint expressions, page 177.
Enforcement	Specifies whether applications should enforce this constraint or not. Click the table cell in the Enforcement column to enable or disable constraint enforcement for the associated expression.The enforcement field can have two values, as follows: • **disabled**: The constraint is disabled • **ApplicationEnforced**: The constraint is enforced by the applications that use this type.
Ignore supertype constraints when executing application	Select this option to specify that applications should ignore supertype constraints for attributes of this type.
Attribute cannot be blank	Select this option to specify that the attribute must have a value. End users must enter a value for this attribute when the application executes.
Attribute is read-only	Select this option to specify that the attribute is read-only. End users cannot change the value of the attribute when the application executes.
Attribute can have NULL value	Select this option to specify that the attribute does not need to have a value assigned. End users do not need to enter a value for this attribute when the application executes.
Attribute can be modified on immutable objects	Select this option if you want users to be able to modify the attribute even though the object itself is immutable (unchangeable).

Configuring the type attribute UI

The attribute UI specifies how the attribute is displayed in client applications and is configured in the **UI** view. To open the **UI** view, click the UI icon in the attribute directory tree.

The **UI** view appears (Figure 10, page 182).

Figure 10. Type attribute UI view

Enter or specify the attribute UI properties in the **General** and **Search** sections, as described in Table 57, page 182.

Table 57. Type attribute UI properties

Property	Description
General	
Label	The name that is displayed for this attribute in the client application.
Input mask	Specifies the characters and format that an end user can enter for this attribute using the client application. An input mask consists of mask characters and literals. A backslash (\) converts the character following it to a literal. The input mask can have the following values: • #—Numeric characters (0-9). • **A**—Alphanumeric characters (0-9, a-z, A-Z) • **&**—Any ASCII character • **?**—Alphabetic characters (a-z, A-Z) • **U**—Alphabetic, gets converted to uppercase

Property	Description
	• **L**—Alphabetic, gets converted to lowercase For input mask examples, see Table 58, page 183.
Category	A string that specifies a custom tab that is displayed in Desktop client. The value entered in the category field is used unless the inheritance with the parent types display configuration is broken or a display configuration is already specified for either the type or its parent.
User help	Optional description for the type that is displayed in the application.
Comment for developers	Optional comments for developers.
Search	
Is searchable	Specifies whether an attribute is searchable. Select this option to enable clients to allow users to search a repository for attribute values in objects, if the objects are derived from the attribute's type.
Available operators	Lists the operators that can be specified for an attribute search. Select one or more operators and click **Add** to move the operators to the **Allows operators** column.
Allowed operators	Specifies the search operators than a client application can use to search for an attribute value.
Default search value	The search value that clients display by default. Optional parameter. If no default search value is specified, the client displays a blank field.
Default search operator	The search operator that clients display by default. Optional parameter. If no default search operator is specified, the client displays a blank field.

Table 58, page 183 shows a mask value and an example of valid user input.

Table 58. Input mask examples

Mask Value	Description	Valid User Input
##/##/####	Specifies a date entry.	08/23/1968
##:## UU	Specifies a time entry.	2:42 PM
###-##-####	Specifies a format for entering a numerical sequence, such a US social security number.	111223333
??????????????	Specifies a sequence for entering up to 15 letters, for example a city name.	Munich

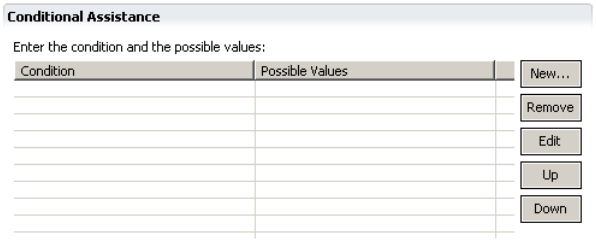
Configuring conditional attribute values

Conditional value mapping provides a list of values that a client program displays at runtime for an object attribute. A user can select a value from this list or add a new one to it. There are two kinds of conditional value mappings:

- **Default value mapping**—Values that are displayed when no value is selected, for example when a new object is created.
- **Conditional value mapping**—Values are displayed when a specified condition is satisfied, for example, when the values displayed in one attribute depend on the value of another attribute.

Conditional value mappings are configured in the **Conditional Assistance** section of the **Value Mapping** view (Figure 11, page 184).

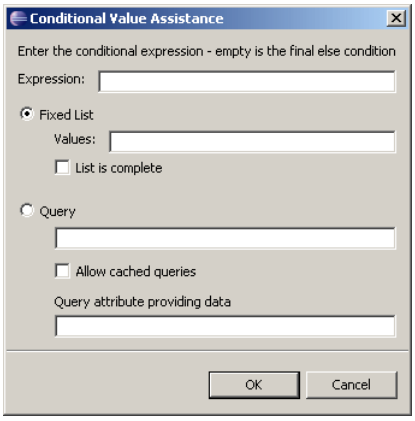
Figure 11. Conditional assistance view



To create a conditional value mapping:

1. Click **New**.

   The **Conditional Value Assistance** dialog appears.



2. Specify the conditional value properties, as described in Table 59, page 184.

Table 59. Conditional value properties

Property	Description
Expression	A Docbasic expression that specifies the condition. The Docbasic expression must resolve to true or false.

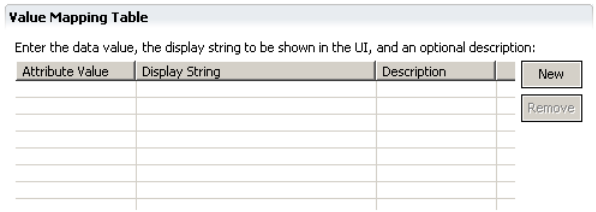
Property	Description
Fixed list	Specifies that the values are associated with the condition in form of a fixed list. Select this option if you want to use a fixed list of values and enter the respective values in the **Values** field.
Values	Specifies the values that are associated with the condition. Type the value in the **Value** field, separated by a comma.
List is complete	Specifies that the user cannot add any more values to the list.
Query	Specifies that the values that are associated with the condition are obtained by a query. Enter the query in the **Query** textbox. You can use the $value keyword to resolve attribute values at runtime.
Allow cached queries	Select this option to allow cached queries.
Query attribute providing data	Specifies the query attribute that contains the data value for the condition. Enter the name of the query attribute.

3. Click **OK** to save your changes.

Configuring attribute value mapping

Value mapping associates or maps attribute values to strings that are displayed in a client program. When a mapped string, which is displayed in the client program, is selected and the object is saved to the repository, the corresponding value is saved to the mapped string. Attribute values are mapped in the **Value Mapping Table** section of the **Value Mapping** view (Figure 12, page 185).

Figure 12. Value mapping table



To create or modify a value mapping:

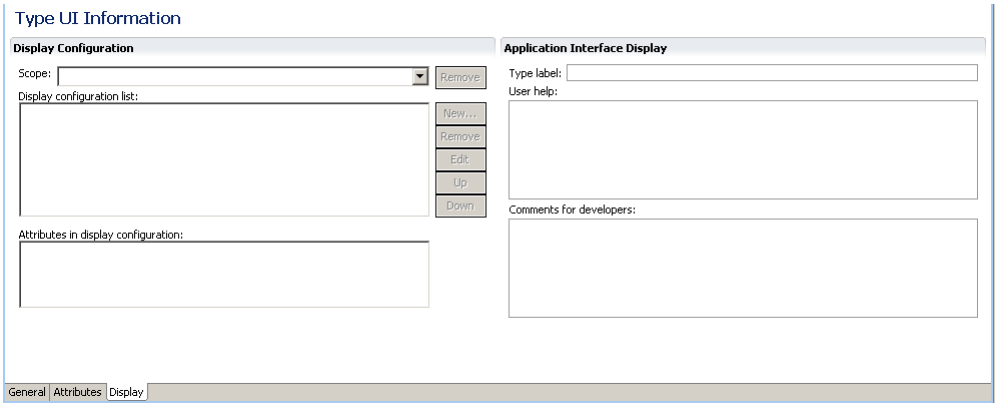
1. Click **New** to create a new dataValue/displayValue pair in the **Attribute Value** and the **Display String** columns.
2. Click the **dataValue** field in the **Attribute Value** column and enter the attribute value that is saved in the repository.
3. Click the **displayValue** field in the **Display String** column and enter the string that is displayed as the mapped value in the client application.
4. Click the field in the **Description** column to enter a string that describes the value mapping.

