opentext™

OpenText™ Documentum™ Content Management

Composer User Guide

Create a new project and artifacts, import DocApps and artifacts from the repository, build and install the project, and install the Documentum Archive file.

EDCPC250400-UGD-EN-01

OpenText™ Documentum™ Content Management
Composer User Guide

EDCPC250400-UGD-EN-01
Rev.: 2025-Oct-20

This documentation has been created for OpenText™ Documentum™ Content Management CE 25.4.
It is also valid for subsequent software releases unless OpenText has made newer documentation available with the product, on an OpenText website, or by any other means.

Open Text Corporation

275 Frank Tompa Drive, Waterloo, Ontario, Canada, N2L 0A1

Tel: +1-519-888-7111
Toll Free Canada/USA: 1-800-499-6544 International: +800-4996-5440
Fax: +1-519-888-0677
Support: https://support.opentext.com
For more information, visit https://www.opentext.com

Table of Contents

Chapter 1

Documentum Composer

1.1 Introduction to Documentum Composer

Documentum Composer provides tools to create and customize applications for OpenText™ Documentum™ Content Management Server. These applications specify how OpenText Documentum Content Management (CM) Server handles different types of content.

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

1.1.1 About the Documentum Composer user interface

The Documentum Composer user interface provides for multiple navigations to various dialog boxes and screens. For example, Documentum Composer enables you to access the **New Alias Set** dialog to create an alias set artifact in one of the following ways:

- Right-click in the Documentum Navigator area and select **New > Alias Set**.
- From the Documentum Composer menu, select **File > New > Alias Set**.
- In your project, right-click the **Artifacts** folder, and select **New > Alias Set**.
- In your project, expand the **Artifacts** folder, right-click **Alias Set**, and select **New > Alias Set**.
- From the toolbar, click the down arrow button next to the **Create a new Documentum Artifact** icon, and then select **Alias Set** from the list.
- From the Documentum Composer menu, select **File > New > Other**, expand the **Documentum Artifact** folder, select **Alias Set**, and then click **Next**.

The various procedures in this document generally show only one navigation to a dialog box or screen. This is done intentionally.

1.1.2 Headless Composer and Documentum Composer UI

There are two Documentum Composer versions: Documentum Composer and Headless Composer. Documentum Composer is the full integrated development environment (IDE) that provides a user interface to create, build, and install Documentum Composer projects. Headless Composer is a command-line driven build tool to create, build, and install Documentum Composer projects with Ant tasks. The Ant tasks enable you to integrate the building of Documentum projects and installing of DAR files into standard Ant build scripts. Because the Ant tasks leverage Documentum Composer and Eclipse infrastructure, any build scripts that use these tasks must be executed through the Eclipse AntRunner. Apache Ant website contains more information. The following table describes the differences between the two Documentum Composer packages.

Features or Functionality	UI-based Documentum Composer	Headless Composer
Create new project	Yes	Yes
Create new artifacts	Yes	No
Import DocApps from repository	Yes	No
Import DocApp archives	Yes	No
Import project from local directory	Yes	Yes
Import artifact from repository	Yes	Yes
Build project	Yes	Yes
Install project	Yes	Yes
Install DAR file	No The Documentum Composer UI lets you install the project, a process that automatically generates and installs a DAR file "behind the scenes." However, there is no separate Install DAR File option in the Documentum Composer UI. Use the DAR Installer to install a DAR file interactively.	Yes Use the *emc.install* Ant task to install a DAR file.

1.2 Installing Documentum Composer

Documentum Composer is packaged as a compressed zip file that contains the Eclipse platform and all required plug-ins. To install Documentum Composer, extract the contents of the zip file to a directory of your choice.

Before installing Documentum Composer, ensure that you meet the following prerequisites:

- Documentum repositories
- The supported version of Java JDK

  OpenText Documentum Composer Release Notes contains the current Java JDK version update required for Documentum Composer.

To install Documentum Composer:

1. Extract the content of the `DCTM_Composer_<version>.zip` file to a directory on your local machine. The system creates a directory named Documentum Composer.
2. Set the *JAVA_HOME* environment variable on your local machine to point to your installation of Java JDK. For example, if the Java JDK is installed in `C:\Program Files\Java\jdk` directory, set the *JAVA_HOME* variable to that path.
3. Edit the `<composer_root>\plugins\com.emc.ide.external.dfc_1.0.0\documentum.config\dfc.properties` file and add the connection broker information, like the following:

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

To work with lightweight SysObjects, install the lightweight SysObject plug-in as described in "Installing the lightweight SysObject plug-in" on page 11.

1.2.1 Installing the lightweight SysObject plug-in

Documentum also offers a lightweight SysObject plug-in for Documentum Composer that must be installed separately. The lightweight SysObject plug-in is not part of the main Documentum Composer distribution and must be installed in the `<Composer_root>/plugins` directory after you install Documentum Composer.

Currently, only applications designed for OpenText Documentum Content Management (CM) High-Volume Server can make proper use of lightweight objects. High-Volume Server is an extension of Documentum CM Server that supports features implemented to solve common problems with large content stores, such as email archiving. It requires an additional license key that you specify when you install OpenText™ Documentum™ Content Management Server. The *OpenText Documentum High-Volume Server Development Guide* provides more information about lightweight object types and High-Volume Server.

To install the lightweight SysObject plug-in:

1. Download the `LightweightObject_<version>.zip` file from OpenText My Support (https://support.opentext.com).
2. Extract the plug-in to the same directory as Documentum Composer. For example, if you extracted Documentum Composer to the `C:\` directory, extract the `LightweightObject_<version>.zip` file to the `C:\` directory.
3. Change to the `<Composer_root>/plugins` directory and verify that the following files are in the directory:
   - `com.emc.ide.artifact.lwdclass_1.0.0.jar`
   - `com.emc.ide.artifact.lwdclass_ui_1.0.0.jar`

1.3 Installing other Documentum Composer plug-ins

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

Depending on how the plug-ins are packaged, extract the package to the main Documentum 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.

1.4 Installing Headless Composer

Headless Composer is distributed in a different .zip file than the UI-based Documentum Composer package.

To install Headless Composer:

1. Extract the Headless Composer zip file to a directory on your local machine. The directory name must not contain any spaces. The Headless Composer zip file has the following format: `DCTM_Headless_Composer_<platform>_<version>.zip`
2. Edit the `Composer_root>/plugins/com.emc.ide.external.dfc_1.0.0/documentum.config/dfc.properties` file and add the connection broker information. The following is an example:

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

   You must have a valid username and password for all of the repositories that you want to access and that the connection broker is aware of these repositories.

   Note: dfc.globalregistry.password and dfc.security.ssl.truststore_password can be managed by Vault. For more information about Vault, see *Documentum Server* documentation.

"Creating a Headless Composer build" on page 225 provides information on how to use Headless Composer.

1.5 Linux support in Documentum Composer

You can use Headless Composer on Linux systems to install DAR files to Documentum CM Server repositories on Linux and Windows systems. Only the Headless Composer distribution that is bundled with Documentum CM Server is supported in Linux environment.

Alternatively, you can use the DAR Installer or Headless Composer on Windows systems to install DAR files to Documentum CM Server repositories on Linux systems.

"Installing a DAR file with Headless Composer on Linux systems" on page 236 provides information on how to run Headless Composer with Ant tasks.

1.6 Configuring the connection broker

Each time you import a project or artifacts, you access a Documentum repository. The connection broker handles repository access. You can update the connection broker at any time.

To configure the connection broker:

1. Edit the `<Composer_root>\plugins\com.emc.ide.external.dfc_1.0.0\documentum.config\dfc.properties` file and add the connection broker information, like the following:

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

2. Save your changes.

1.7 Starting and configuring Documentum Composer

Documentum Composer runs on top of the Eclipse platform and uses a similar development concept. To run Documentum Composer, configure at least one workspace. The workspace is the directory where Documentum Composer stores your work. Specify the location for the workspace before using Documentum Composer.

Note: Newer versions of Documentum Composer cannot use workspaces created by an older version of Documentum Composer. Create a workspace first and then import projects from the old workspace into the new workspace.

To start Documentum Composer and configure a workspace, do the following:

1. Go to the `..\Composer` installation directory on the machine where you extracted the Documentum Composer ZIP file and double-click `composer.exe`.

2. When you start Documentum Composer for the first time, select the location of your workspace in the **Workspace Launcher** dialog box.

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

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

   The Documentum Composer workbench appears.

1.8 Configuring the Java compiler preferences

To configure the Java compiler preferences:

1. In the Documentum Composer main menu, navigate to **Window > Preferences**.

   The **Preferences** dialog appears.

2. Click the **Java** option to expand it, then click **Installed JREs**.

   The **Installed JREs** page appears. Ensure that this page mentions the supported JRE version.

3. Select **Java > Compiler** from the tree on the left and set the **Compiler compliance level** to one of the Java versions listed.

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

1.9 Installing language packs

This section describes how to install Documentum Composer language packs to enable localization. You must install the following items in the order given:

- English Documentum Composer
- Lightweight SysObject plug-in (optional)
- Documentum Composer <language name> language pack
- Eclipse <language name> language pack

Ensure that your system meets the requirements listed in the *OpenText Documentum Composer Release Notes* before installing or upgrading the software. The product documentation are available from at OpenText My Support (https://support.opentext.com).

"Installing a Documentum Composer language pack" on page 15 provides instructions to install a Documentum Composer language pack.

1.9.1 Installing a Documentum Composer language pack

This section contains instructions to install Documentum Composer and relevant language packs. The language pack installs language-specific files that consist of a plug-in containing the graphical user interface and online help that has been localized (translated) into a language other than the default language, which is US English. Currently, Documentum Composer is available only in Japanese.

To install a language pack:

1. Download the English Composer ZIP file.
2. Unzip the English Composer ZIP file to a directory on your local drive. This creates a directory named "Composer" and will be referred to as <*Composer_root*>.
3. If you are installing the Lightweight SysObject plugin, unzip the plugin to the Documentum Composer parent directory. For instance, if Documentum Composer is installed in the `<Composer_parent_directory>\Composer` directory, unzip the file to `<Composer_parent_directory>`. You must install this plug-in before you install the Eclipse <language name> language pack.
4. Extract the Composer <language name> Language pack into the `<Composer_root>\plugins` directory. You must install the Composer <language name> language pack before you install the Eclipse <language name> language pack.
5. Start Documentum Composer by running `<Composer_root>\composer.exe`.
6. Download the Japanese language pack(s) for Eclipse from the Eclipse website. The *OpenText Documentum Composer Release Notes* contains the details of supported language packs and versions.
7. Extract the Eclipse Japanese language pack in the `<Composer_root>\dropins` folder.
8. Exit Documentum Composer.
9. Use the command line to run Documentum Composer from the Documentum Composer installation root: `composer -nl <languagename>`. For example, to localize Documentum Composer in the Japanese language, run the command `composer -nl ja`.

Chapter 2

Managing Projects

2.1 Documentum Composer projects

A Documentum Composer project specifies the objects that make up an application. Therefore, create a project before you start to develop 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. "Documentum Composer artifacts" on page 24 contains a complete list of artifacts. A Documentum Composer project is marked with an  icon.

There are several ways to create a Documentum Composer project:

- Create an empty project as described in "Creating a project" on page 17.
- Import an existing project into Documentum Composer as described in "Importing a project" on page 19.
- Create a Documentum Composer project from a local 5.3 DocApp archive as described in "Converting a DocApp archive" on page 42.
- Create a Documentum Composer project from a 5.3 DocApp, as described in "Converting a DocApp to a Documentum Composer project" on page 37.

Note: Newer versions of Documentum Composer cannot use workspaces created by an older version of Documentum Composer. Create a workspace first and then import projects from the old workspace into the new workspace.

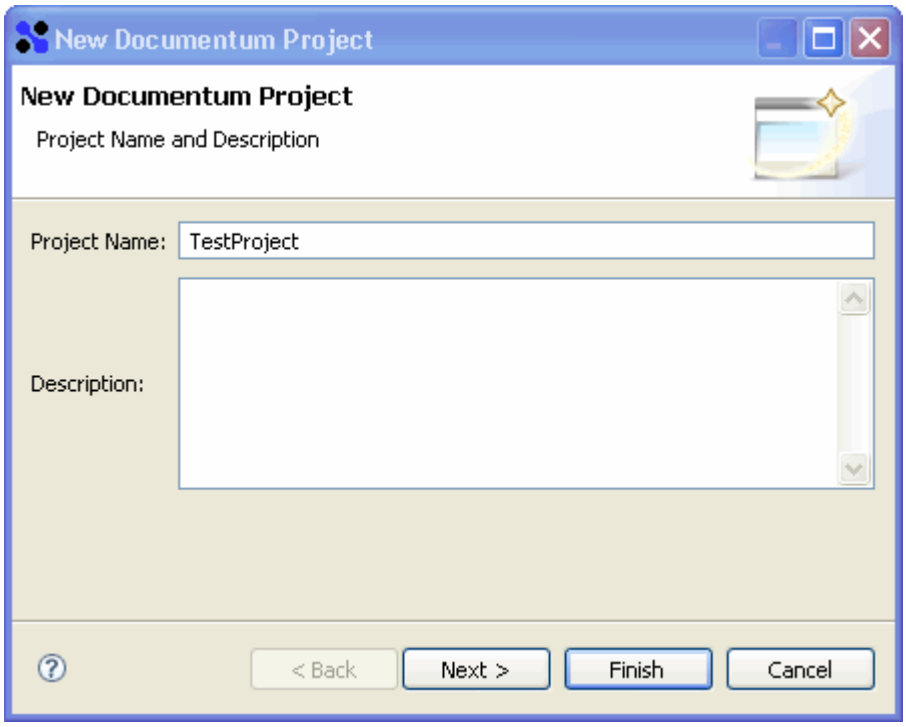
2.1.1 Creating a project

Create a project whenever you want to create an application from scratch.

To create a project:

1. Right-click in the Documentum Navigator area and select **New > Documentum Project**.

   The **New Documentum Project** dialog appears.



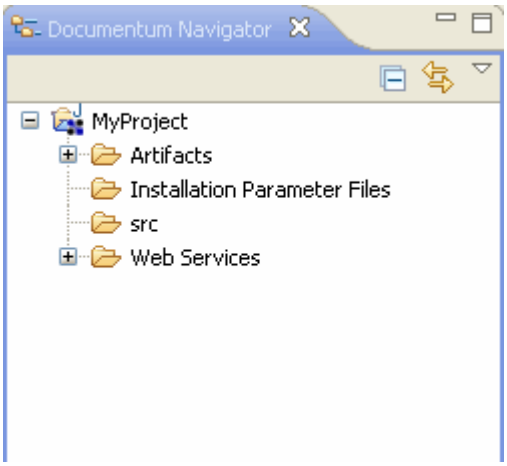
2. Type a name for your project in the **Project Name** field, type an optional description, and then click **Next**.

   The **Referenced projects** dialog box appears.

3. Select projects to designate them as reference projects and click **Finish**. "Documentum Composer reference projects" on page 20 provides more information about reference projects.

Note: If a dialog box prompts you to select the associated **Documentum Artifacts** perspective, click **Yes**.

Documentum Composer creates the project and displays it in the **Documentum Navigator** view.