To remove a value mapping, select the corresponding row in the **Value Mapping Table** and click **Remove**.

Configuring the type UI information

The type UI information view lets you specify which attributes are displayed in Documentum clients and custom applications. Click the **Type UI** tab in the type editor to display the **Type UI Information** view (Figure 13, page 186).

Figure 13. Type UI information view



To configure one or more attributes to be displayed in clients:

1. Enter the type UI information as described in Table 60, page 186.

Table 60. Type UI information

Property	Description
Display Configuration	
Scope	The name of the application, in which the type attribute is displayed. The name of the application must exist in the repository.
Display configuration list	Specifies the tab on which the attribute is displayed. You can add, remove, rename, and change the position of a tab, as follows: • Click **New** to add a new tab. The Display Configuration dialog appears. For more information about adding a tab to display an attribute in a client application, see Adding a tab , page 187. • To remove a tab, select the tab name in the list, then click **Remove**. • To rename a tab, select the tab name in the list, then click **Rename**.

Property	Description
	• To change the order in which the tabs are displayed, select the tab name in the list, then click **Up** or **Down** to move the tab to the desired position.
Attributes in display configuration	Lets you modify the attributes that are displayed on a tab.
Application Interface Display	
Type label	A string that the client application displays for this type.
User help	Optional description for the type that is displayed in the application.
Comments for developers	Optional comments for developers.

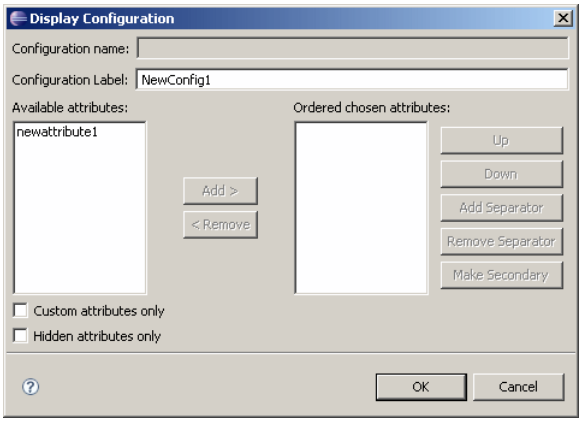
Adding a tab

Use the **Display Configuration** dialog to add a tab.

To add a tab:

1. Click **Add** in the **Display Configuration List** section of the **Type UI Information** view (Figure 13, page 186). A default tab (**NewConfig1)** for the new tab appears in the **Display Configuration List** textbox.
2. Select the default tab and then click **Edit**.

   The **Display Configuration** dialog appears.



3. Configure the tab properties, as described in Table 61, page 188.

Table 61. Tab configuration properties

Tab properties	Description
Configuration name	A string that specifies the tab name. You can either enter a new tab name or accept the default tab name. The configuration name is displayed in the client application.
Available attributes	Shows a list of the attributes that can be displayed on the tab. Select the attribute that you want to display on the tab and click **Add**. The attribute appears in the **Ordered chosen attributes** list. If the available attributes list is empty, no attributes have been configured yet. For more information about configuring attributes, see Adding type attributes, page 178.
Ordered chosen attributes	Specifies which attributes are displayed on the tab and how they are displayed. You can arrange how the attributes are displayed on the tab by selecting the attribute and using the following buttons: • **Up**—Moves the attribute up in the display order. • **Down**—Move the attribute down in the display order. • **Add Separator**—Adds a separator between the selected and the following attribute. • **Remove Separator**—Removes the separator. • **Make Secondary**—Force attributes to be displayed on a secondary page, if not all attributes can fit on one tab.
Custom attributes only	Select this option to display only custom attributes in the **Available attributes** list.
Hidden attributes only	Select this option to display only hidden attributes in the **Available attributes** list.

4. Click **OK** to save your changes.

Chapter 18

Managing XML Applications

This chapter contains the following topics:

Understanding XML applications and the application configuration file

XML applications customize and automate how XML objects and linked unparsed entities are stored in a repository. XML applications can be configured to automatically recognize different types of XML documents and set up rules to determine where they are stored, whether they should be divided into smaller chunks, how to extract and assign metadata to an object in the repository, what level of security to assign, and whether to attach a document lifecycle.

You cannot create a new XML application using Composer, but you can import an existing XML application and modify the associated XML application configuration file using the Composer XML configuration file editor. The XML application configuration file defines the rules for processing XML documents during content transfer into and out of a repository.

Viewing or modifying an XML application configuration file

Composer lets you import an existing XML application into a project. You can then modify the XML application configuration file using the **XML Configuration File** editor. **The XML Configuration File** editor provides lists of rules organized into types and allows you to base rules on elements in your XML document. There are four different types of rules:

- **XML content rule**

  The XML content rule is the most frequently used rule and applies to parsed XML content. The main function of an XML Content Rule is usually to chunk XML content in the document, but it can have other primary purposes, such as assigning metadata to an XML document that is not chunked.

- **Link rule**

A link rule uses links in the XML document to external files or references to NDATA entities to locate and handle unparsed entities, such as graphics files. The Link Rule works like an XML Content Rule in terms of assigning an object location, metadata values, permissions, and so on. You can also specify whether the linked file should be treated as a child or a peer of the XML virtual document, and whether this is a permanent link.

- **Non XML content rule**

  A non-XML content rule specifies how the server handles non-XML content contained in the XML document, such as base64-encoded data. A non-XML content rule works like XML content rules and link rules in terms of assigning an object location, metadata values, or permissions. In addition you can specify the file format of the repository object that contains the decoded data.

- **Element rule**

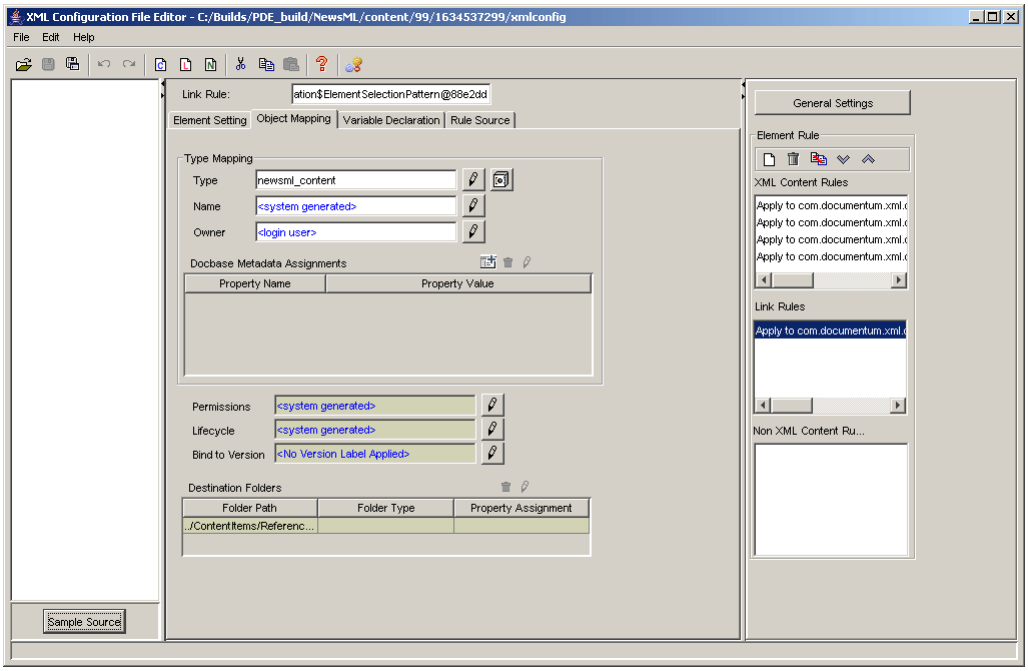
  An element rule or entity rule instructs the XML application to preserve all external parsed (XML) entities as separate components of the XML virtual document when imported or checked in, and to maintain their status as external entities when the main XML document is exported or checked out.