By default, a project contains the following folders:

- **Artifacts**: The Artifacts folder contains subfolders for all artifacts that are available in Documentum Composer. When you create a project, these artifacts 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 project. After you add artifacts and configure installation options for the project and artifacts, the associated .installparam installation parameter files are stored in this folder.
- **src**: The src folder is used to store source files that you want to add to your project. By default, the src folder is empty when you create a 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 project.

2.1.2 Importing a project

This section describes how to import projects from a local directory. You can import existing projects from a local directory into the Documentum Composer workspace. If you use a source control system to manage your files, check out the project from the source control system before importing it into Documentum Composer. "Using a source control system" on page 237 provides more information about how to use Documentum Composer with a source control system.

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. Right-click in the Documentum Navigator area and select **Import > Existing Projects into Workspace**.

The **Import Projects** dialog box appears.

2. Select **Select root directory** and type the project directory or click **Browse** to search for the directory.

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

   Note: The **Select archive file** option is not supported in Documentum Composer 7.2.

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

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

   Note: If you use source control, do not use the **Copy projects into workspace** option.

   Documentum Composer does not support importing renditions of documents.

2.2 Documentum Composer reference projects

Documentum Composer allows you to create references between projects. This functionality is useful if you have projects that share resources such as Documentum artifacts, libraries, or Java Archive (JAR) files. You can specify reference projects when you create a 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 enable you to access Documentum functionality.

2.2.1 Documentum-supplied reference projects

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

Every project created within Documentum Composer has the DocumentumCoreProject designated as a reference project by default. The DocumentumCoreProject contains all of the artifacts that Documentum CM Server provides, so you can use or extend these artifacts out of the box. The project is read-only and cannot be modified. The project is marked with the icon and is displayed only in the **Package Explorer** view, and not the **Documentum Navigator** view. In addition to DocumentumCoreProject, the TCMReferenceProject is also assigned automatically as a reference project if your project contains any xCelerated Composition Platform (xCP) artifacts, such as TaskSpace types.

If you 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 Documentum Composer reference projects in their respective download areas on OpenText My Support (https://support.opentext.com).

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

- Your Documentum Composer project cannot contain artifacts with names that begin with *dm* because *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 Documentum Composer. However, a *dm* artifact can exist in Documentum-supplied reference projects, such as DocumentumCoreProject or TCMReferenceProject. This provides you with a mechanism to use and extend *dm* artifacts.
- You can use or extend any *dm* artifact that DocumentumCoreProject or TCMReferenceProject provides without the need to download a separate reference project. Obtain the reference projects for all other *dm* artifacts that you require.
- 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, Documentum Composer imports these artifacts from the repository automatically. If the artifacts that are imported automatically have names that begin with *dm*, the following error occurs: `Type name is invalid. Type names must not begin with 'dm'. For more information, see the 'Reference projects' section in the Documentum Composer User Guide.`

  If this error occurs, delete the newly-imported artifacts, import and designate the appropriate projects as reference projects, and import the desired artifacts again.
- If you import an artifact that indirectly references a *dm* artifact, import the project that contains the *dm* artifact and designate it as a reference project. For example, if you import a type named my_child_type that depends on a type named my_parent_type that depends on a *dm* type, then 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. If the DocApp or DocApp archive uses or extends *dm* artifacts that are not in DocumentumCoreProject, import all required Documentum-supplied reference projects into your workspace before converting the DocApp or DocApp archive. During the conversion, Documentum Composer prompts you to specify the necessary reference projects.

2.2.2 Designating projects as reference projects

There are two ways to designate projects as reference projects:

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

If you are converting a DocApp or DocApp archive, 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, designate the reference project first before importing the artifact. You can designate the reference project when creating a project or by editing an existing project.

2.2.2.1 Designating reference projects for new Documentum Composer projects

If you know beforehand that your project uses or extends *dm* artifacts that are not in DocumentumCoreProject or TCMReferenceProject, obtain the appropriate reference projects and import them into your workspace. When the import completes, designate the projects as reference projects.

Follow this procedure to convert DocApps or DocApp archives into Documentum Composer projects when those DocApps or DocApp archives use or extend a *dm* artifact that is not in DocumentumCoreProject or TCMReferenceProject.

If you are importing *dm* artifacts or artifacts that extend a *dm* artifact that is not in DocumentumCoreProject or TCMReferenceProject, follow this procedure as well.

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

1. Download the necessary reference projects from OpenText My Support (https://support.opentext.com). The reference projects are located in the Documentum download area for the product.
2. Import the ZIP file of the reference project into your Documentum Composer workspace as described in "Importing a project" on page 19. When the import is complete, the project appears in the Documentum Navigator view of Documentum Composer.
3. Create a project from scratch or from a DocApp or DocApp archive. When prompted, designate the appropriate reference projects.
   - To create a project from scratch, see "Creating a project" on page 17.
   - To create a Documentum Composer project from a 5.3 DocApp, see "Converting a DocApp to a Documentum Composer project" on page 37.
   - To create a Documentum Composer project from a local 5.3 DocApp archive, see "Converting a DocApp archive" on page 42.

2.2.2.2 Designating reference projects for existing Documentum Composer projects

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

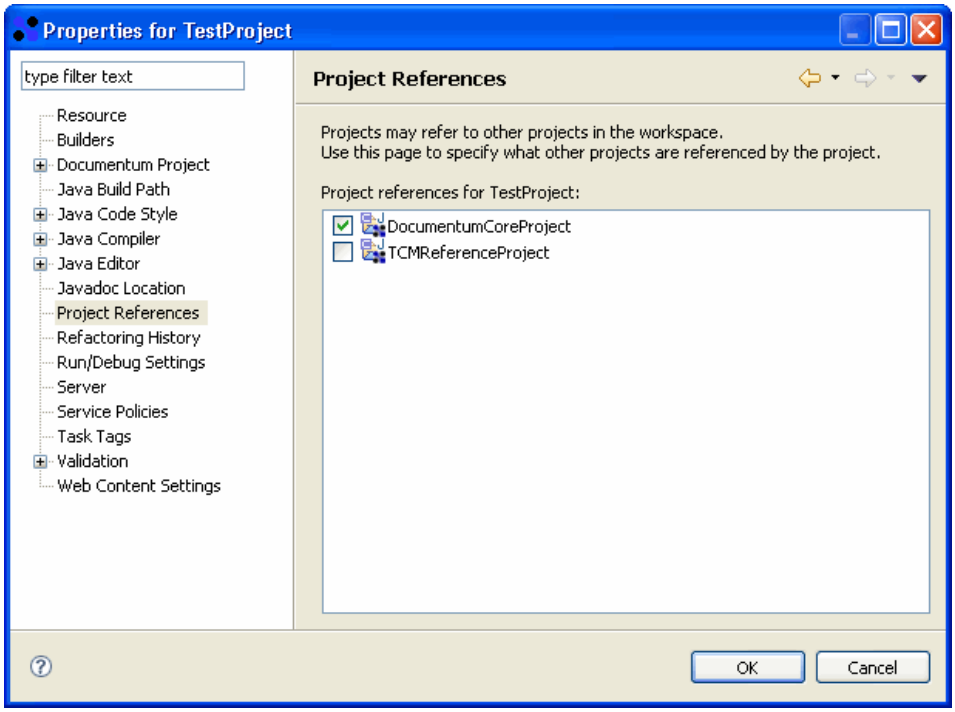
Before you can designate a project as a reference project, the project must be in your Documentum Composer workspace. If it is not in your workspace, import the project first, as described in "Importing a project" on page 19.

Note: If you created a project from a DocApp and want to reference Documentum-supplied reference projects, do not follow this procedure. 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 can encounter errors during the import process.

To create a reference to another project:

1. In the **Documentum Navigator** area, 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.

3. Select one or more projects that the project references.

4. Click **OK**.

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

2.3 Documentum Composer artifacts

Artifacts are Documentum resources, such as object types, modules, methods, permission sets, and procedures. You can create artifacts in Documentum Composer or you can import artifacts from existing repositories.

Documentum Composer offers various artifacts as shown in the following table:

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.
BAM Configuration	Configuration file used to drive Business Activity Monitor (BAM).
BAM Report	Report created by using BAM and imported into Documentum Composer so that it can be deployed to another repository. BAM Reports can be used to track key performance issues such as SLA enforcement, cycle time, and transaction revenue.
Folder	Folder object within a Documentum repository that is used as a containment structure. A folder contains Documentum objects such as documents.
Group	Defines a logical grouping of users or child groups.
Format	Contains information about a file format recognized by OpenText Documentum Content Management (CM) 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 Documentum 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 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 typically used to specify pre- and post-installation tasks.
Relation Type	Defines the relationship between two objects in a repository.
Smart Container	Defines objects and relationships in a template that is used to create instances at runtime.
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 of its subtypes.

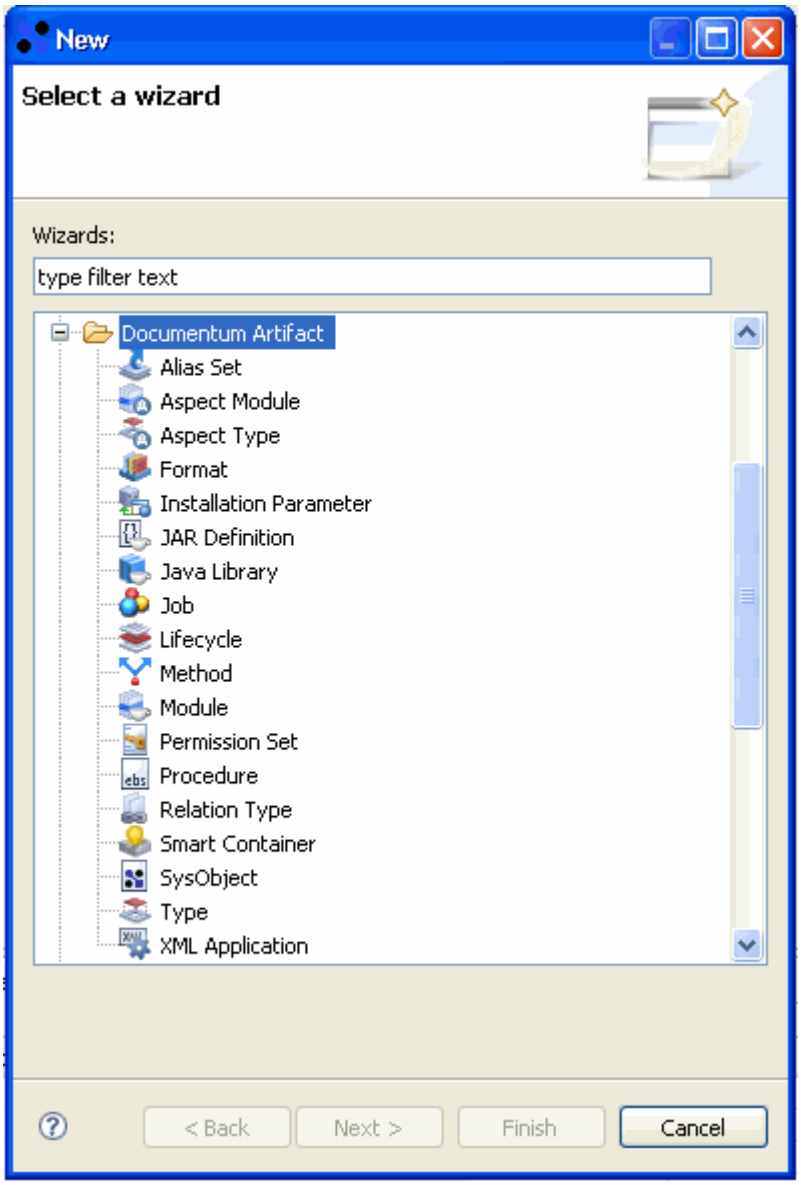
Artifact name	Description
Type	Represents a class of objects. The definition of an object type is a set of attributes. The attribute values describe individual objects of the type.
XML Application	Customizes and automates how XML content is handled in a repository.

2.3.1 Creating an artifact

Use the artifact wizard to create an artifact.

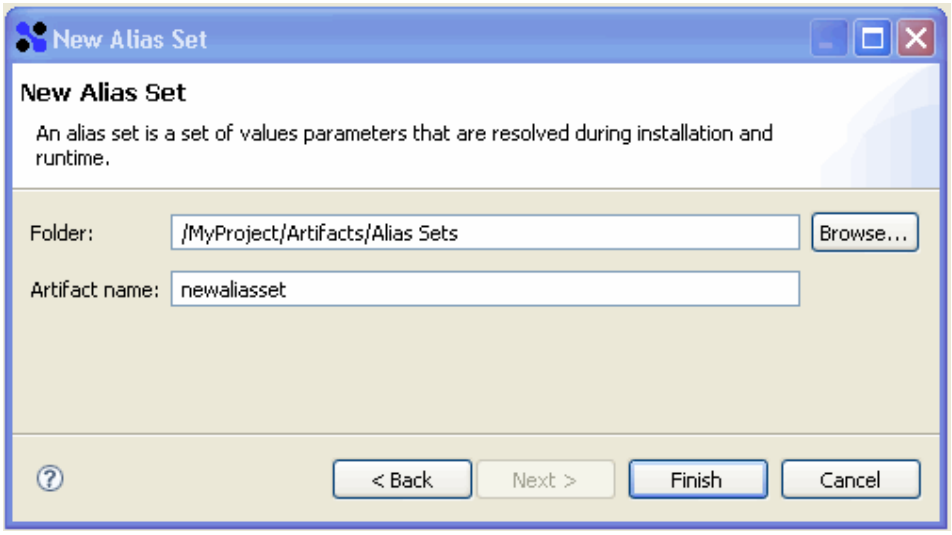
To create an artifact:

1. Right-click in the Documentum Navigator area and select **New > Other**.
   The **Select a wizard** dialog box appears.



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

   The **New [artifact type]** dialog box appears, where [artifact type] is the artifact name that you previously selected. For example, if you selected Alias Set from the artifact list, the **New Alias Set** dialog box appears.



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

   Note: Always create an artifact in the **Artifacts** folder. If you create an artifact directly under the project root, the artifact is not installed properly in the repository.

4. Type 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 the properties for each artifact, refer to the associated chapters in this guide.

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

2.3.2 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 Documentum 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, Documentum Composer tries to import all other required artifacts from the repository. "Documentum Composer reference projects" on page 20 provides instructions to create references between projects.

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. Right-click in the Documentum Navigator area and select **Import > Artifacts from Repository** from the pop-up menu.

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

2. Enter the project and repository information as described in the following table and then click **Login**.

Properties	Description
Project name	The name of an existing project into which the artifacts are imported. If you do not have an existing project, create a project before you can import any artifacts. "Creating a project" on page 17 provides more information about creating a project.
Repository name	The name of the repository that contains the artifacts.
User name	The name used to log in to the repository that contains the artifacts.
Password	The password used to log in to the repository that contains the artifacts.
Domain	The domain name of the repository. If the repository resides in a different domain than the client from which the repository is accessed, specify the domain name.

   If the login credentials for the repository are correct, you are logged in to the repository.

3. Click **Next**.

   The **Artifact Selection** dialog box appears.

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

   When you click on some of the available artifacts, such as FormTemplate, the **Properties** section appears and displays information about the selected artifact, such as name and value.

5. Select one or more objects from the **Choose From Available Artifacts** list, then click **Add**.

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

6. 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 do the following, you get duplicate artifacts in two different locations in your project:

1. Import an artifact from a repository.
2. Move the artifact to a different location within the project.
3. Import the artifact from the repository again.