For more information about XML applications and XML application configuration files, see the *Documentum XML Applications Development Guide*.

To view or modify an XML application configuration file:

1. In your project, expand the **XML Applications** folder.
2. Open the **.xmlapplications** file you want to modify in one of the following ways:
   - Double-click the **.xmlapplications** file.
   - Right-click the **.xmlapplications** file and select **Open** from the menu.

   The **XML Configuration File** editor appears.



3. Modify the XML configuration file as desired, then save your changes.

Note: If you added or deleted references to repository objects in the XML application configuration file, you also need to make the same modifications in your project.

Chapter 19

Building and Installing a Project

This chapter contains the following topics:

Understanding the build and installation process

Composer projects must be built and installed in a repository in order to run. During the build process, Composer generates an executable version of the project. This executable version of a Composer project is called a Documentum Archive (DAR) and consists of one single file that contains the executable binary files of the project. There are two ways to install a Composer project:

- Composer user interface

  The Composer user interface lets you install a Documentum project in a repository. As part of the installation process, you configure certain installation parameters that apply to the whole project and to individual artifacts, as described in Configuring the project installation options, page 194 and Configuring artifact install options, page 197.

  You also have the option to just build the application and generate a DAR archive file (.dar), as described in Generating a DAR file, page 199

- Ant tasks and headless Composer

  Ant tasks and headless Composer let you build a project, generate a DAR file, and install the DAR file into a repository using a command line interface. For more information about using Ant tasks and DAR files, see Chapter 20, Managing Projects and DAR Files Using Ant tasks and Headless Composer.

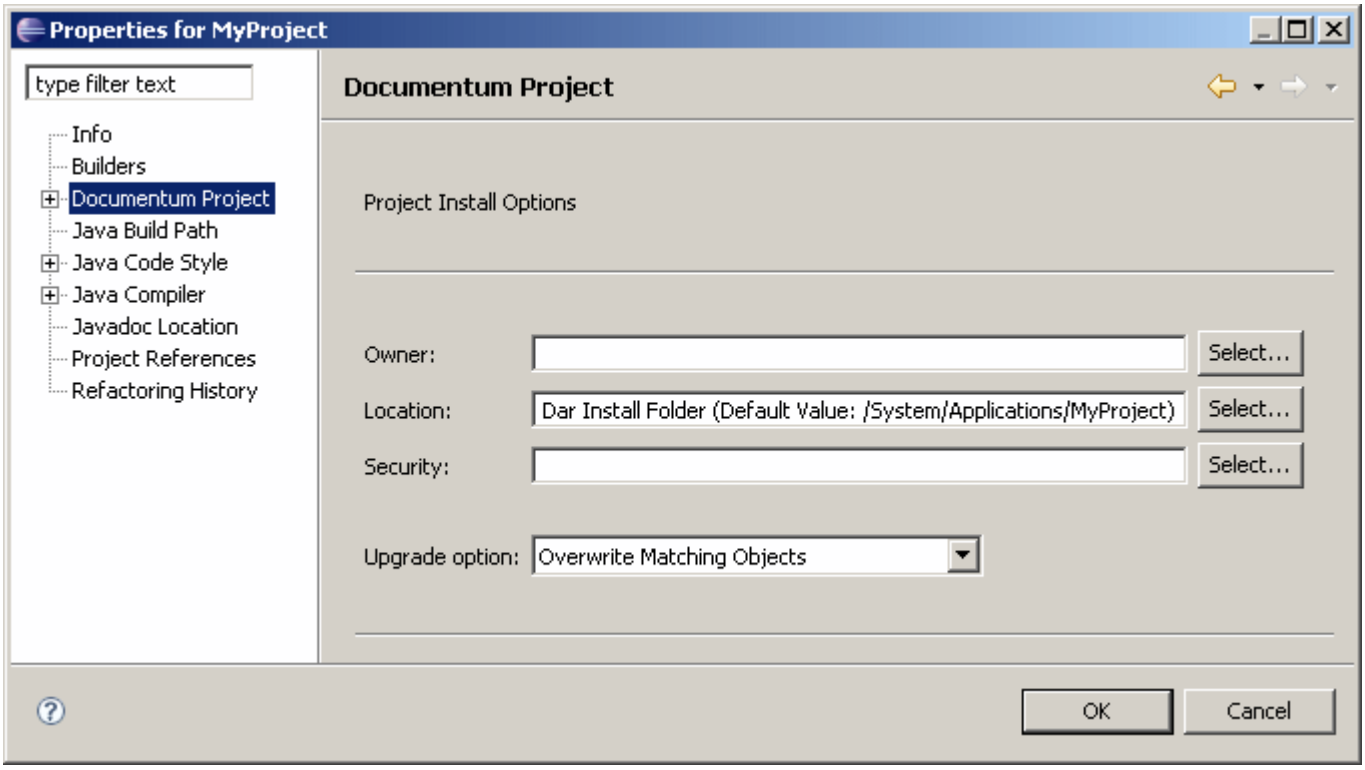
Configuring the project installation options

The project installation options let you set installation parameters that apply to the entire project, such as DFS module options, pre- and post-installation procedures and upgrade options.

To configure project installation options:

1. Open the project **Properties** dialog using one of two options, as follows:
   - In the Composer main menu select **Project** > **Properties**
   - Right-click the name of the project you want to install and select **Properties** from the drop-down menu.

   The **Properties** dialog appears.



2. Specify the installation options for the project, as described in Table 62, page 194.

Table 62. Project installation options

Install Option	Description
Owner	Specifies the owner installation parameter for installing this project. The owner must be a valid user in the repository where the project is installed. Click **Select ...** to select an owner from the listbox. For more information about adding a new owner installation parameter, see Adding an owner installation parameter, page 195.

Install Option	Description
Location	Specifies the location installation parameter for installing this project. Click **Select ...** to select a location from the listbox or accept the default value.
Security	Specifies the permission set (ACL) parameter for installing this project. Click **Select ...** to select a location from the listbox.
Upgrade option	Specifies the upgrade option used when installing this project. There are three upgrade options, as follows: • **Overwrite Matching Objects** — Overwrites all objects in the repository have matching objects in the project. If there are new objects in the project, they are installed as new objects in the repository. • **Ignore Matching Objects** — Ignores all objects in the repository that have a matching object in the project. If there are new objects in the project, they are installed as new objects in the repository. • **Create New Version Of Matching Objects** — Creates a new version of all objects in the project that have a matching object in the repository. If there are new objects in the project, they are installed as new objects in the repository.

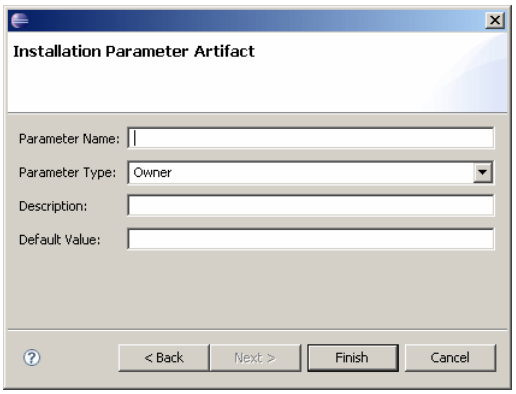
3. Double-click **Documentum Project** to expand the menu tree.
4. Select **Install Procedure** to specify a pre- and a post-installation procedure, if required, and enter the procedure names in the associated Pre-installation procedure and Post-installation procedure fields.
5. Click **OK** to save your project installation options.

Adding an owner installation parameter

You can add an owner installation parameter using the **Owner Installation Parameter Artifact** dialog.

To add an owner installation parameter:

1. Click **Select ...** next to the **Owner** field in the Project Install Options wizard.
   The **Owner Installation Parameter Artifact** dialog appears.
2. Click **New**.
   The **New Documentum Artifact – Name and Location** dialog appears.
3. Accept the default folder and artifact name by clicking **Next.**
   The **Installation Parameter Artifact** dialog appears.