2.4 Configuring project properties

You can configure various project properties by using the **Properties** dialog box, such as install options, OpenText™ Documentum™ Content Management Foundation SOAP API module options, and project install procedures.

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

The **Properties** dialog box appears.

For more information about configuring:

- "Designating reference projects for existing Documentum Composer projects" on page 23 provides information about project references.
- "Configuring Foundation SOAP API module options" on page 61 provides information about OpenText Documentum Content Management (CM) Foundation SOAP API module options.
- "Configuring the Foundation SOAP API services library" on page 62 provides information about Foundation SOAP API library options.
- "Configuring the project installation options" on page 207 provides information about project install options.
- "Configuring pre-installation and post-installation procedures" on page 210 provides information about project install procedures.

2.5 Localizing a Documentum Composer project

Documentum Composer currently only supports localization of type labels, type attribute labels, and UI strings within properties files. Documentum Composer also creates a directory structure for Business Object Framework (BOF) JAR files and Java libraries so you can specify localized JAR files, if you have them. Documentum Composer does not generate the localized JAR files themselves. It only creates the directory structure for you to copy the JAR files into.

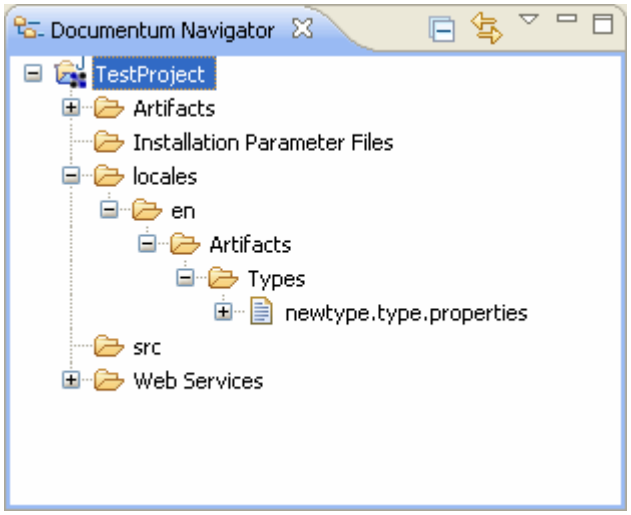
Before installing the DAR or Documentum Composer project with the localized data dictionary, enable the required locale in the repository. When a repository is created, a set of data dictionary information is loaded, based on the Documentum CM Server host locale. If the host locale is Russian, Arabic, Japanese, Korean, or Simplified Chinese, the data dictionary information for that locale is loaded during repository

creation. Otherwise, the host locale is always English. To add a locale, use the population script provided by Documentum. You can populate and publish the data dictionary file by following the procedures in the *Populating and Publishing the Data Dictionary* appendix in *OpenText Documentum Server Administration and Configuration Guide*.

To localize a Documentum Composer project:

1. For each type in the project, do the following:
   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.

   Documentum Composer generates a **locales** directory under the project 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.

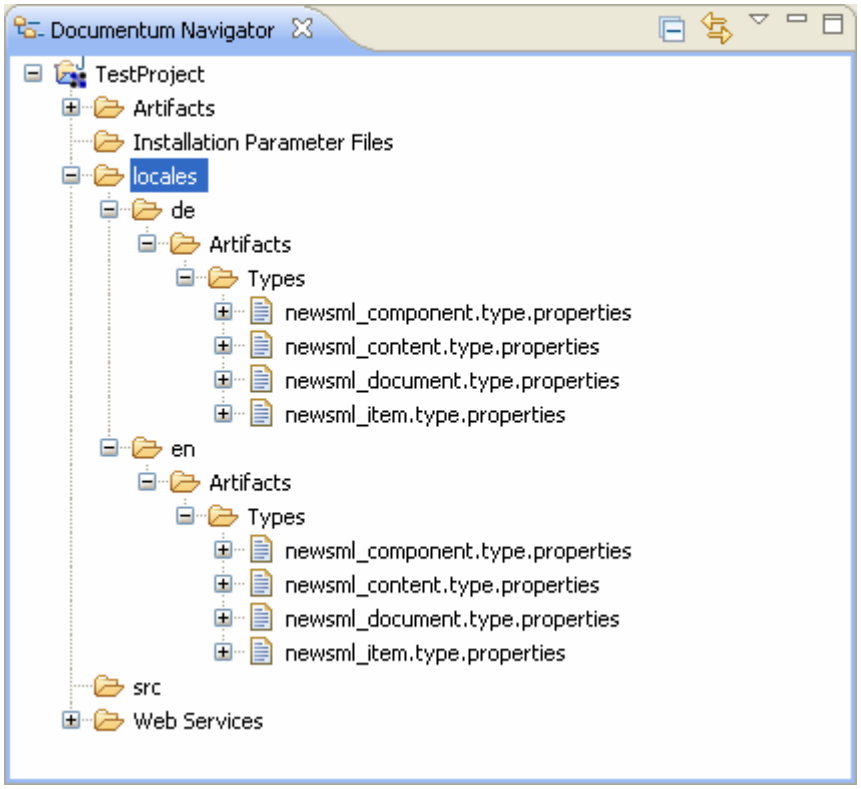
   If you import a project that is a different locale than the version of Documentum Composer that you are using, the localizable information in the project is still associated with the locale where it was originally created. The Documentum Composer that you are currently using still displays the changes and additions in the original locale. The localization template is also generated in the original locale.

4. If you have BOF JAR files or Java libraries that require localization, copy the English version of these JAR files into the appropriate directories in the `locales/<lang>/Artifacts` directory. The JAR file must contain the .properties files with the localizable strings. For example, if you had a type-based object (TBO) named my_tbo in your Documentum Composer project and generated a localization template, a corresponding `locales/<lang>/Artifacts/JARs/Modules/TBO/my_tbo` directory would be created so that you can place a localizable JAR file into it. If you do not have any localizable content for your BOF JAR files or Java libraries, delete the directories that you do not need. Documentum Composer generates the directories for BOF objects or JAR libraries even if the JAR file does not contain any localizable data. The directories that are created are described in the following list:

   - For a Library JAR: `locales/<lang>/Artifacts/Java Libraries/<library name>/`
   - For a standard Module JAR: `locales/<lang>/Artifacts/[Standard Module]/<module name>`
   - For a TBO/SBO Module JAR: `locales/<lang>/Artifacts/TBO/<module name>`
   - For another (typed) Module JAR: `locales/<lang>/Artifacts/<module type name>/<module name>`

   Note: When localizing the .properties files in your JAR files, append the locale string to the English .properties filename. For example, if the English .properties filename is `localizedStrings.properties`, name the localized file `localizedStrings_ja.properties` if you are translating to Japanese. Name the localized JAR filename the same as the English JAR filename.

5. 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, as shown in the following:



If you are sending out the .properties files for translation, use the following procedure to export the files:

a. In Documentum Composer, navigate to **File > Export**.

   The **Export** dialog box appears.

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

   The **File System** dialog box appears.

c. Expand the project that contains the *locales* folder that you want to export, then select 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**. Documentum 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.

f. Convert the translated labels to Unicode format for the following languages: Russian, Arabic, Japanese, Korean, or Simplified Chinese.

The translation team translates the strings on the right side of the equal sign (=) in the .properties file in the *locales* folder. Do not change the locales folder directory structure or .properties filenames.

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

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

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

      The **Import** dialog box appears.

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

      The **File system** dialog box 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 names are identical to the project name in the **From directory** field.

   f. Select **Create top-level folder**. If you do not want to confirm the import of each file, select **Overwrite existing resources without warning**.

   g. Click **Finish**.

7. Start the Install Wizard, select the **Install Localized Artifact Data** option, and then enter *<Composer_project_root>/locales* as the path to the localized data.

   "Installing a project" on page 214 provides information about how to install your project.

   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: Do not change labels, descriptions, or tabs, or move the .properties file after you create the localization template because the new language strings might not merge properly. If you rename labels and other

localizable application data, regenerate the localization template before you translate the strings.

2.6 Enabling tracing

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

Note: Because it impacts Documentum Composer performance, use tracing for debugging purposes only.

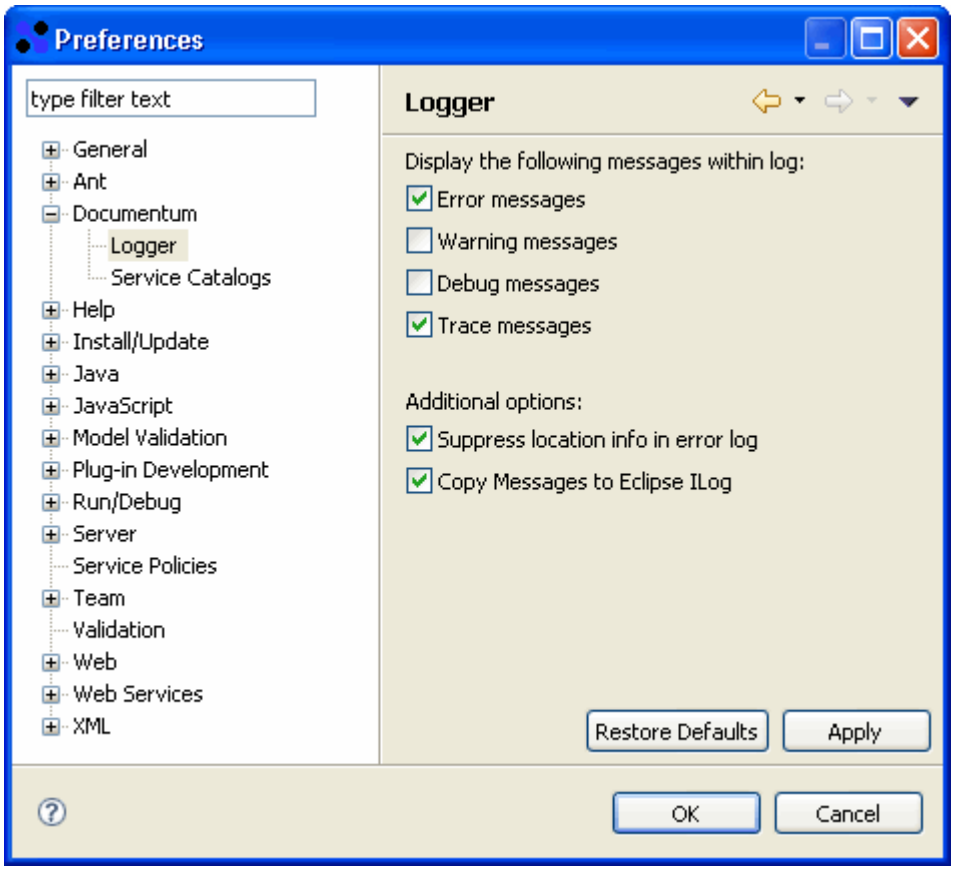
To enable tracing:

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

   The **Preference** dialog appears.

2. Click Documentum to expand the tree structure and select **Logger**.

   The **Logger** options appear.



3. Select **Trace Messages** and **Copy Messages to Eclipse ILog**, then click **OK**.

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

Chapter 3

Converting DocApps and DocApp Archives to Documentum Composer Projects

3.1 About DocApps and DocApp archives

Convert DocApps into Documentum Composer projects by importing the DocApp straight from the repository. You can also convert DocApp archives into Documentum Composer projects by having Documentum 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 Documentum Composer project.
- You can install the resulting Documentum Composer project to any 5.3 or later repository. You can install DocApps that you have converted to Documentum Composer projects to an older repository providing that the functionality is supported in the older repository. For example, if your DocApp has only custom subtypes of dm_document, you can convert a 6.0 DocApp into a Documentum Composer project and install it to a 5.3 repository. However, if the DocApp contains artifacts not supported by Documentum 5.3, such as Smart Containers or Aspects, you cannot install it to a 5.3 repository.
- You cannot convert a DocApp or DocApp archive that is stored in a repository earlier than version 5.3. If you want to convert, upgrade the DocApp or DocApp archive to version 5.3.
- If you are upgrading the repository, do so before converting the DocApp.

3.2 Converting a DocApp to a Documentum Composer project

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

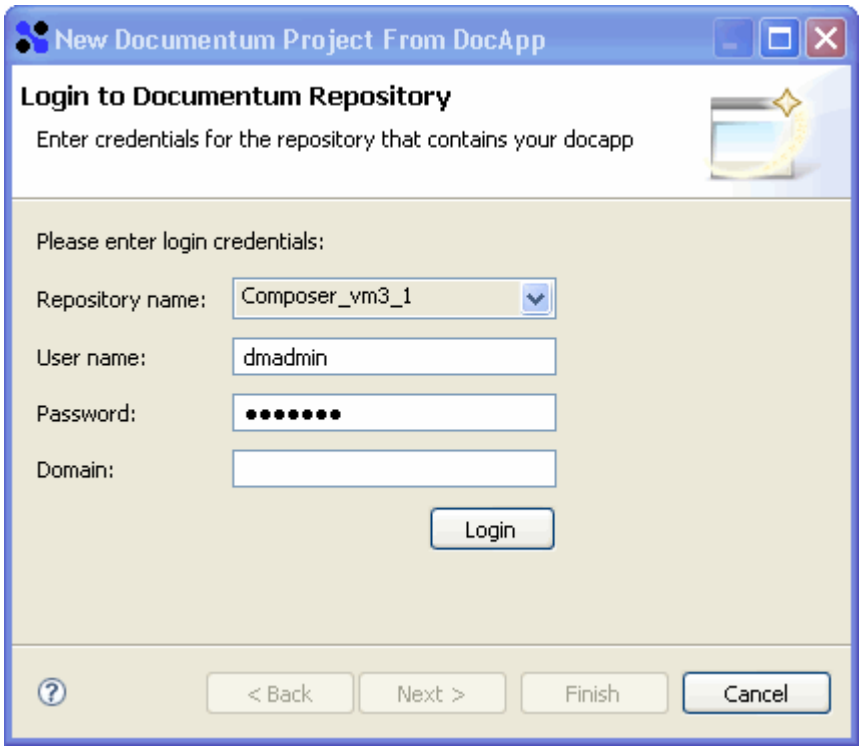
To convert a DocApp to a Documentum Composer project:

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, convert the DocApp containing the JAR files first.
2. Import all Documentum 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. "Documentum Composer reference projects" on page 20 provides information about reference projects.