4. Enter the parameter name, type, optional description, and default value, as follows:

Table 63. Owner installation parameters

Parameter	Description
Parameter name	A string specifying the name of the owner installation parameter.
Parameter type	A string specifying the type of the owner installation parameter. The type is set to **Owner** by default and cannot be changed.
Description	An optional description of the owner installation parameter.
Default value	An optional default value for the owner installation parameter. If you specify a default value for the owner installation parameter, the owner must be a valid user in the repository where the project is installed.

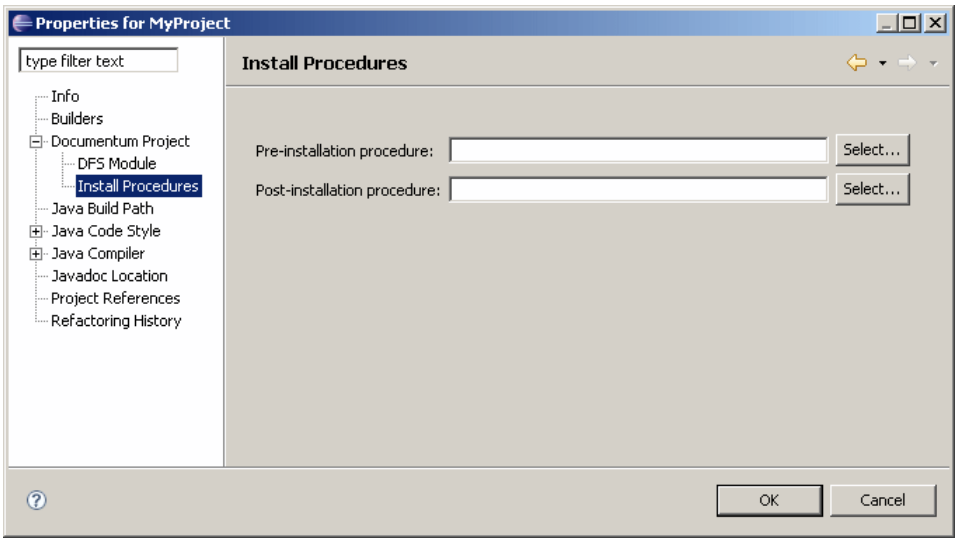
5. Click **Finish**.

Configuring pre- and post-installation procedures

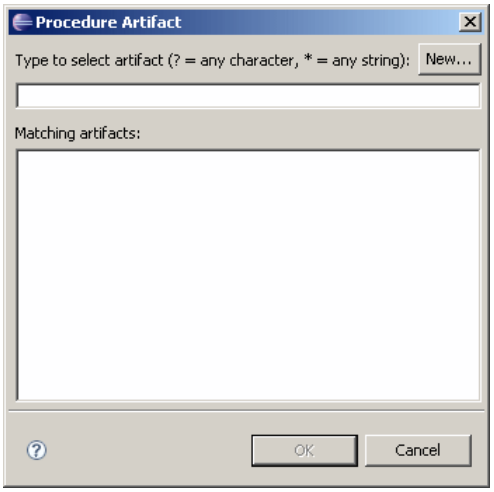
You can configure pre- and post-installation procedures for a project in the **Install Procedures** dialog.

To configure pre- and post-installation procedures:

1. Right-click the project and select Properties from the drop-down list.
   The **Properties** dialog appears (Figure 2, page 38).
2. Expand **Documentum Project** and select **Install Procedures**.
   The **Install Procedures** dialog appears.



3. Click the **Select** button next to the **Pre-installation procedure** or **Post-installation procedure** field.
   The **Procedure Artifact** dialog appears.



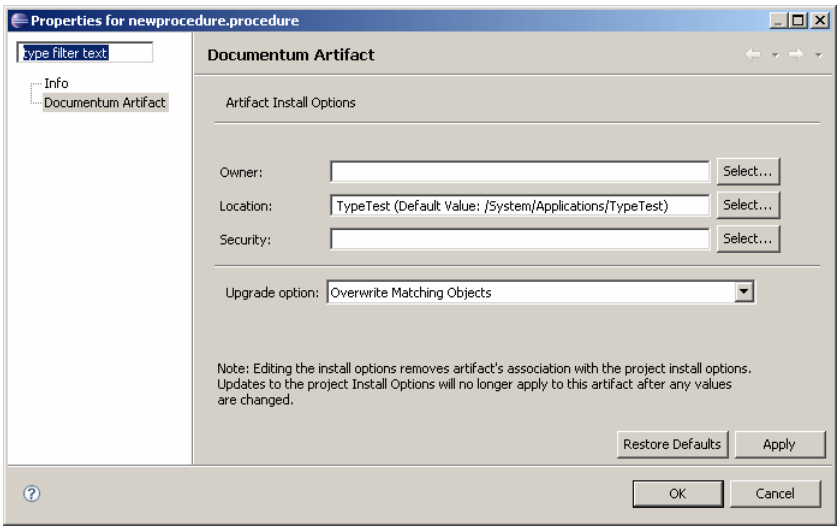
4. Select a procedure from the **Matching Artifacts** list or click **New** to create a new procedure.
5. Click OK.

Configuring artifact install options

Composer lets you configure installation options not only at a project level, but also for each individual artifact. The installation option for an artifact overrides the project installation option.

To configure install options for an individual artifact:

1. In the **Documentum Navigator** view, right-click on the artifact for which you want to configure installation options, then select **Properties** from the drop-down list.
   The **Documentum Artifact** properties dialog appears.



2. Configure the install options for the artifact, as described in Table 64, page 198.

Table 64. Artifact installation options

Install Option	Description
Owner	Specifies the owner installation parameter for installing this artifact. The owner must be a valid user in the repository where the artifact is installed. Click **Select ...** to select an owner from the listbox. For more information about adding a new owner installation parameter, see Adding an owner installation parameter, page 195.
Location	Specifies the location installation parameter for installing this artifact. Click **Select ...** to select a location from the listbox.
Security	Specifies the permission set (ACL) parameter for installing this artifact. Click **Select ...** to select a location from the listbox.
Upgrade option	Specifies the upgrade option used when installing this artifact. There are three upgrade options, as follows: • **Overwrite Matching Objects**—Overwrites all objects in the repository have matching objects in the project. If there are new objects in the project, they are installed as new objects in the repository. • **Ignore Matching Objects**—Ignores all objects in the repository that have a matching object in the project. If there are new objects in the project, they are installed as new objects in the repository. • **Create New Version Of Matching Objects**—Creates a new version of all objects in the project that have a matching object in the repository. If there are new objects in the project, they are installed as new objects in the repository. **Note:** If you are configuring the installation options for a smart container artifact, be sure to set the upgrade option to **Create New**

Install Option	Description
	Version Of Matching Objects. Do not select **Overwrite Matching Objects** for smart container artifacts because overwriting smart container objects invalidates the model-instance association for existing instances.

3. Click **OK** to save your changes.

Generating a DAR file

A DAR file is the executable version of a project that gets installed in a Documentum repository. A DAR file contains only the binary files of a project but not the source files, so you cannot convert a DAR file into a Composer project. You can install a DAR file with the DAR Installer or headless Composer.

If you have the **Project > Build Automatically** option turned on, you can obtain the **<project>.dar** file from the **...\<workspace>\<project>\bin-dar** directory of the Composer workspace. If you have the **Project > Build Automatically** option turned off, complete the following steps:

To generate a DAR file:

1. Right-click the project you want to build and select **Build Project** from the drop-down list.

   Composer builds the project and generates a **<project>.dar** file in the **...\<workspace>\<project>\bin-dar** directory, where <workspace> is the name of your workspace and <project> is the name of your project.