3. Right-click in the **Documentum Navigator** view and select **New > Documentum Project From Repository DocApp**.

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



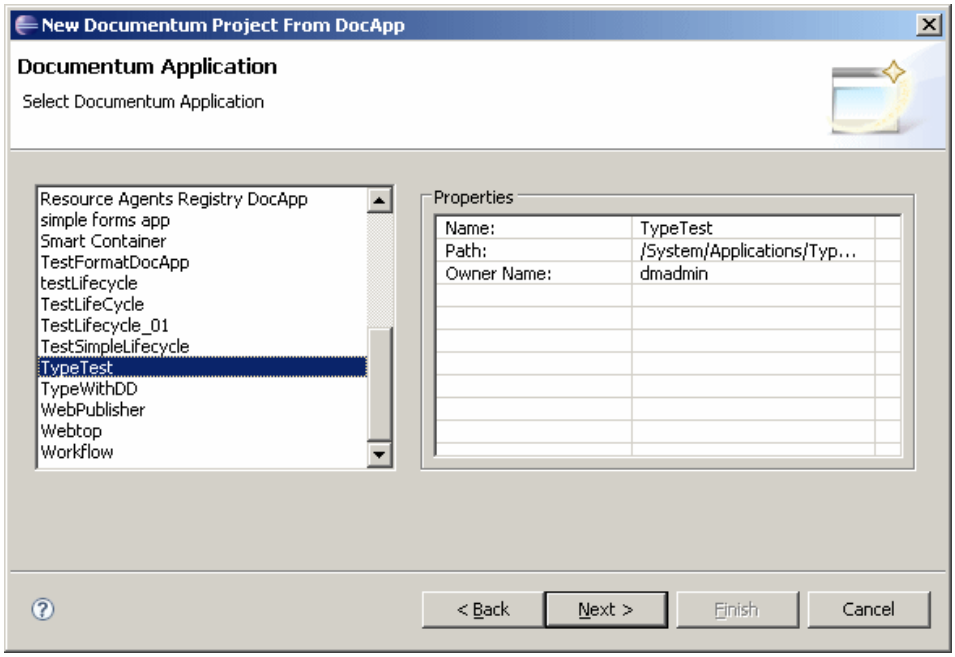
4. Enter your login credentials for the repository that contains the DocApp that you want to import, as described in the following table and then click **Login**.

Properties	Description
Repository	Required parameter. The name of the repository. 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. If the repository resides in a different domain than the client from which the repository is accessed, specify the domain.

Documentum Composer connects to the repository and verifies your login credentials.

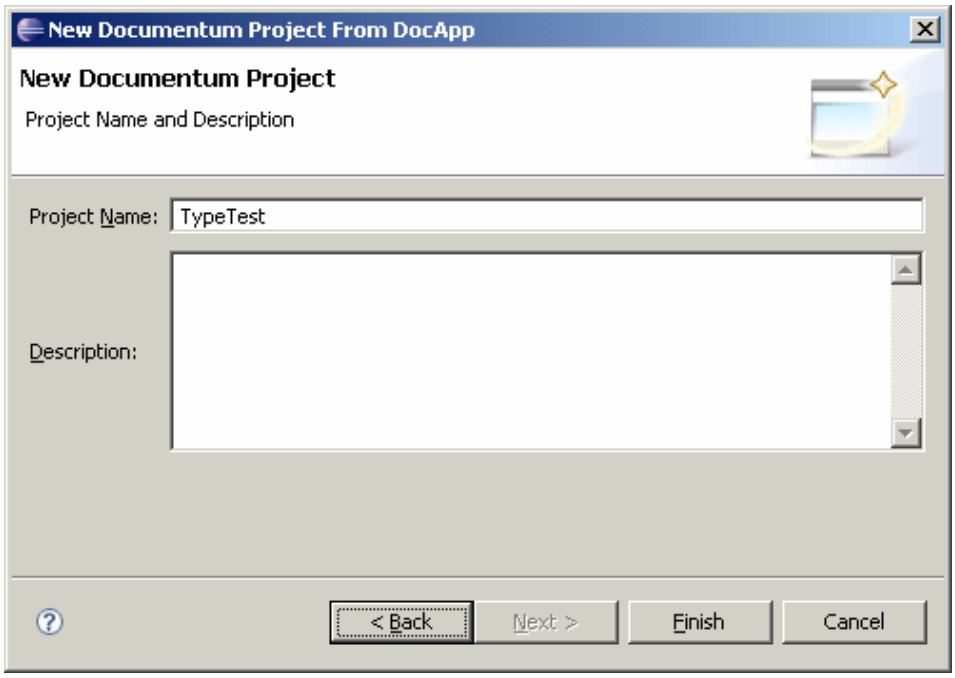
5. After your login credentials have been verified, click **Next**.

   The **Documentum Application** dialog appears.



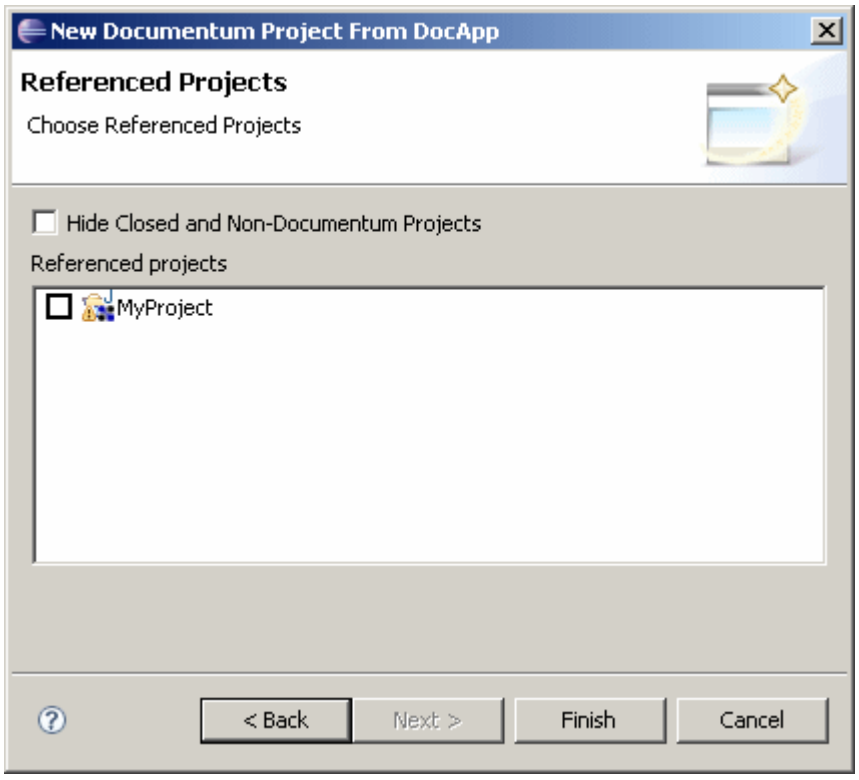
6. Select the DocApp that you want to convert from the listbox, then click **Next**.

   The **New Documentum Project** appears.



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

   The **Referenced Projects** dialog box appears.



8. Select the projects that your project references and click **Finish**.

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

3.3 Converting a DocApp archive to a Documentum Composer project

To convert a DocApp archive to a Documentum Composer project, Documentum Composer installs it to a target repository and creates a project from the DocApp archive.

3.3.1 Preparing for DocApp archive conversion

Before you convert your DocApp archive into Documentum Composer, ensure that you meet the following requirements:

- The target repository for the DocApp archive is a clean repository, meaning that it does not contain any remnant DocApps or artifacts. This repository is where you plan to deploy future changes to the resulting Documentum Composer project.
- Verify that the DocApp archive you are converting is version 5.3 or later. You cannot convert DocApp archives earlier than version 5.3. If your DocApp archive is a version earlier than 5.3, 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. "Configuring the connection broker" on page 13 provides information about configuring the connection broker.
- Verify that you have SUPERUSER privileges to access the target repository.
- If the DocApp archive you want to convert depends on other DocApps, convert those DocApps first. Also, ensure that the resulting Documentum Composer project is in your workspace. If your DocApp archive depends on other reference projects, import the reference projects into your workspace as well. "Documentum Composer reference projects" on page 20 provides information about reference projects.

3.3.2 Converting a DocApp archive

Documentum Composer lets you convert a DocApp archive into a Documentum Composer project.

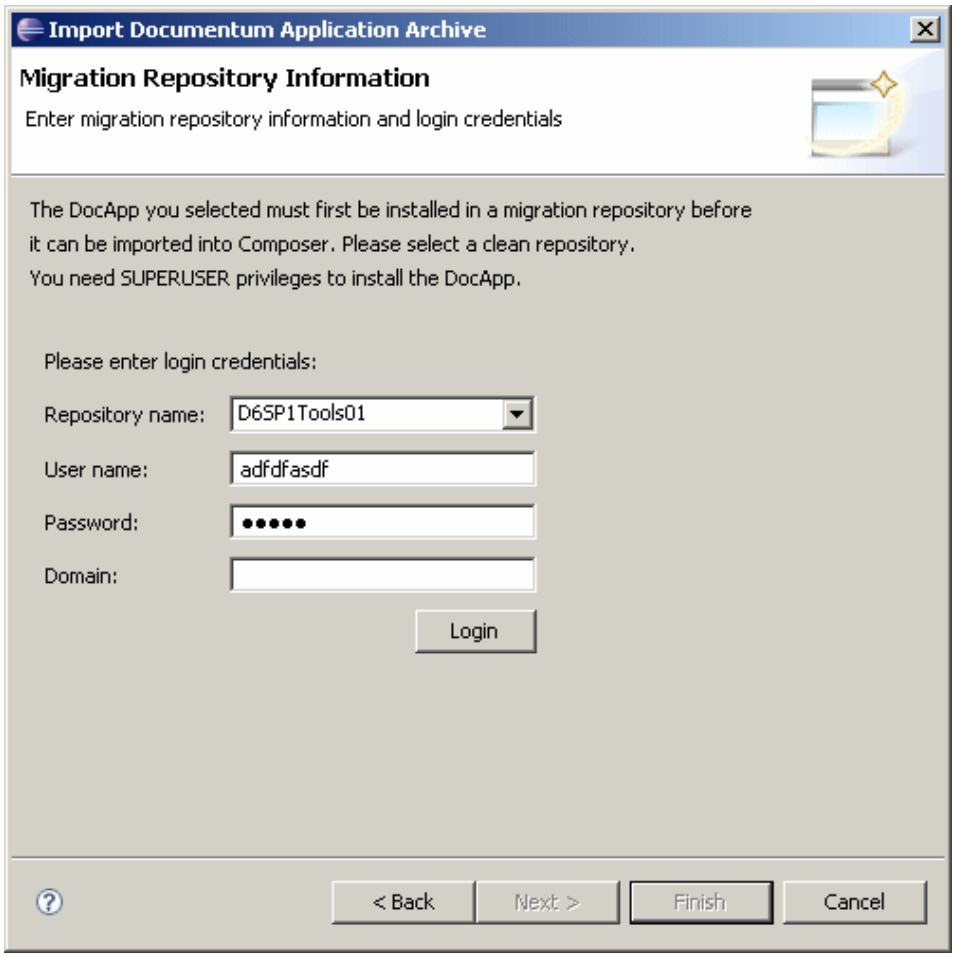
To convert a DocApp archive:

1. Unzip the DocApp archive to a folder on your local machine or a network drive.
2. Right-click in the **Documentum Navigator** view and select **New > Documentum Project from Local DocApp Archive**.

   The **Archive and Project Information** dialog box appears.
3. Enter the folder name for the DocApp archive and a project name, as described in the following table and then click **Next**.

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

   The **Migration Repository Information** dialog box appears.



4. Enter the migration repository information for the target repository, as described in the following table:

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

Properties	Description
Domain	The domain name of the target repository. If the target repository resides in a different domain than the client from which the repository is accessed, specify the domain name.

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

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

3.4 Post-conversion tasks

After Documentum Composer converts the DocApp or DocApp archive into a Documentum Composer project, complete the following steps:

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

  You might have to modify certain scripts to avoid artifact duplication or conflicts. For example, Documentum Composer creates install parameters for users. The pre-install script must create these users or the users must exist in the target repository where the project is to be installed.

Chapter 4

Documentum Composer and xCelerated Composition Platform

4.1 About Documentum Composer and xCelerated Composition Platform

Documentum xCelerated Composition Platform (xCP) is an application composition platform. It provides pre-integrated technologies that combine the power of enterprise content management, business process management, intelligent capture, customer communications management, collaboration, and compliance. It offers rapid application development tools along with deployment best practices to deliver solutions faster and at a lower cost. Because xCP involves composition of services, components, and user interface elements with little or no coding, deployment of a solution requires fewer resources and reduces project risks.

xCP includes the following Documentum products:

- Documentum Composer
- Documentum CM Server
- Business Activity Monitor
- OpenText™ Documentum™ Content Management Process Engine
- Process Integrator
- TaskSpace
- Process Reporting Services
- Process Analyzer (optional add-on, licensed separately)
- xCP Designer

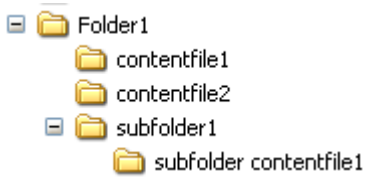
In the context of xCP, Documentum Composer is used mainly as a packaging and deployment tool for TaskSpace applications or xCP 1.x artifacts. TaskSpace applications serve as a container for related xCP 1.x artifacts, which can then be imported into Documentum Composer as projects and packaged as DAR files.

Note: Documentum Composer does not support importing xCP 2.x artifacts because the artifacts listed in Sysobjects do not have proper object names. In addition, when the artifacts are installed on another repository the references are lost.

4.2 Considerations for packaging and installing TaskSpace applications or xCP artifacts

When packaging and deploying TaskSpace applications or xCP artifacts with Documentum Composer, be aware of the following:

- If a TaskSpace application or xCP artifact changes in the source repository and you want to repackage it into a Documentum Composer project, create a Documentum Composer project and perform a clean import of the artifacts. Do not use a previously created Documentum Composer project that contains old versions of the artifacts.
- You can split a TaskSpace application into multiple Documentum Composer projects to increase portability and manageability. If a part of your application is updated frequently, bundle those artifacts into one Documentum Composer project and keep the remaining application in another Documentum Composer project. If the project contains many artifacts, you can split up TaskSpace applications into multiple Documentum Composer projects. For example, you can have a separate Documentum Composer project to contain the TaskSpace application. If the project is modified, it is a best practice to repackage and rebuild all of your related Documentum Composer projects.
- When importing a process that uses task forms with embedded forms inside them, the embedded forms are automatically included in the Documentum Composer project.
- When installing a process that uses task forms with embedded forms inside them, the embedded forms do not inherit the installation upgrade setting of the process. For example, if the process is set to *Version*, the task form is automatically set to Version, regardless of the individual settings of the task form. Embedded forms inside the task form do not inherit the installation upgrade setting.
- Before installing any project or DAR files that have form templates in them, install OpenText Documentum Content Management (CM) Process Engine or the `FORMS.dar` file in the target repository.
- To associate user contents with the TaskSpace application, such as sample documents or libraries of documents, add the corresponding folder to the TaskSpace application. For example, consider the following folder structure in the repository:



To associate all of the contents under Folder1, add Folder1 to the TaskSpace application. To associate only the subfolder1 contents, add only subfolder1 to the TaskSpace application.

- If you are importing a TaskSpace application, some content and artifacts are not imported. Manually import the following items:
  - ACLs and ACL templates (Documentum Composer creates installation parameters for the ACLs and ACL templates, but does not import the artifacts themselves.)
  - A group's children groups.
  - Alias sets that a process references.
  - User content that is not explicitly associated with the TaskSpace application, such as sample documents or libraries of documents.
  - Custom activity templates that are implemented as BOF modules.
  - Necessary .class files for custom workflow methods that are not implemented as BOF modules.
  - BOF modules that a method references when importing the method.
  - .class files for custom form adapters:
    - Copy the .class file to the application server if the custom form adapter was built this way. If the custom form adapter was built using BOF modules, then the BOF module and related JAR files are imported automatically.
    - For adapters that use JDBC for connectivity, copy the properties file that contains the JDBC string to the application server.
    - If the form adapter fetches data from a properties file, manually import the properties file.
  - Custom types that are used in form adapters for items such as drop-down menus. It is recommended that you add these artifacts to the TaskSpace application explicitly to ensure that they are imported into Documentum Composer.
  - The data (objects) in the custom types that are used in form adapters.
  - Registered tables
  - Form instances

4.3 Documentum Composer projects and DAR files

You can package TaskSpace applications or xCP artifacts into Documentum Composer projects or as DAR files. DAR files are read-only binary representations of a Documentum Composer project.

Documentum Composer projects are useful to develop and deploy your applications within the Documentum Composer IDE. DAR files are useful if you want to decouple development and deployment. A developer would typically hand off a DAR file to an administrator for deployment in this scenario.

The following table describes the main differences between a DAR file and a Documentum Composer project:

DAR files	Documentum Composer projects
A single file that is a binary representation of a Documentum Composer project. A DAR file must be generated from a Documentum Composer project.	A set of files that contain the source for a Documentum application.
Cannot be edited.	Can be edited.
Cannot be converted into a Documentum Composer project.	Can be built into a DAR file.
Installed with the DAR Installer or Headless Composer.	Installed with the Documentum Composer UI or Headless Composer (must be built into a DAR first.

4.4 Packaging TaskSpace applications

Documentum Composer provides a separate plugin to import Documentum TaskSpace applications as Documentum Composer projects. Documentum Composer can also package the project into a DAR file, so that it can be deployed in other repositories.

4.4.1 Packaging a TaskSpace application with Documentum Composer

Packaging a TaskSpace application involves creating a project in Documentum Composer and importing the TaskSpace application. The TCMReferenceProject is now packaged with Documentum Composer, so you no longer have to designate this project as a reference project manually.

To package a TaskSpace application with Documentum Composer:

1. Right-click in the **Documentum Navigator** area and select **New > Project > Documentum Project from TaskSpace Application**.

   The **New Documentum Project From TaskSpace Application in the Repository** dialog box appears.

2. Enter your login credentials for the repository that contains the TaskSpace application and then click **Login**.
3. Select the TaskSpace application that you want to import into Documentum Composer from the list on the left and click **Next**.
4. Select any necessary reference projects for your TaskSpace application and click **Finish** to import the TaskSpace application from the repository. After the import, if you are prompted to switch to the Documentum Artifacts perspective, do so.
5. To obtain the DAR file to install with the DAR Installer or Headless Composer, complete one of the following steps:
   - If you have the **Project > Build Automatically** option turned on, you can obtain the `<project>.dar` file from the `...\<workspace>\<project-name>\bin-dar` directory.
   - If you have the **Project > Build Automatically** option turned off, right-click the TaskSpace project you want to build and select **Build Project** from the drop-down list.

   Documentum Composer builds the TaskSpace project and generates a `<project>.dar` file in the `...\<workspace>\<project-name>\bin-dar` directory.

4.4.2 Packaging a TaskSpace application with Headless Composer

Headless Composer provides Ant tasks to import TaskSpace applications as Documentum Composer projects and to build the Documentum Composer project into a DAR file.

To package a TaskSpace application with Headless Composer:

1. Create a build directory to hold the files for your build.
2. In your build directory, create a file named `build.xml`. Open the file for editing.
3. In the `build.xml` file, create a target to create a Documentum Composer project with the emc.createTaskSpaceApplicationProject task. The task creates a Documentum Composer project that has the same name as the TaskSpace application, and it imports the TaskSpace application from the repository.
4. Create a target to build the project with the emc.build task. Call this task before the emc.dar task to ensure that the DAR file contains the latest built code.
5. Create a target to generate the DAR file with the emc.dar task. The following script shows you how to create the Ant targets for an example project. You can modify the property values at the top of the script for your environment.

```
<?xml version="1.0"?>
        <project name="xCPBuild">
        <property name="project.name" value="project_name" />
        <property name="repository" value="repository" />
```

```
        <property name="username" value="username" />
        <property name="password" value="password" />
        <property name="dar.filename" value="myDAR.dar" />

        <target name ="create-taskspace-project" description="Create a Composer
            project from a TaskSpace application in the repository">
            <emc.createTaskSpaceApplicationProject name="${project.name}"
                docbase="${repository}
                username="${username}"
                password="${password}">
            </emc.createTaskSpaceApplicationProject>
        </target>

        <target name="build-project" description="Build the project">
            <emc.build dmproject="${project.name}" failonerror="true"/>
        </target>

        <target name="package-project"
            description="Package the project into a DAR for installation">
            <delete file="${dar.filename}" />
            <emc.dar
                dmproject="${project.name}"
                manifest="bin/dar/default.dardef.artifact"
                dar="${dar.filename}" />
        </target>
</project>
```

6. Create a batch file, `build.bat`, to run the build. The batch file sets up the Documentum Composer workspace and calls the Ant script. When calling the Ant script, call the targets in the exact order as shown in the example. It is important to preserve the order of how the targets are run. In general, you create the Documentum Composer project, import the artifacts into the project, build the project, and then generate the DAR file. The following batch file shows how to run the example `build.xml` Ant script:

```
REM Set environment variables to apply to this command prompt only
SETLOCAL

REM Sets the root location of headless Composer
SET ECLIPSE="C:\ComposerHeadless"

REM Sets the workspace directory where Composer builds the projects
REM that you want to install to a repository
SET BUILDWORKSPACE=".\build_workspace"

REM Sets the Ant script that builds your projects
SET BUILDFILE=".\build.xml"

REM Delete old build and installation workspaces
RMDIR /S /Q %BUILDWORKSPACE%

REM Run Ant scripts to build the project.
REM The JAVA command must be on one line.
JAVA -cp %ECLIPSE%\startup.jar org.eclipse.equinox.launcher.Main -data
        %BUILDWORKSPACE%
-application org.eclipse.ant.core.antRunner -buildfile %BUILDFILE%
        create-taskspace-project
build-project package-project
```

7. Run the `build.bat` file from the command line. When the job completes, the DAR file is output to the location that you specified.

4.5 Packaging xCP artifacts

Packaging xCP artifacts outside the context of a TaskSpace application is useful if you want to separate a TaskSpace application into multiple projects. It is also useful if you want to modify a few xCP artifacts in a TaskSpace application. You can package individual xCP artifacts with Documentum Composer or Headless Composer.

4.5.1 Packaging xCP artifacts with Documentum Composer

Using the Documentum Composer UI enables you to select relevant artifacts for packaging and is useful for non-repeatable or one time packaging of xCP artifacts. This process is no different from importing artifacts from a repository.

To package xCP artifacts with Documentum Composer:

1. Import the desired artifacts as described in "Importing artifacts" on page 28.
2. To obtain the DAR file to install with the DAR Installer or Headless Composer, complete one of the following steps:
   - If you have the **Project > Build Automatically** option turned on, you can obtain the `<project>.dar` file from the `...\<workspace>\<project-name>\bin-dar` directory.
   - If you have the **Project > Build Automatically** option turned off, right-click the TaskSpace project you want to build and select **Build Project** from the drop-down list.

   Documentum Composer builds the TaskSpace project and generates a `<project>.dar` file in the `...\<workspace>\<project-name>\bin-dar` directory.

4.5.2 Packaging xCP artifacts with Headless Composer

Headless Composer provides Ant tasks to create a project and import artifacts directly from a repository. Using the Ant tasks is a helpful way of automating migration of xCP applications from a development environment to a production or QA environment. The following procedure describes how to package an existing xCP application that resides in a repository into a Documentum Composer project. It also describes how to generate a DAR file that can be deployed on another system.

To package xCP artifacts with Headless Composer:

1. Create a build directory to hold the files for your build.
2. In your build directory, create a file named `build.xml`. Open the file for editing.
3. In the `build.xml` file, create a target to create a Documentum Composer project with the emc.createArtifactsProject task.
4. Create a target to import the desired xCP artifacts into the project with the emc.importArtifacts task.

5. Create a target to build the project with the emc.build task. Call this task before the emc.dar task to ensure that the DAR file contains the latest built code.

6. Create a target to generate the DAR file with the emc.dar task. The following script shows you how to create the Ant targets for an example project. You can modify the property values at the top of the script for your environment.

```
<?xml version="1.0"?>
        <project name="xCPBuild">
        <property name="project.name" value="project_name" />
        <property name="repository" value="repository" />
        <property name="username" value="username" />
        <property name="password" value="password" />
        <property name="artifact.path"
         value="/System/Applications/app_name/artifact_name" />
        <property name="dar.filename" value="myDAR.dar" />

      <target name ="create-project" description="Create a project to import
       artifacts into">
       <emc.createArtifactProject name="${project.name}" overwrite="true">
       </emc.createArtifactProject>
      </target>

   <target name ="import-project" description="
         Must import a project before updating, building, or installing it">
    <emc.importProject dmproject="HelloWorldArtifacts" failonerror="true"/>
      </target>

        <target name="import-artifacts">
            <emc.importArtifacts project="${project.name}" docbase="${repository}"
             username="${username}" password="${password}">
               <objectIdentities>
                    <path value="${artifact.path}"/>
                </objectIdentities>
            </emc.importArtifacts>
        </target>

        <target name="build-project" description="Build the project">
            <emc.build dmproject="${project.name}" failonerror="true"/>
        </target>

        <target name="package-project"
          description="Package the project into a DAR for installation">
            <delete file="${dar.filename}" />
            <emc.dar
              dmproject="${project.name}"
              manifest="bin/dar/default.dardef.artifact"
              dar="${dar.filename}" />
        </target>

</project>
```

7. Create a batch file, `build.bat`, to run the build. The batch file sets up the Documentum Composer workspace and calls the Ant script. When calling the Ant script, call the targets in the exact order as shown in the example. It is important to preserve the order of how the targets are run. In general, you create the Documentum Composer project, import the artifacts into the project, build the project, and then generate the DAR file.

   Note: Ensure the required version of Java JDK is installed. The *OpenText Documentum Composer Release Notes* contains the supported Java JDK version.

   The following batch file shows how to run the example `build.xml` Ant script:

```
REM Set environment variables to apply to this command prompt only
SETLOCAL

REM Sets the root location of headless Composer
SET ECLIPSE="C:\ComposerHeadless"

REM Sets the workspace directory where Composer builds the projects
REM that you want to install to a repository
SET BUILDWORKSPACE=".\build_workspace"

REM Sets the Ant script that builds your projects
SET BUILDFILE=".\build.xml"

REM Delete old build and installation workspaces
RMDIR /S /Q %BUILDWORKSPACE%

REM Run Ant scripts to build the project.
REM The JAVA command must be on one line.
JAVA -cp %ECLIPSE%\startup.jar org.eclipse.equinox.launcher.Main -data
        %BUILDWORKSPACE%
-application org.eclipse.ant.core.antRunner -buildfile %BUILDFILE%
        create-project
import-artifacts build-project package-project
```

When the `build.bat` file is done running, a DAR file appears in the location that you specified for the ${dar.filename} property. You can deploy the DAR with the DAR Installer or with Headless Composer.

4.6 Installing TaskSpace applications and xCP artifacts

Installing TaskSpace applications and xCP artifacts follow the same process as any other Documentum Composer project. Use the Documentum Composer UI to install Documentum Composer projects, the DAR Installer to install DAR files, or Headless Composer to install Documentum Composer projects and DAR files.

4.6.1 Installing a TaskSpace application and xCP artifacts with Documentum Composer

After you import a TaskSpace application successfully into Documentum Composer as a project, use Documentum Composer to install the project into a repository. The process is the same as installing any other Documentum Composer project as described in “Installing a project” on page 214.

4.6.2 Installing TaskSpace applications and xCP artifacts with the DAR installer

If you have a TaskSpace application DAR file or xCP DAR file, you can install it with the DAR Installer. The process is the same as installing any other Documentum Composer DAR file as described in "Installing a DAR file with the DAR installer" on page 221. You no longer must copy the TCMReferenceProject.dar file to the same directory as your DAR file for installation.

4.6.3 Installing TaskSpace applications and xCP artifacts with Headless Composer

To deploy TaskSpace applications or xCP artifacts with Headless Composer, package the application or artifacts into a DAR file as described in the following sections:

- "Packaging a TaskSpace application with Documentum Composer" on page 48
- "Packaging a TaskSpace application with Headless Composer" on page 49
- "Packaging xCP artifacts with Documentum Composer" on page 51
- "Packaging xCP artifacts with Headless Composer" on page 51

When you have built a DAR file from a Documentum Composer project, you can install the DAR file with the emc.install task.

To deploy xCP artifacts with Headless Composer:

1. Create a build directory to hold the files for your build.
2. Create a file named `install.xml`.
3. Create a target to install a DAR file with the emc.install task. The following script is an example of how to declare the emc.install task:

```
<?xml version="1.0"?>
    <project name="headless-install">

        <property name="repository" value="repository" />
        <property name="username" value="username" />
        <property name="password" value="password" />
        <property name="dar.filename" value="myDAR.dar" />

        <target name="install-project"
         description="Install the project to the specified repository.
         dfc.properties must be configured">
            <emc.install
             dar="${dar.filename}"
             docbase="${repository}"
             username="${username}"
             password="${password}"
             domain="" />
        </target>

    </project>
```

4. Create a batch file, `install.bat`, to run the installation script. The batch file sets up the Documentum Composer workspace and calls the Ant script. When calling the Ant script, call the targets in the exact order as shown in the example. The following batch file shows how to run the example `install.xml` Ant script:

```
REM Set environment variables to apply to this command prompt only
SETLOCAL

REM Sets the root location of headless Composer
SET ECLIPSE="C:\ComposerHeadless"

REM Sets the workspace directory where Composer extracts built DAR files
REM before installing them to a repository
SET INSTALLWORKSPACE=".\install_workspace"

REM Sets the Ant script that builds your projects
SET INSTALLFILE=".\install.xml"

REM Delete old build and installation workspaces
RMDIR /S /Q %INSTALLWORKSPACE%

REM Run Ant scripts to install the project
REM The JAVA command must be on one line.
JAVA -cp %ECLIPSE%\startup.jar org.eclipse.equinox.launcher.Main -data
%INSTALLWORKSPACE%
-application org.eclipse.ant.core.antRunner -buildfile %INSTALLFILE%
install-project
```

4.7 Building and installing Documentum Composer projects that already contain xCP artifacts with Headless Composer

If you have a Documentum Composer project that already contains xCP artifacts and you want to build and deploy the project, Headless Composer allows you to automate this procedure.