Installing a project

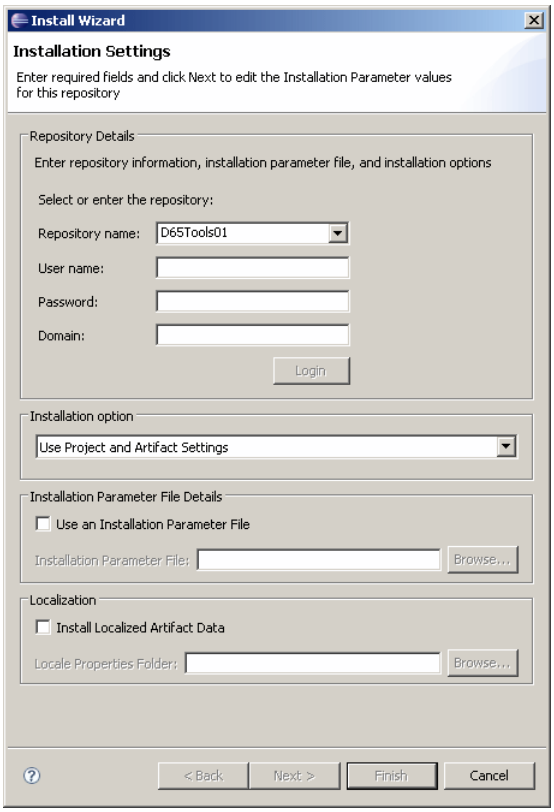
After you have created your project and you are ready to deploy the application, you need to build and install the application in a repository. If you are using a source control system, you need to check out the project from source control before you build and deploy the application.

Note: If you have projects that reference each other, be sure to install the projects in the correct order. For example, if project B references artifacts in project A, then project A must be installed first.

To install a project:

1. In the **Documentum Project Navigator** view, right-click the project you want to install and select **Install Documentum Project ...** from the drop-down menu. Composer automatically builds the project in the background. Any errors during the build process are displayed in the **Error** view.

   The **Installation Settings** dialog appears.



2. Enter the installation information, as described in Table 65, page 200.

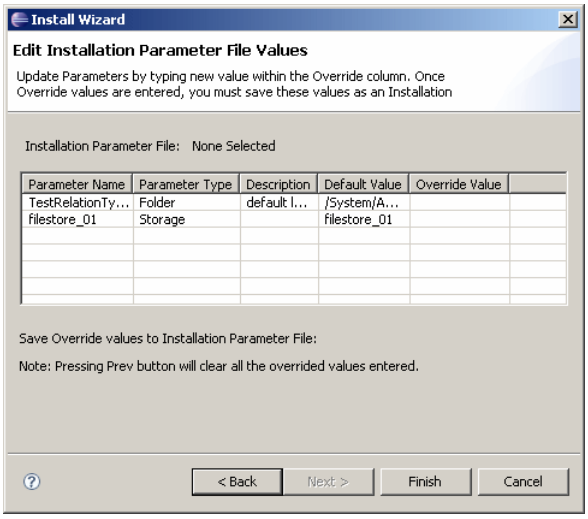
Table 65. Install parameter information

Install Parameter	Description
Repository	The name of the installation repository. Mandatory parameter. Type the repository name or select a repository from the drop-down list. You must have SUPERUSER privileges to access the repository.
User name	The login user name for the repository.
Password	The login password for the repository.
Domain	The domain name of the repository. The domain name only needs to be specified if the repository resides in a different domain than the client from which the repository is accessed.

Install Parameter	Description
Install options	Specifies how the project is installed in the repository. There are three install options, as follows: • **Use Project and Artifact Settings**—Installs the project according to the options that are configured using the Project Install Option dialog in the project properties. • **Overwrite**—If the project already exists in the repository, all objects are overwritten when a modified version of the project is installed. • **Version**—If the project already exists in the repository, objects that are versionable are versioned when a modified version of the project is installed. Objects that are not versionable are overwritten.
Use an Installation Parameter File	Select this option if you want to use an installation parameter file. Optional parameter. Click **Browse** and select the installation parameter file. If you are installing this project for the first time, and you want to use an input parameter file, you need to create the input parameter file first, as described in Creating an installation parameter file, page 202.
Install Localized Artifact Data	Select this option if you want to localize your project. Optional parameter. Click **Browse** and select the Locale Properties Folder containing the localized files. For more information about localizing a project, see Localizing a project's types, page 38.

3. Click **Next**.

   The **Edit Installation Parameter File Values** dialog appears.



   The installation parameter table lists the name, type, and default value for each installation parameter.

Note: You can change the default value for each installation parameter and save the new value to an input parameter file. For more information about creating an input parameter file, see Creating an installation parameter file, page 202.

4. Click **Finish** to install the project in the repository.

   Note: If you install a project in the repository, then edit the project in Composer and remove one or more objects and re-install the project, the old objects remain in the repository. Composer does not delete objects from the repository during installation, even if the objects were removed from the project itself.

Creating an installation parameter file

If you want to change the default installation values for one or more installation parameters, you need to create an installation parameter file. The installation parameter file contains the name, type, and default value for each installation parameter in the project.

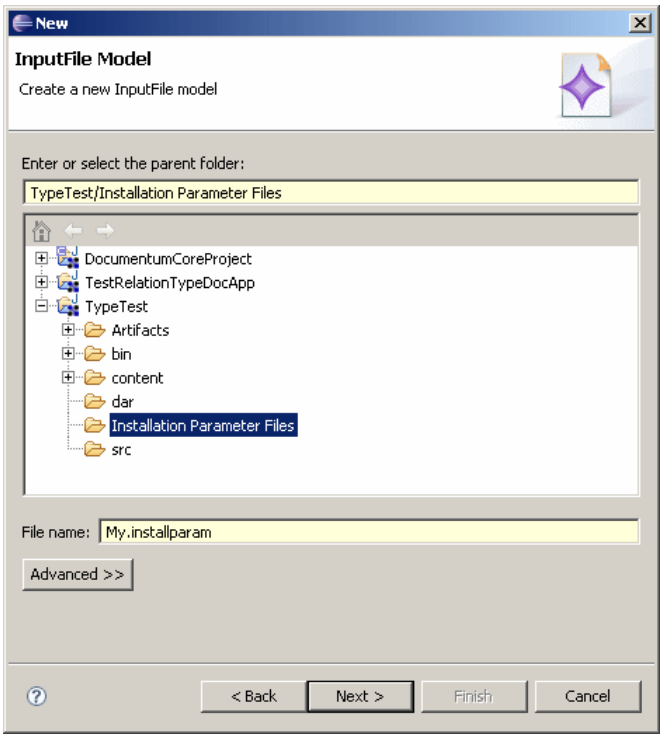
To create an installation parameter file:

1. Open the **Installation Parameter File** wizard in one of the following ways:
   - From the Composer menu, select **File > New > Other**.
   - In your Composer project, right-click **Installation Parameter Files**. Select **New > Other** from the drop-down menu.

   The **Select a wizard** dialog appears.

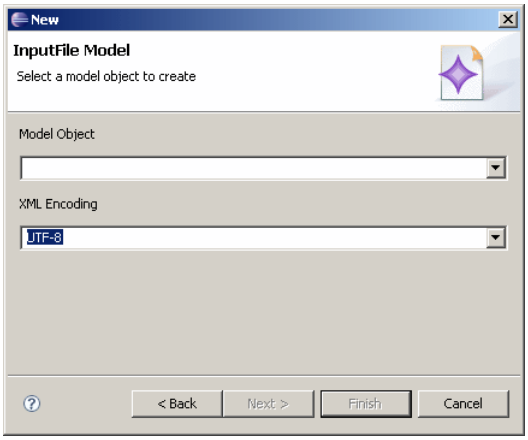
2. Double-click **Installation Parameter File** to expand it, select **Installation Parameter File New Wizard**, then click **Next**.

   The **Input File** dialog appears.



3. Select a parent folder for your installation parameter file and enter a file name in the **File name:** field, then click **Next**.

   The **Input File Model** dialog appears.



4. Select **Installation Parameter File** from the **Model Object** drop-down list, and **UTF-8** for XML encoding, then click **Finish**.

Composer creates an input parameter file and displays the file in the **Resource Set** view.

Chapter 20

Managing Projects and DAR Files Using Ant tasks and Headless Composer

This chapter contains the following topics:

Headless Composer

Composer is shipped in two packages, the UI-based version with wizards, dialogs, and editors, and a non-UI command line version that includes a set of Ant tasks for each of the common Composer functions, such as import, build, and install. The non-UI package of Composer is also referred to as headless Composer. Table 66, page 205 describes the differences between the two Composer packages. The Ant tasks let you integrate building Documentum projects and installing of DAR files into standard Ant build scripts.

Note: Because these Ant tasks leverage Composer and Eclipse infrastructure, any build scripts including these tasks must be executed through the Eclipse AntRunner. For more information on Ant, see **http://ant.apache.org**.

Table 66. UI-based and Headless Composer Comparison

Features/Functionality	UI-based Composer	Headless Composer
Create new project and artifacts	Yes	No

Features/Functionality	UI-based Composer	Headless Composer
Import DocApps from repository	Yes	No
Import DocApp archives	Yes	No
Import project from local directory	Yes	Yes
Import Artifact from repository	Yes	No
Build project	Yes	Yes
Install project	Yes	Yes
Import DAR	No	No
Install DAR file	No	Yes
	The Composer UI lets you install the project, a process that includes automatically generating and installing a DAR file "behind the scenes". However, there is no separate "Install DAR File" option in the Composer UI.	Headless Composer lets you install a DAR file using the **emc.install** Ant task.

Installing headless Composer

Headless Composer is distributed in a different .Zip file than the UI-based Composer package. The headless Composer .Zip file has the format **DCTM_Headless_Composer_<platform>_<version>.zip**.

To install headless Composer:

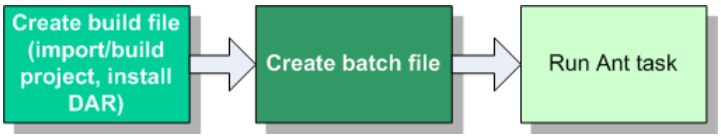
1. Extract the headless Composer .Zip file to a directory of your choice on your local machine. The directory name must not contain any spaces.
2. Configure the connection broker, as follows:
   a. Change to the **..\ComposerHeadless\plugins** directory.
   b. Double-click the **com.emc.ide.external.dfc_1.00** folder.
   c. Double-click the **documentum.config** folder.
   d. Open the **dfc.properties** file with a text editor, such as Notepad. Add the DFC and connection broker information, similar to the following:

      ```
      dfc.docbroker.host[0]=[DocBroker IP address or host name]
      ```

   e. Save your changes.

After you have installed headless Composer, you need to create your build and batch files to run Ant tasks. Figure 14, page 207 describes a typical workflow when using headless Composer.

Figure 14. Headless Composer Workflow



In your build scripts and batch file you should:

- Create a clean build workspace.
- Create a clean install workspace.
- Copy the projects into the build workspace.
- Import projects into the clean build workspace by calling the **emc.importProject** Ant task.
- Build projects and create DAR by calling **emc.build** and **emc.dar** Ant tasks.
- Install the DAR using the previous build process by calling the **emc.install** Ant task.

Headless Composer supports several Ant tasks, as described in Table 67, page 207.

Table 67. Ant tasks

Ant task	Description
emc.importProject	Imports the specified project.
emc.build	Builds a Composer project.
emc.dar	Generates a DAR file from the specified project.
emc.install	Installs a DAR file into a repository.
emc.importContent	Imports or updates a content file in a specified artifact and project.

Importing a project

Use the **emc.importProject** command to import a Documentum project. This Ant task also automatically loads the Documentum core project if it does not exist yet. The project needs to be located in the headless Composer workspace or in the Composer workspace into which the project is imported. The import process requires a batch file, as described in Creating a batch file, page 214, and a build file in XML format that contains the import parameters and the **emc.importProject** command.

The build file should look similar to the following:

```
<emc.importProject
    dmproject="TestProject"
    failonerror="true"/>
```

Table 68. emc.importProject parameters

Parameter	Required	Description
dmproject	Yes	The name of the project to import.
failonerror	No	A flag specifying whether the Ant task should fail if any import errors occur. By default, failonerror is set to false, meaning the Ant task does not fail in case of an import error.

To import a project:

1. Retrieve the project from the workspace or a source control repository and store the project in a directory local to the import script.
2. Create the build file, as described in Creating a build file, page 213.
3. Create the batch file, as described in Creating a batch file, page 214.
4. Open the command line editor on your local machine.
5. Run the Ant task, as follows:

```
C:\><batch file name>.bat <install file name>.xml
```

Creating a project

Use the **emc.createArtifactProject** target to create a Composer project in the workspace. Items in brackets are optional. Do not include the brackets when creating the targets in your build file.

```
<emc.createArtifactProject name="project_name" [overwrite="true|false"]>
   [<projectReferences [failOnMissingReferences="true|false"]>
      <project name="reference_project_name"/>
   <projectReferences>]
   </emc.createArtifactProject>
```

For example, the following target creates a project named Test. It sets the overwrite attribute to true, which overwrites any existing project with that name (the default is false). It defines DocumentumProject as a reference project and sets the failOnMissingReferences attribute to true, which causes the creation of the project to fail if the reference project does not exist (default is false).

```
<emc.createArtifactProject name="Test" overwrite="true">
   <projectReferences failOnMissingReferences="true">
      <project name="DocumentumProject"/>
   </projectReferences>
</emc.createArtifactProject>
```

Importing artifacts into a project

Use the **emc.importArtifacts** target to create a import artifacts into a specified project. Items in italics are optional.

```
<emc.importArtifacts project="project_name" docbase="repo_name"
username="username" password="password" [domain="xyz"] >
   [<objectIdentities>
      [<id value="object_id"/>]
```

```
      [<path value="object_path"/>]
      [<qualification value="dql_qualifer"/>]
   </objectIdentities>]
</emc.importArtifacts>
```

For example, the following target imports artifacts from test_repository into the Test project. The artifacts to import are declared with the objectIdentities, which specifies one object by object ID, one by path (i.e. must be a sysobject), and one by DQL qualifier (typically used for importing non-sysobjects such as a type or group).

```
<emc.importArtifacts project="Test" docbase="test_repository"
username="dmadmin" password="n0lif3">
   <objectIdentities>
      <id value="08000001800737f7"/>
      <path value="/System/Applications/Collaboration Services/CreateAcl"/>
      <qualification value="dm_group where group_name = 'my_group"/>
   </objectIdentities>
</emc.importArtifacts>
```

Building a project

Use the **emc.build** command to build a Documentum project. The build process requires a batch file, as described in Creating a batch file, page 214, and a build file. The build file must be in XML format and contain the **emc.build** command and specify the build parameters. When executing an automated build script, a fresh workspace should be created for each automated build.

The build file should look similar to the following:

```
<emc.build
     dmproject="project"
     failonerror="true/false"/>
```

This example XML file builds the project. Any errors encountered building any of the project's artifacts result in the task failing.

Note: If the project you want to build has any dependencies on other projects, you must import those projects into your workspace before you start building your project.

Table 69. emc.build command parameters

Parameter	Required	Description
dmproject	yes	The name of the project that is build.
failonerror	no	A flag specifying whether the Ant task should fail if any build errors are encountered. The default is **false**, meaning that if there are errors during the build process, the build will not fail.

To build a project:

1. Retrieve the primary Composer project and any referenced Composer projects from the workspace or a source control repository and store them in a directory local to the build script.
2. Create the build file, as described in Creating a build file, page 213.
3. Create the batch file, as described in Creating a batch file, page 214.

4. Open the command line editor on your local machine.
5. Run the Ant task, as follows:

```
C:\><batch file name>.bat <build file name>.xml
```

Generating a DAR file

Use the **emc.dar** command to generate a DAR file. The emc.dar task depends on the output from the **emc.build** task and must run in the same Java process. It is recommended that you define both tasks in the same Ant script and call them within the same build to ensure proper functionality.

The process requires a batch file, as described in Creating a batch file, page 214, and a build file in XML format that contains the DAR parameters and the **emc.dar** command.

The build file should look similar to the following:

```
<emc.dar depends="buildTarget"
  dmproject="project"
  manifest="bin/dar/project|default.dardef.artifact"
  dar="c:/project.dar" />
```

The example packages the project into the .dar archive file. The **.dardef.artifact** manifest file is used to produce the archive.