To build and deploy a Documentum Composer project that contains xCP artifacts:

1. Create a build directory to hold the files for your build.
2. In your build directory, create a file named `build.xml`. Open the file for editing.
3. In the `build.xml` file, create a target to import the Documentum Composer project that contains the xCP artifacts with the emc.import task.
4. Create a target to build the project with the emc.build task. Call this task before the emc.dar task to ensure that the DAR file contains the latest built code.
5. Create a target to generate the DAR file with the emc.dar task. Call this task before the emc.install task to ensure that the code built from emc.build task makes it into the new DAR file.
6. Create a target to install the DAR file with the emc.install task. The following script shows you how to create the Ant targets for an example project. You can modify the property values at the top of the script for your environment.

```
<?xml version="1.0"?>
        <project name="xCPBuild">
        <property name="project.name" value="project_name" />
        <property name="repository" value="repository" />
        <property name="username" value="username" />
        <property name="password" value="password" />
        <property name="dar.filename" value="myDAR.dar" />

      <target name ="import-project" description="
         Must import a project before updating, building, or installing it">
       <emc.importProject dmproject="${project.name}" failonerror="true"/>
     </target>

        <target name="build-project" description="Build the project">
            <emc.build dmproject="${project.name}" failonerror="true"/>
        </target>

        <target name="package-project"
          description="Package the project into a DAR for installation">
            <delete file="${dar.filename}" />
            <emc.dar
              dmproject="${project.name}"
              manifest="bin/dar/default.dardef.artifact"
              dar="${dar.filename}" />
        </target>

        <target name="install-project"
         description="Install the project to the specified repository.
         dfc.properties must be configured">
            <emc.install
             dar="${dar.filename}"
             docbase="${repository}"
             username="${username}"
             password="${password}"
             domain="" />
        </target>

</project>
```

7. Create a batch file, `build.bat`, to run the build. The batch file sets up the Documentum Composer workspace and calls the Ant script. When calling the Ant script, call the targets in the exact order as shown in the example. It is important to preserve the order of how the targets are run. In general, you create the Documentum Composer project, import the artifacts into the project, build the project, and then generate the DAR file. The following batch file shows how to run the example `build.xml` Ant script:

```
REM Set environment variables to apply to this command prompt only
SETLOCAL

REM Sets the root location of headless Composer
SET ECLIPSE="C:\ComposerHeadless"

REM Sets the location of your source projects.
REM This location gets copied into your build workspace directory
SET PROJECTSDIR="C:\Documents and Settings\Administrator\composer-workspace"

REM Sets the workspace directory where Composer builds the projects
REM that you want to install to a repository
SET BUILDWORKSPACE=".\build_workspace"

REM Sets the workspace directory where Composer extracts built DAR files
REM before installing them to a repository
SET INSTALLWORKSPACE=".\install_workspace"

REM Sets the Ant script that builds your projects
SET BUILDFILE=".\build.xml"
```

```
REM Delete old build and installation workspaces
RMDIR /S /Q %BUILDWORKSPACE%
RMDIR /S /Q %INSTALLWORKSPACE%

REM Copy source projects into build workspace
XCOPY %PROJECTSDIR% %BUILDWORKSPACE% /E

REM Run Ant scripts to build and install the projects
REM Each JAVA command must be on one line.
JAVA -cp %ECLIPSE%\startup.jar org.eclipse.equinox.launcher.Main -data
%BUILDWORKSPACE%
-application org.eclipse.ant.core.antRunner -buildfile %BUILDFILE%
import-project build-project package-project
JAVA -cp %ECLIPSE%\startup.jar org.eclipse.equinox.launcher.Main -data
%INSTALLWORKSPACE%
-application org.eclipse.ant.core.antRunner -buildfile %BUILDFILE%
install-project
```

4.8 Migrating a TaskSpace application or xCP artifacts from a source environment to a target environment

Documentum Composer allows you to migrate TaskSpace applications or individual xCP artifacts from a source environment to a target environment. This procedure is valid only if the source and target environment contain their own xCP stack (including Documentum CM Server, Process Engine, TaskSpace, and Documentum Composer). An example of this scenario is migrating a project from a development environment to a production environment.

Note: Follow this procedure to migrate a TaskSpace application or xCP artifacts from a D6.5 SP2 environment to a D6.6 or later environment.

4.8.1 Packaging the TaskSpace application or xCP artifacts on the source environment

Packaging your application involves building the artifacts into a Documentum Composer DAR file. Before beginning this procedure, the following prerequisites must be met:

- The target environment must be version 6.6 or later.
- Documentum Composer version 6.6 or later must be installed even if the source environment version is earlier than version 6.6. As a best practice, use the same version of Documentum Composer for both packaging and deployment.
- The `dfc.properties` file must point to a valid connection broker. The file is located in the folder `<Composer_root>\plugins\com.emc.ide.external.dfc_1.0.0\documentum.config`. "Configuring the connection broker" on page 13 provides information on how to configure the connection broker.

To migrate your TaskSpace application, package it as a DAR:

1. Start Documentum Composer version 6.6 and create a workspace.
2. Create a Documentum Composer project from a TaskSpace application as described in "Packaging a TaskSpace application with Documentum Composer" on page 48. To migrate individual xCP artifacts, import them as described in "Importing artifacts" on page 28.
3. If you have sample content that you want to migrate, such as folders or individual documents, import them individually as described in "Importing artifacts" on page 28.

   Documentum Composer automatically builds the project into a DAR file that you can use to install to the target repository. The DAR file is located in the <workspace_root>/<project_name>/bin-dar directory. If you do not see a DAR file, read the "Generating a DAR file" on page 213 section for information on how to generate one.
4. If the source environment cannot communicate with the target environment, zip the entire Documentum Composer project directory, <workspace_root>/<project_name>, and transfer it to the target environment.

   Note: To improve performance, transfer the Documentum Composer project directory to the target environment to decrease installation time.

4.8.2 Deploying the TaskSpace application or xCP artifacts on the target repository

When the DAR file is ready, deploy it to the target environment. Before you begin this procedure, Documentum Composer version 6.7 must be installed.

To deploy the TaskSpace application or xCP artifacts:

1. If you transferred the Documentum Composer project zip file to the target environment:
   a. Unzip the Documentum Composer project that contains the TaskSpace application or xCP artifacts to a directory of your choice.
   b. Copy the <Composer_project>/bin-dar/<xCP_app>.dar to a directory of your choice.
2. Run `<Composer_root>\dardeployer.exe`.
3. Install the DAR file with the DAR Installer.
4. If you migrated a TaskSpace application:
   a. Access the TaskSpace application by going to http://host:port/Taskspace/?appname=<TaskSpace_app_name> using a TaskSpace administrator username and password.

b. Navigate to the **Administration** tab.

c. Create users if desired.

d. Assign users to roles within the TaskSpace application.

4.8.3 Troubleshooting tips

If you receive an error regarding presets after trying to access the TaskSpace application on the target environment, it was not installed correctly.

To fix the presets error:

1. Log in to Documentum Administrator.
2. Go to the folder `System /Applications/<TaskSpace_application>`.
3. Verify that the owner of the following artifacts is dmc_wdk_presets_owner:
   - Presets
   - Presets / TaskSpace_App.definition
   - Presets / TaskSpace_Role.definition
   - Presets / Preset Packages
   - Files within the folder Presets / Preset Packages

Chapter 5

Managing Web Services

5.1 Web services

A web service is a software system designed to support interoperable machine-to-machine interaction over a network. Web services are often web APIs accessed over a network, such as the internet, and executed on a remote system hosting the requested services.

Documentum Composer supports web services by providing an OpenText™ Documentum™ Content Management Foundation SOAP API registry plugin. The plugin enables users to:

- Connect to a web services registry.
- Import WSDLs to create a Java client library.
- Create services.
- Export the services to an EAR file.

Documentum Composer includes a Foundation SOAP API Builder and a Foundation SOAP API Services Library for each new Documentum project. The Foundation SOAP API Builder and Foundation SOAP API Service Library can be configured in the property settings of a project.

5.2 Configuring Foundation SOAP API module options

Configure the Foundation SOAP API context root and module name for a project in the Foundation SOAP API Module dialog box.

To configure the Foundation SOAP API module options:

1. Right-click the project and select **Properties** from the drop-down list. The **Properties for [projectname]** dialog appears.
2. Expand **Documentum Project** and select **DFS Module**. The **DFS Module** dialog appears.
3. Type the Foundation SOAP API context root and module name in the **Context Root** and **DFS Module Name** fields.
4. Click **OK**.

5.3 Configuring the Foundation SOAP API services library

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

To configure the Foundation SOAP API services library:

1. Right-click the project and select **Properties** from the drop-down list. The **Properties for [projectname]** dialog appears.
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 Foundation SOAP API library to add, then click **Finish**.

5.4 Configuring catalog services

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

To configure catalog services:

1. In the Documentum Composer main menu, select **Window > Preferences**.

   The **Preferences** dialog appears.
2. In the **Preferences** list, double-click **Documentum** and select **Service Catalogs**. The **Service Catalogs** dialog appears.

   The **Service Registries** 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 the following table:

Properties	Description
Registry Alias	A string specifying a name for the service registry.
Retrieve URL	A string specifying the URL of the server that hosts the service. The URL must have the format http://<domain>:<port>/catalog/inquiry.

Properties	Description
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.
Retrieve Services	Select to display services from this service registry in the Service Catalog dialog.

5.5 Viewing web services

In Documentum 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 Documentum Composer main menu, navigate to **Window > Perspective > Open Perspective > Other**.

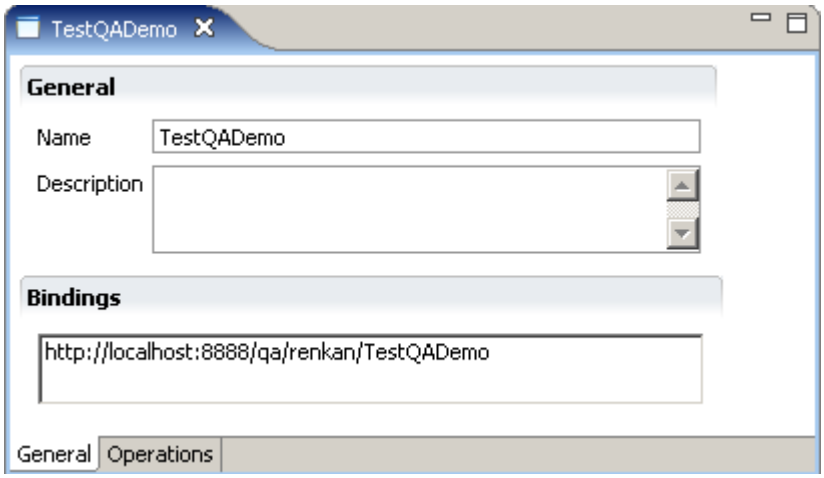
   The **Open Perspective** dialog appears.

2. Select **Documentum Solutions** and click **OK**.

   The **My Services** and **Service Catalogs** tabs appear.

   The **Service Catalogs** tab displays all configured service catalogs and services on the server. The **My Services** tab displays 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 Documentum Composer. If no services are published, Documentum 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.



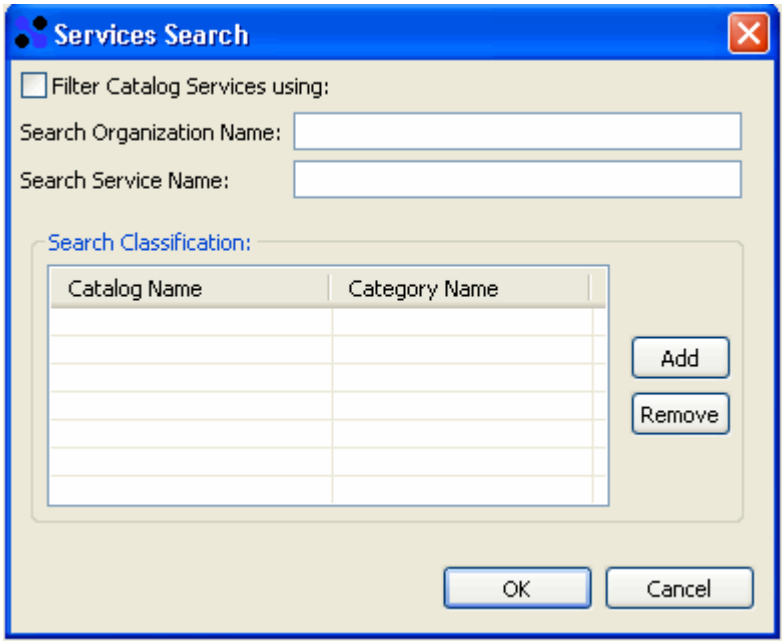
4. Click the **Operations** tab to view the service methods.

5.5.1 Filtering services

Specify which services 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 box appears.



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

3. Click **OK**.

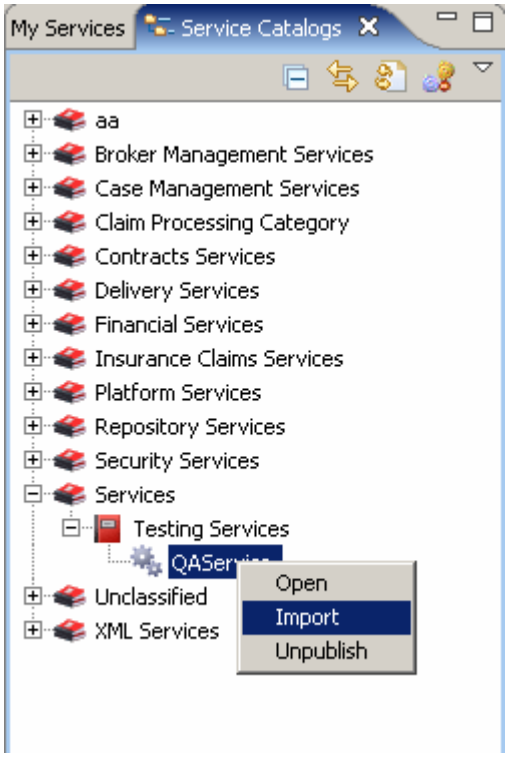
   The services that match the filter criteria are displayed in the **Catalog Services** view.

5.6 Generating a client proxy

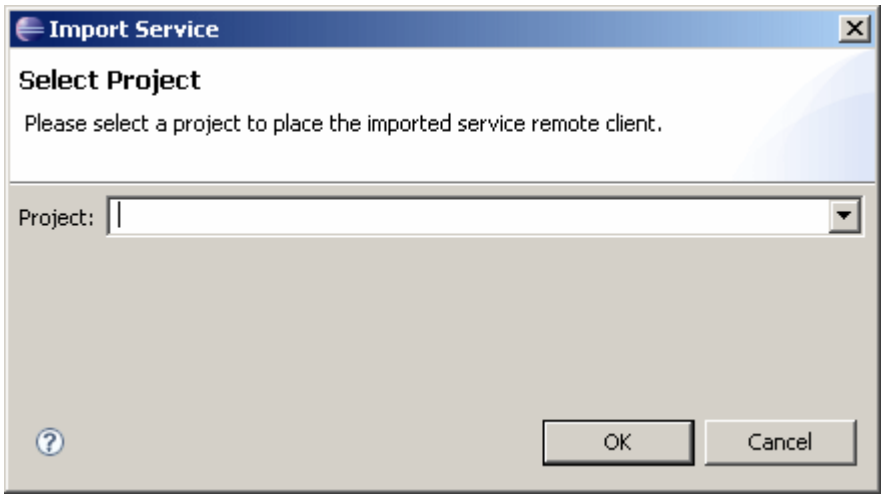
The Import option in the Service Catalogs tab lets you generate the client proxy of a service and make it available in a Documentum Composer project.

To generate the client proxy of a service:

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

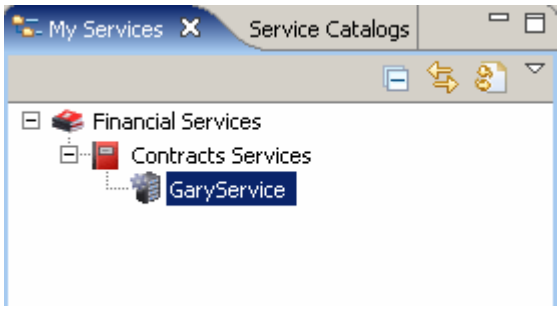


   The **Import Service** dialog appears.