Table 70. emc.dar command parameters

Parameter	Required	Description
dmproject	Yes	A string specifying the name of the project that is packaged into the .dar file.
manifest	Yes	The relative file path to the *project*.dardef.artifact or default.dardef.artifact file that is located in the **bin/dar** directory of your project. This file is usually named default.dardef.artifact unless the project was created by importing a DocApp from a repository. In this case, the file is named *project*.dardef.artifact, where *project* is the name of your project. However, an empty default.dardef.artifact file is still created in this instance.
dar	Yes	The absolute file path to the target .dar file. The .dar file must not exist yet.
depends	No, but recommended	Specify the corresponding build target that builds the DAR file's source. If you do not specify a value for this parameter, you must run the emc.build task before the emc.dar task within the same process. It is recommended you specify a value for the depends parameter for simplicity.

To generate a DAR file:

1. Retrieve the primary Composer project and the project's .dardef file from the workspace or a source control repository and store them in a directory local to the dar script.

2. Create the build file, as described in Creating a build file, page 213.
3. Create the batch file, as described in Creating a batch file, page 214.
4. Open the command line editor on your local machine.
5. Run the Ant task, as follows:

```
C:\><batch file name>.bat <build file name>.xml
```

Installing a DAR file

Use the **emc.install** command to install a project's DAR file into a repository. The install process requires a batch file, as described in Creating a batch file, page 214, and a build fie in XML format that contains the install parameters and the **emc.install** command.

The build file should look similar to the following:

```
<emc.install
  dar="C:\builds\headless\project.dar"
  docbase="repository name"
  username="user1"
  password="xxyyzz"
  domain=""/>
```

This example XML file installs the ***project*.dar** file.

Note: The build file should be saved in Unicode (UTF-8) encoding in case the file path contains any I18N characters. The installation of the DAR file fails if the file path contains any other characters than Unicode.

Table 71. emc.install command parameters

Parameter	Required	Description
dar	Yes	The absolute file path to the .dar file being installed . The file path must contain only Unicode (UTF-8) characters or the installation will fail.
inputfile	No	The absolute file path to the install-based parameter file.
localesFolder	No	The absolute file path to localized .properties files. If you want to make your application available in other languages, you need to localize the project data, for example labels, tabs, and descriptions. For more information about localizing a project, see Localizing a project's types, page 38.
docbase	Yes	The name of the repository into which the .dar file is installed.
username	Yes	The login name for the repository.
password	Yes	The password for logging into the repository.
domain	No	The domain in which the repository resides.

If you want to enable debugging messages during the installation, call the following Ant task to set logging preferences:

```
<emc.preferences logTraceMessages="false" logDebugMessages="true />
```

To install a DAR file:

1. Retrieve the DAR file from the workspace or a source control repository and store the file in a directory local to the install script.
2. Create the build file, as described in Creating a build file, page 213.
3. Create the batch file, as described in Creating a batch file, page 214.
4. Open the command line editor on your local machine.
5. Run the Ant task, as follows:

   ```
   C:\><batch file name>.bat <build file name>.xml
   ```

Importing content

Use the **emc.importContent** command to import or update a content file in a specified artifact and project. The process requires a batch file, as described in Creating a batch file, page 214, and a build file in XML format that contains the import parameters and the **emc.importContent** command. Currently, the **emc.importContent** command only supports importing content into procedure artifacts and JAR definitions.

The build file should look similar to the following:

```
<emc.importContent
    dmproject="TestProject"
    artifactpath="inst/procedure/MyProc.procedure"
    contentfile="myproc.ebs" />
```

The following example imports the content of the **myproc.ebs** file into the **MyProc.procedure** artifact in the TestProject project, and assigns the unique identifier **myproc_content** to the content.

```
<emc.importContent
    dmproject="TestProject"
    artifactpath="inst/procedure/MyProc.procedure"
    contentfile="myproc.ebs"
    contentid="myproc_content" />
```

Table 72. emc.importContent parameters

Parameter	Required	Description
dmproject	Yes	A string specifying the name of the project containing the artifact into which the content is imported.
artifactpath	Yes	The path name of the artifact relative to the project, including the name of the artifact itself.

Parameter	Required	Description
contentfile	Yes	The absolute file path of the content file to import. The file must exist in a readable format.
contentid	No	The identifier of the content. If specified, the contentid is used to uniquely name the content within the content store of the project. If no contentid is specified, Composer generates and assigns a random name.

To import a content file:

1. Retrieve the content file from the workspace or a source control repository and store the file in a directory local to the import script.
2. Create the build file, as described in Creating a build file, page 213.
3. Create the batch file, as described in Creating a batch file, page 214.
4. Open the command line editor on your local machine.
5. Run the Ant task, as follows:

```
C:\><batch file name>.bat <install file name>.xml
```

Creating a build file

The build file calls the Ant task you want to execute. The build file must be in XML format. Table 73, page 213 describes the XML code for a build file that imports a project (“SmartContainer”) into a workspace, then builds the project, and generates a DAR file (“SmartContainer.dar”).

Table 73. Build file example

XML Code	Description
`<?xml version="1.0"?> <project name="build-smart-container" default="build"> <target name="build">`	XML declarations and tags.
`<emc.importProject dmproject= "SmartContainer" failonerror="true"/>`	Imports the **SmartContainer** project. The **failonerror** parameter specifies that the import fails if any errors are encountered during the import process.
`<emc.build dmproject="SmartContainer" failonerror="true"/>`	Builds the **SmartContainer** project. The **failonerror** parameter specifies that the build fails if any errors are encountered during the build process.

XML Code	Description
<emc.dar dmproject="SmartContainer" manifest="bin/dar/smartcontainer.dardef. artifact" dar="c:/DCTM_Composer_R.6.5.0.013/ Composer/build_workspace/SmartContainer/ SmartContainer.dar"/>	Generates the **SmartContainer.dar** file from the **SmartContainer** project. The **manifest** parameter specifies the location of the manifest file, which contains the packaging instructions for the DAR file. The **dar** parameter specifies the location where generated the .dar file is stored.
</target> </project>	XML closing tags.

Creating a batch file

The batch file is used to configure the environment variables and workspaces on your local machine, and passes the build file containing the import, build, or install instructions for your project.

To create a batch file:

1. Open a text editor, such as Wordpad.
2. Create a batch file that
   - configures the path to the headless Composer directory by setting the ECLIPSE environment variable.
   - configures at least two separate workspaces, one for importing and building a projects, and one for installing a DAR files.
   - cleans each workspace after a project has been build or a DAR file has been installed.

     You should use an empty workspace every time you import or build a project, or install a DAR file. Do not reuse workspaces.
   - invokes the settings in your **eclipse.ini** file.

     Ant tasks do not pick up the settings in the *eclipse.ini* file. This means those parameters have to be set on the command line when invoking Java.
3. Save your changes.

Batch file example

The following example batch file defines the Eclipse directory path, creates two workspaces, one for importing and building a project (BUILDWORKSPACE) and one for installing a DAR file (INSTALLWORKSPACE).

```
echo off
setlocal
```

```
set ECLIPSE="C:\<Composer package>\Composer"
set BUILDWORKSPACE="C:\<Composer package>\Composer\build_workspace"
set BUILDFILE="R:\Smart_Container\Main\buildEnv\DARBuild.xml"
set INSTALLFILE="R:\Smart_Container\Main\buildEnv\DARInstall.xml"
set DARPROJECTSDIR="R:\Smart_Container\Main\darProjects"
set INSTALLWORKSPACE="c:\<Composer package>\Composer\install_workspace"

rmdir /S /Q %BUILDWORKSPACE%
rmdir /S /Q %INSTALLWORKSPACE%
xcopy %DARPROJECTSDIR% %BUILDWORKSPACE% /E /I

java -cp %ECLIPSE%\startup.jar org.eclipse.core.launcher.Main
-data %BUILDWORKSPACE% -application org.eclipse.ant.core.antRunner
-buildfile %BUILDFILE%

java -cp %ECLIPSE%\startup.jar org.eclipse.core.launcher.Main
-data %INSTALLWORKSPACE% -application org.eclipse.ant.core.antRunner
-buildfile %INSTALLFILE%
```

Note: In your batch file you need to set the paths for the ECLIPSE and WORKSPACE environment variables according to your local environment.