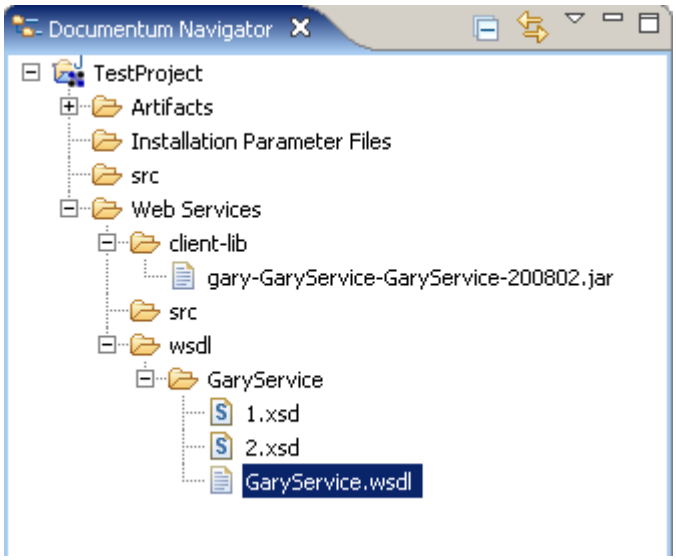


2. Enter the project in which you want to generate the client proxy or select a project from the drop-down list, then click **OK**.

   Documentum 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.