Installing a DAR file with headless Composer on UNIX and Linux systems

The headless Composer distribution that is bundled with Content Server for UNIX or Linux can be used to install a DAR file to UNIX, Linux, or Windows systems. Scripts are provided for setting up the environment and installing the DAR file if you do not want to go through the trouble of creating your own deployment scripts. To install a DAR file with headless Composer on UNIX and Linux systems:

1. Login to the Content Server system as the owner of the repository that you want to install the DAR file to.
2. Ensure that your environment variables are set according to the Content Server deployment guide. Most notably, run the $DM_HOME/bin/dm_set_server_env.sh shell script to set the environment variables
3. Run the following command to install a DAR file, replacing the variables with the appropriate values for your environment:

   ```
   $JAVA_HOME/bin/java -Ddar=$PATH_TO_DAR_FILE -dlogpath=$PATH_TO_LOG_FILE
   -Ddocbase=$REPOSITORY_NAME -Duser=$REPOSITORY_SUPER_USER
   -Ddomain=$REPOSITORY_USER_DOMAIN -Dpassword=$PLAIN_TEXT_PASSWORD
   -cp $DM_HOME/install/composer/ComposerHeadless/startup.jar
   org.eclipse.core.launcher.Main -data $DM_HOME/install/composer/
   ```

```
workspace -application org.eclipse.ant.core.antRunner -buildfile
$DM_HOME/install/composer/deploy.xml
```

Chapter 21

Working with Source Control Systems

This chapter contains the following topics:

Using a source control system

If a team of developers is working on the same Composer project and other referenced projects, they often use a source control system to enable collaboration. Most source control systems offer Eclipse-based plug-ins that support an integrated development environment. These plug-ins can be used with Documentum Composer, as long as the plug-ins are compatible with Eclipse 3.2.

Checking in projects

The following guidelines are recommended when using a source control system:

- Check in the project, not the Composer workspace since workspaces are personalized
- Check in the following Documentum project directories and files:
  - Artifacts
  - content
  - dar
  - Web Services
  - src (even if this folder is empty)
  - all non-directory files at the root of the project (such as .classpath, .dmproject, .project, and .template)

Other directories, such as bin and bin-dar, and the files that are in them are derived resources and should not be checked in.

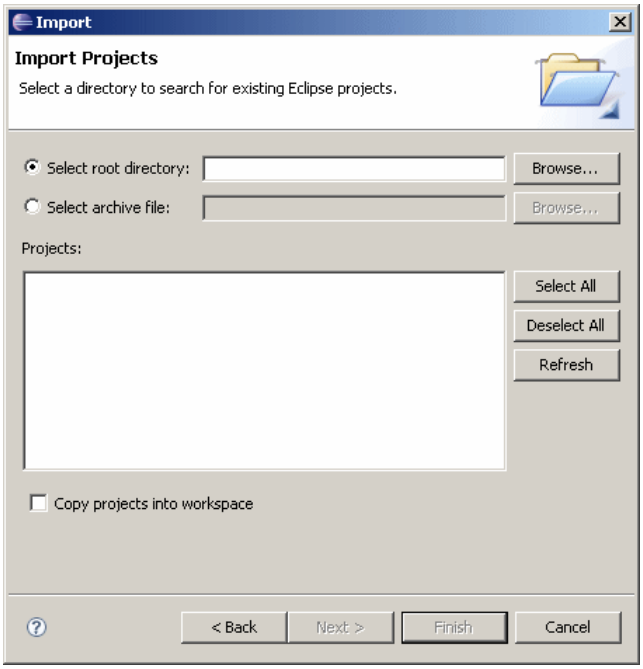
Checking out and importing projects

When working with a source control system, you need to need to import the projects into a Composer workspace after retrieving them from the source control system.

To check out and import projects into a Composer workspace:

1. Check out the project and any referenced projects from the source control system.
2. In Composer, select **File > Import > Documentum > Existing projects into workspace**, then click **Next**.

   The **Import Projects** dialog appears.



3. Select the **Select root directory** field and enter the path to the root directory where your checked out projects are located, or click **Browse** to search for the root directory. The available projects display in the **Projects** list box.

   Note: If you do not see the project(s) you want to import in the **Projects** list box, verify that you do not have a previous version of the project in your Composer workspace. You can only import projects that do not already exist in your local Composer workspace.
4. Select the projects you want to import then click **Finish**.

   Do not check the **Copy projects into workspace** option.

Building the project

Building a project is analogous to compiling code. During the build process Composer validates the artifacts and report any validation errors. The build process can be initiated in one of two ways:

- From the Composer UI
- Using headless Composer and Ant tasks

  Using ANT tasks to automatically build the DAR file is the more efficient option if the Composer projects are part of a significant build/test/deploy development cycle and the projects are on a nightly build schedule. For more information about building a DAR file using Ant tasks, see Building a project, page 209.

Glossary

ACL

An access control list (ACL) specifies the access each user has to a particular item in the repository, such as a file or folder.

artifact

A Documentum IDE resource, for example an object type, lifecycle, process, or procedure.

aspect

An aspect is a BOF module that customizes behavior or records metadata or both for an instance of an object type.

BOF

The business object framework (BOF) centralizes business logic within the Documentum framework.

data dictionary

The data dictionary stores information about object types and attributes in the repository.

DAR file

Documentum ARchive file. A DAR file is the executable, binary version of a Composer project. DAR files are typically used to distribute an application.

DocApp

A package that contains the elements that are part of an application. The term DocApp generally refers to applications prior to release 6 that were built using Documentum Application Builder (DAB) instead of Composer.

DocApp archive

A pre-version 6 mobile version of a DocApp. The archive file is a compressed (zip) file that can be stored on a local machine or network drive.

Eclipse

Eclipse is the name for the overall project supporting the construction of integrated tools for developing applications.

Eclipse platform

Eclipse Platform is the name for the core frameworks and services upon which plug-in extensions are created. It provides the runtime in which plug-ins are loaded and run.

EMF

Eclipse Modeling Framework. A modeling framework and code generation facility for building tools and other applications based on a structured data model. For more information about EMF, see Eclipse Modeling Framework.

IDE

An integrated development environment (IDE), also known as integrated design environment and integrated debugging environment, is a type of computer software that assists computer programmers in developing software.

lifecycle

A lifecycle encodes business rules for changes in the properties of an object in a repository, such as a document, as it moves through the stages of its life.

lightweight object type

Lightweight objects are part of an object model enhancement introduced to share system managed metadata among objects which only hold application specific data.

permission set

Permission sets (also known as ACLs, or access control lists) specify the access each user has to a particular item in the repository, such as a file or folder.

project

A project contains the source code and related files for building a program.

relation type

A relation type defines the relationship between two objects in a repository.

repository

The repository stores the persistent objects managed by Content Server, such as the object metadata and content files.

SBO

A service-based object (SBO) is a type of module, similar to a session bean in an Enterprise JavaBean (EJB) environment. SBOs can operate on multiple object types, retrieve objects unrelated to Documentum objects (for example, external email messages), and perform processing.

smart container

Smart containers define objects and relationships in a template that at runtime is used to instantiate instances that are based on the template.

TBO

A type-based object (TBO) is an extension of a basic Documentum Foundation Class (DFC) type and can be used to provide a new behavior for existing or new object types, or to customize low level operations to enforce data validation, referential integrity and business rules.

Web services

A Web service (also Web Service) is a software system designed to support interoperable machine-to-machine interaction over a network. Composer supports Web services by providing an EMC Documentum Foundation Services (DFS) registry plug-in.

workflow

A workflow formalizes a business process such as an insurance claims process or an engineering development process.

Workspace

A workspace is the general umbrella for managing resources in the Eclipse Platform.

Index

D

E

F

G

H

I

J

L

M

O

P

R

S

T

V

W

X