5.6.1 Consuming a service

Consuming a service requires importing the client proxy of a service, as described in "Generating a client proxy" on 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 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);
        }
    }
}
```

5.7 Creating a service

Create a service from a Java file or generate a service from the WSDL file of a client proxy.

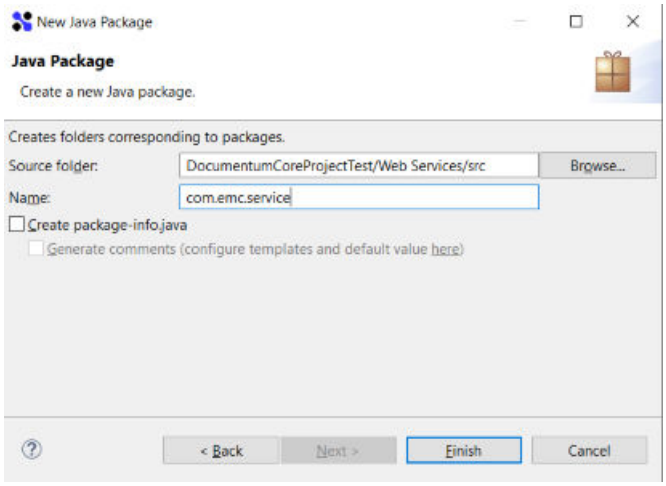
5.7.1 Creating a service from a Java file

To create a service, change to the **Package Explorer** view.

Note: This guide only describes how to create the Java file for the service, not how to develop a service. The *OpenText Documentum Foundation Services Development Guide* provides information about how to develop Foundation SOAP API services.

To create a service from a Java file:

1. Change to the **Package Explorer** view in Documentum 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 Services/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 that you created and select **New > Class**. The **New Java Class** dialog appears.

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

c. Write the code that specifies your service. The *OpenText Documentum Foundation Services Development Guide* provides information about developing Foundation SOAP API services.

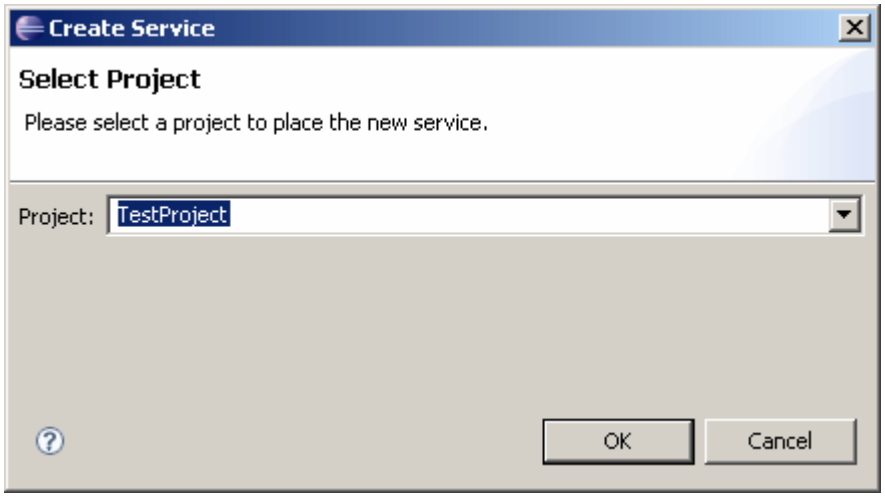
d. Save your changes. Your new service appears on the **My Services** tab in the **Documentum Solutions** perspective under **Unclassified**.

5.7.2 Creating a service from a WSDL

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.



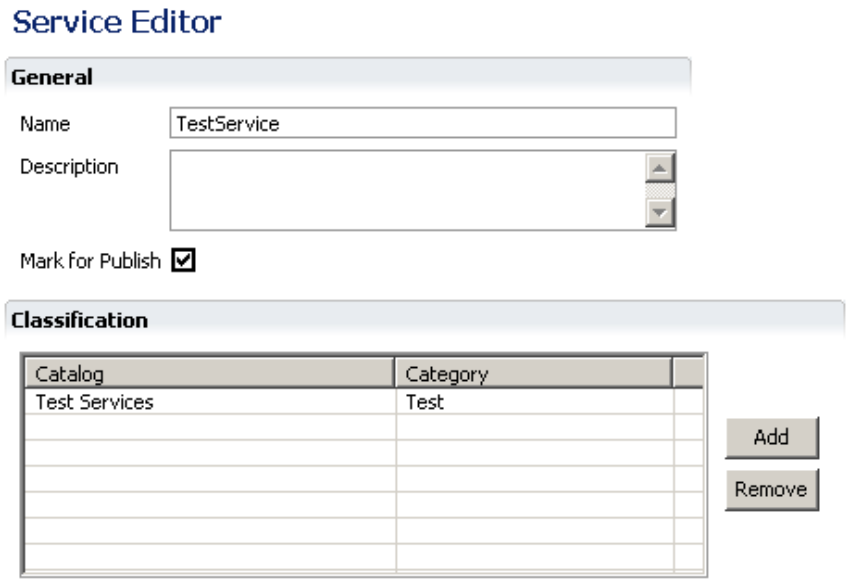
3. Type the name of the project in which you want Documentum Composer to create the service or select a project from the drop-down list, then click **OK**. The Java file for the service appears in the */src* directory of the **Web Services** folder for the project.

5.8 Modifying catalog and category information

When you first create a service, as described in "Creating a service from a Java file" on page 68, the service 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 the following table:

Properties	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.

Properties	Description
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.

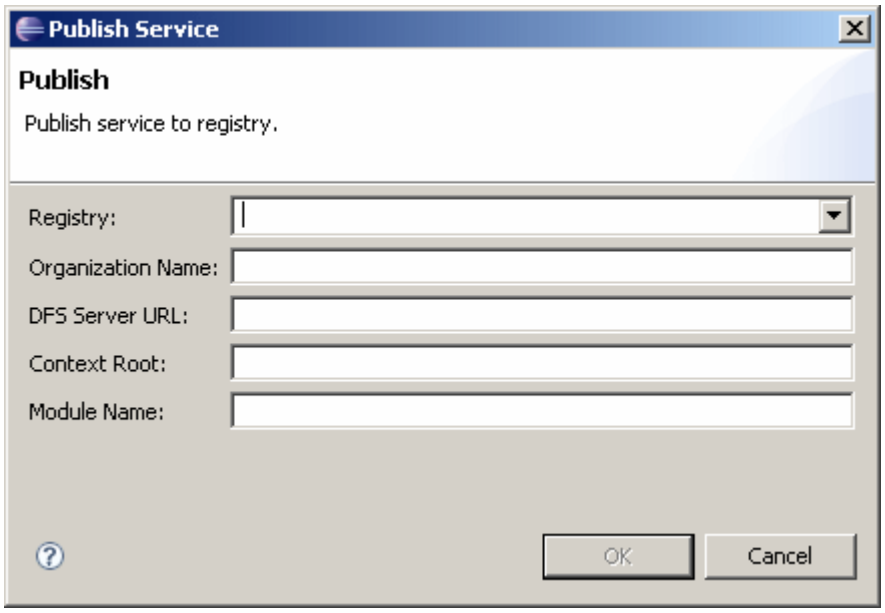
3. Save your changes. The new or modified catalog name and category appear in the **My Services** tab.

5.9 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 the following table and then click **OK**.

Properties	Description
Registry	The alias of the registry to which the service is published. Select the registry from the drop-down list.

Properties	Description
Organization Name	The name of the organization where the service resides.
Foundation SOAP API Server URL	The URL of the Foundation SOAP API 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. If the service was published successfully, it appears in the **Service Catalogs** list.

5.10 Unpublishing a service

When you unpublish a service, it is no longer available on the Foundation SOAP API 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 no longer appears in the **Service Catalogs** view.

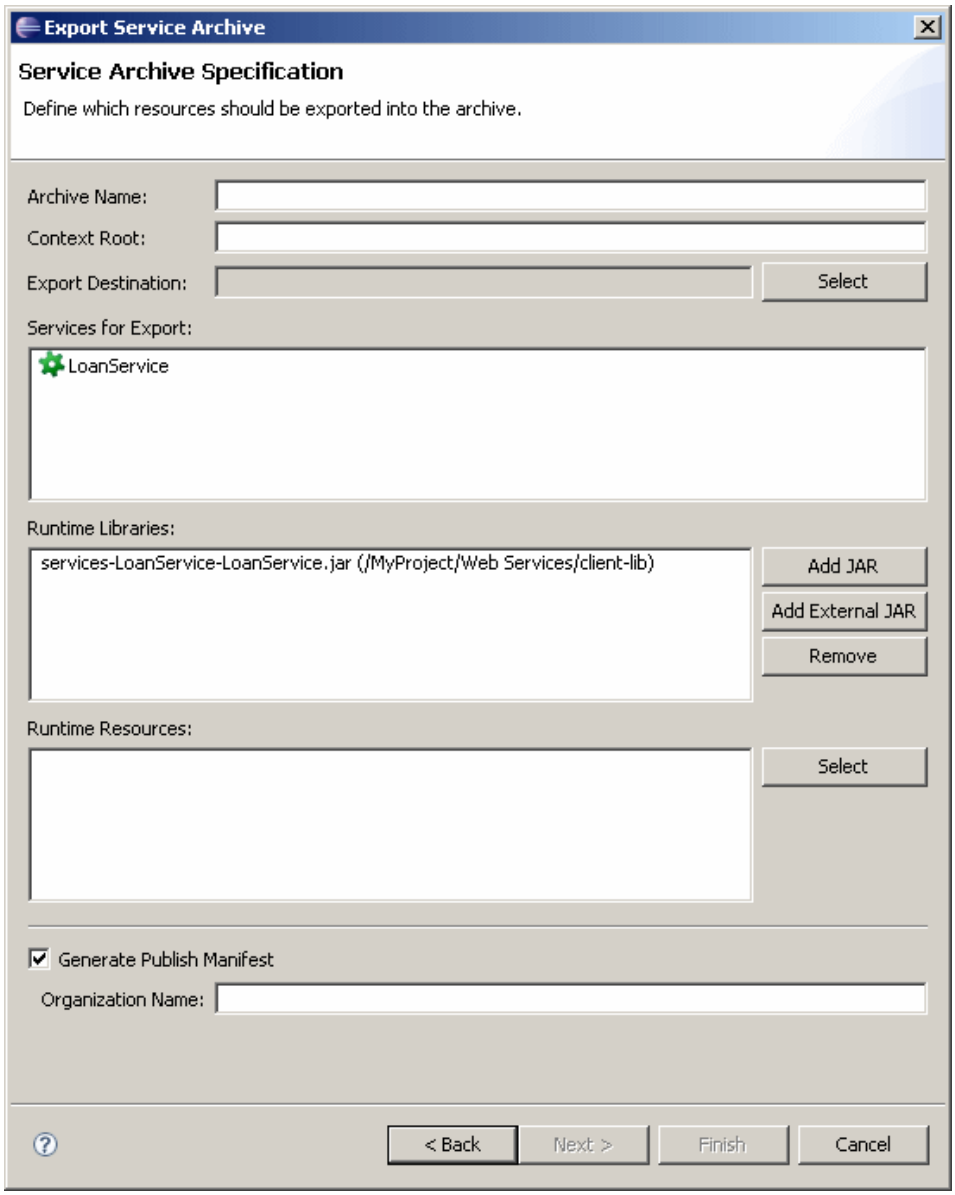
5.11 Exporting a service

When you export a service, Documentum 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 in the **Documentum Navigator** or **Package Explorer** view.

2. Navigate to **Export > Documentum > Export Service Archive** and then click **Next**. The **Export Service Archive** dialog appears.



3. Enter the export service information as described in the following table and then click **Finish**.

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

Properties	Description
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 export.
Runtime Libraries	The JARs that are included in and exported with the EAR file. Click **Add** to include JAR files from your Documentum 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 select the resources to export.
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.

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

5.12 Deploying a service

After you export the service and generate an EAR file, deploy the service by copying the EAR file to the Foundation SOAP API server.

To deploy a service:

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

Chapter 6

Managing Alias Sets

6.1 Alias, alias values, and alias sets

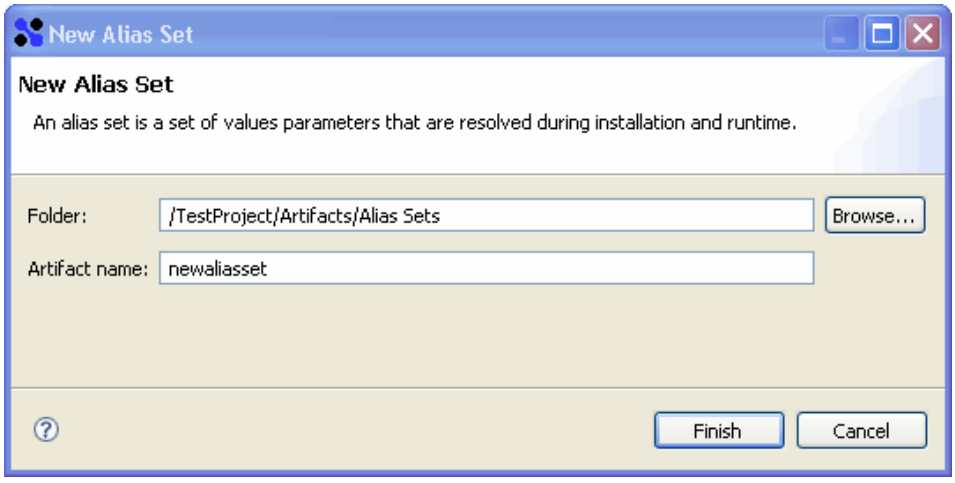
Many aspects of Documentum applications involve references to specific users, groups, permission sets, and repository cabinets and folders. Instead of referencing the actual user, group, cabinet, or folder name, you can assign an alias, a symbolic name. The symbolic name is called the *alias name*. The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

6.2 Creating an alias set

Use the Alias Set editor to create an alias set.

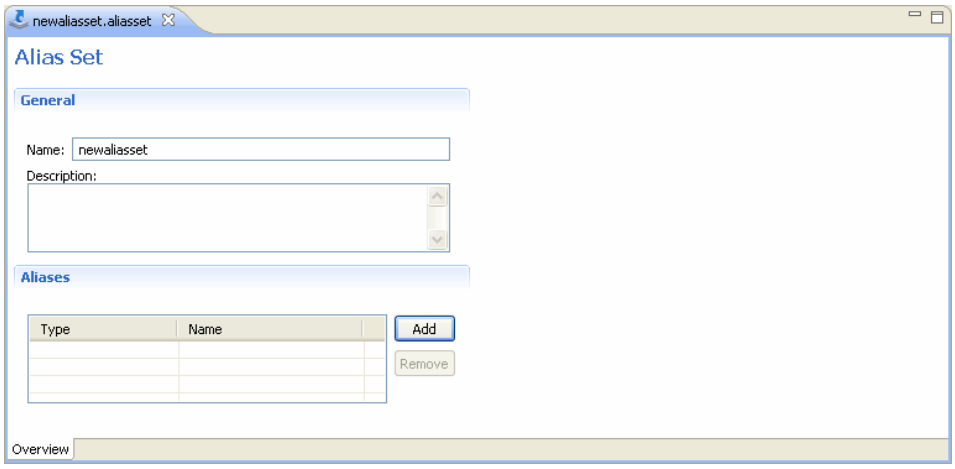
To create an alias set:

1. In your Documentum Composer project, expand the **Artifacts** folder and right-click **Alias Set**. Select **New > Alias Set**. The **New Alias Set** dialog appears.



2. Type 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.

3. Type a file name for the alias set in the **Artifact name** field, then click **Finish**. The **Alias Set** editor appears.



4. Type a name for the alias set in the **Name** field and an optional description in the **Description** field.

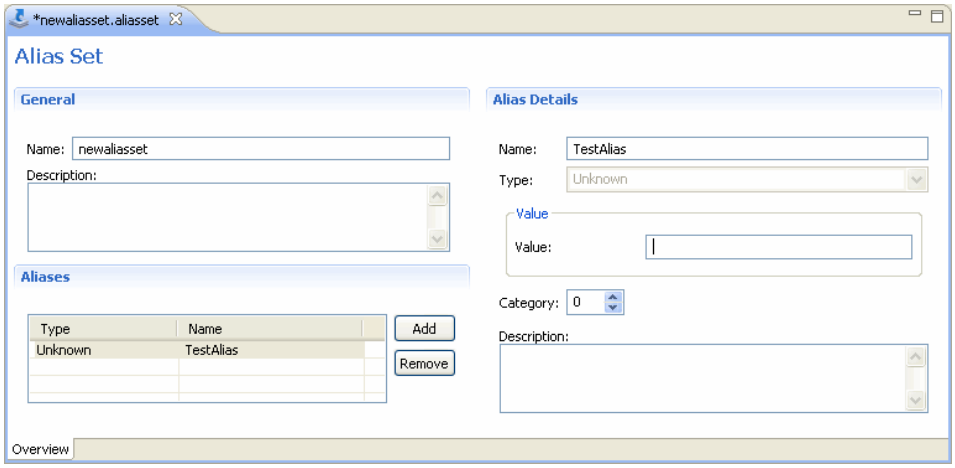
5. Click **Add** in the **Aliases** section to create one or more aliases that make up the alias set. The **New Alias** dialog appears.

6. Type 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 alias types:

   - Unknown
   - User
   - Group
   - User or Group
   - Cabinet Path
   - Cabinet or Folder Path
   - Permission Set

   "Configuring alias values" on page 78 describes the parameters associated with a specific alias type.

7. Click **OK** when you are finished. The **Alias Details** section appears.



8. Enter the details for the alias in the **Alias Details** section as described in the following table:

Properties	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. "Configuring alias values" on page 78 describes the parameters associated with a specific alias type.
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.

9. Save your changes.

6.2.1 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. The following table describes the parameters associated with a specific alias type.

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** and then click **Select**. The **User Installation Parameter** dialog appears. Select a parameter from the list box or click **New** to create a user parameter.
Group	• Leave It Blank • Parameter • Value	To assign a parameter: Click **Parameter** and then click **Select**. The **Group Installation Parameter** dialog appears. Select a parameter from the list box or click **New** to create a group parameter. To assign a value: Click **Value** and then click **Select**. The **Documentum Group Artifact** dialog appears. Select an artifact from the list box and then click **OK**.
User or Group	• Leave It Blank • Parameter	To assign a parameter: Click **Parameter** and then click **Select**. The **Principal (User or Group) Installation Parameter** dialog appears. Select a parameter from the list box or click **New** to create a parameter.

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

Chapter 7

Managing Aspects

7.1 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 following:

- The aspect type definition.
- The JAR files that contain the implementation classes and the interface classes for the behavior the aspect implements.
- Any interface classes on which the aspect depends.

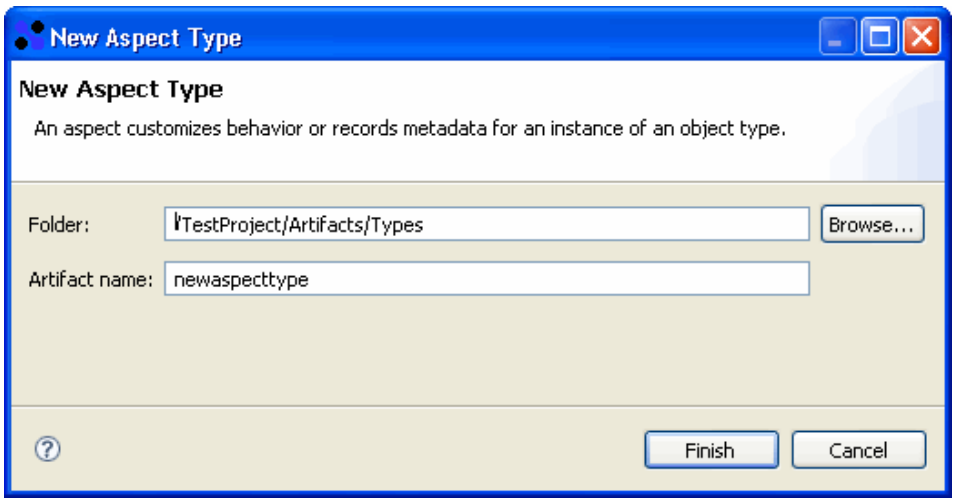
The module can also include Java libraries and documentation.

7.2 Creating an aspect type

Use the Aspect editor to create an aspect type.

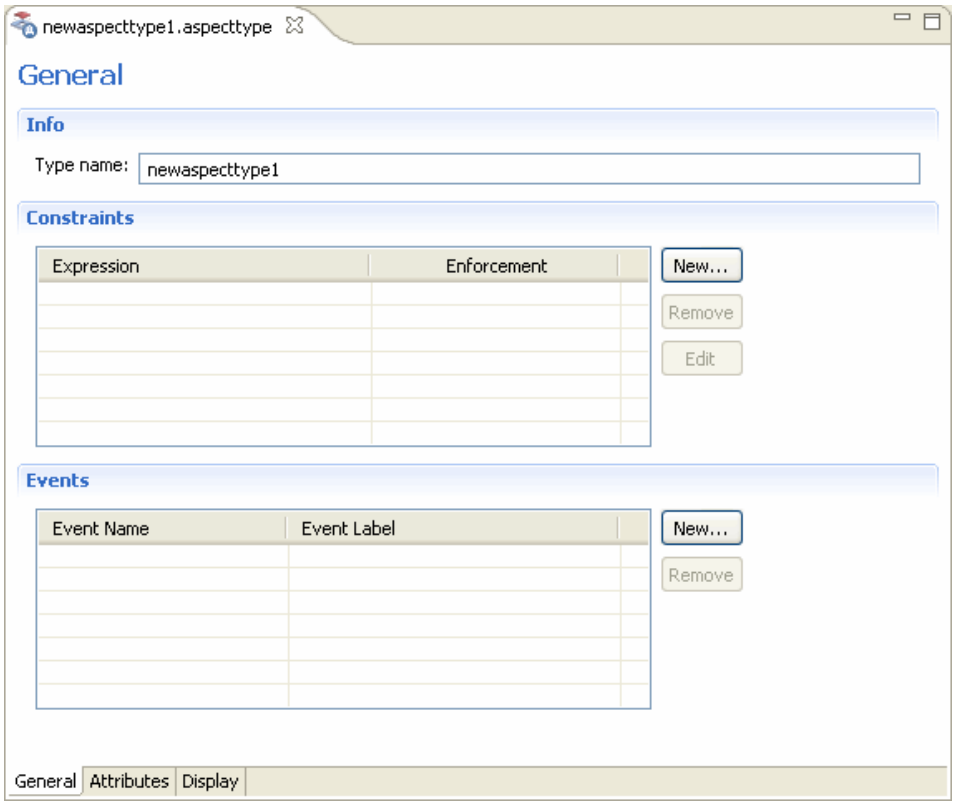
To create an aspect type:

1. In your Documentum Composer project, expand the **Artifacts** folder and right-click **Types**. Select **New > Aspect Type**. The **New Aspect Type** dialog appears.



2. 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.

3. Type a file name for the aspect type in the **Artifact name** field and then click **Next**. The **Aspect** editor appears with the **General** tab selected.



4. Enter the aspect information in the **Info**, **Constraints**, and **Events** sections, as described in the following table:

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 lowercase. The Documentum CM Server is case-insensitive and stores all type names in lowercase. • 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.

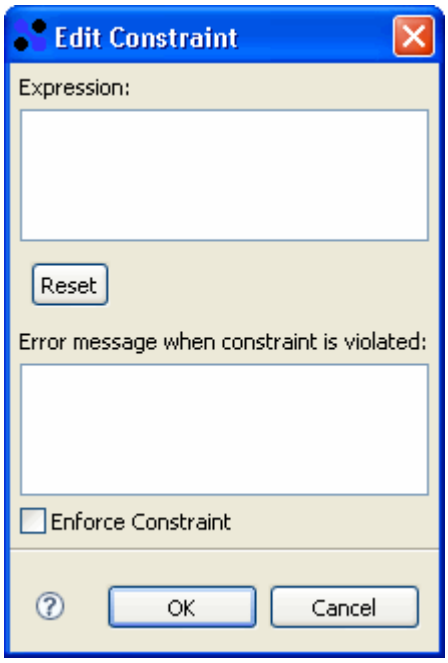
Property	Description
Expression	The Docbasic expression defining the constraint. Click **New** to create an expression. "Configuring constraint expressions" on page 83 provides information on how to add constraints.
Enforcement	Specifies whether applications enforce this constraint or not. Click the 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.

7.2.1 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 **New** in the **Constraints** section of the **General** tab in the Aspect editor. The **Edit Constraint** dialog appears.



2. In the **Expression** text box, type a valid Docbasic constraint expression that resolves to true or false. The Docbasic expression resolves to true when the constraint is fulfilled and false when the constraint is violated.

3. In the **Error message when constraint is violated** text box, type a message for applications to display when the constraint is violated.

4. Select the **Enforce Constraint** check box to instruct applications to enforce this constraint or clear the check box to not enforce the constraint.

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

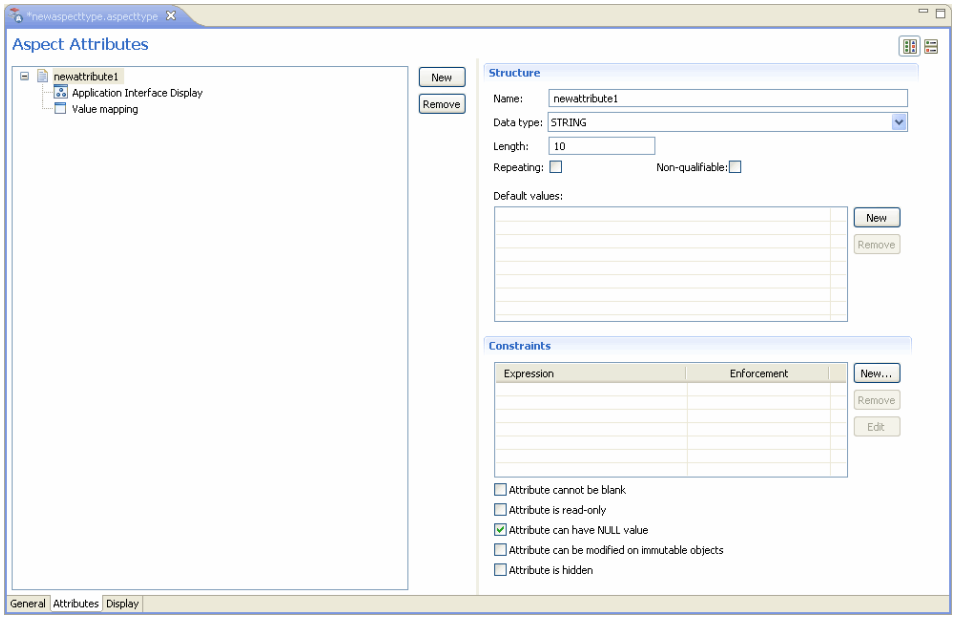
7.3 Adding aspect attributes

Aspect attributes are configured in the Attributes tab of the aspect editor. An aspect attribute is a property that applies to all aspects of that type. When an aspect is created, its attributes are set to values that describe that instance of the aspect type. You must configure attributes for each aspect. If you create an aspect without configuring any attributes, the aspect artifact does not install correctly 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 **Aspect Attributes** view.

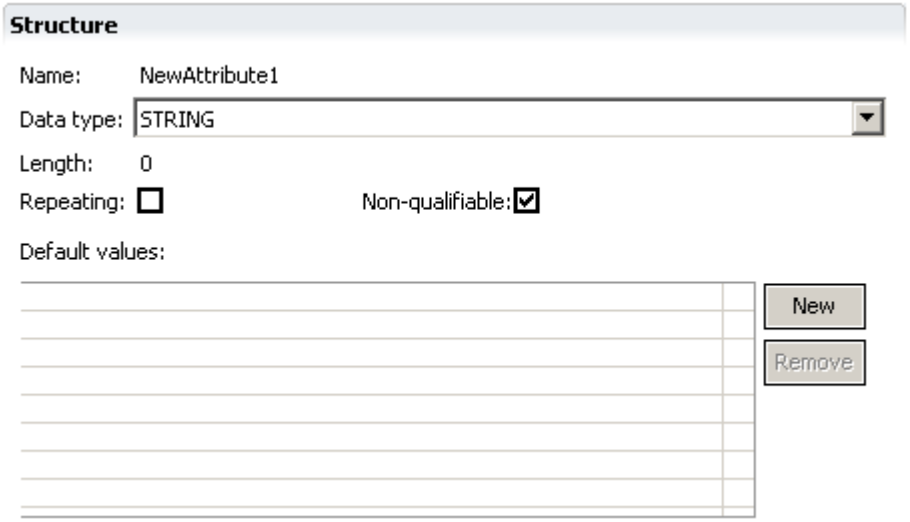
2. Click **New** to create an attribute entry, then select the new attribute entry. The **Aspect Attributes** view expands.



3. Configure the aspect attribute structure, as described in "Configuring the aspect attribute structure" on page 85.
4. Configure the aspect attribute constraints, as described in "Configuring the aspect attribute constraints" on page 87.

7.3.1 Configuring the aspect attribute structure

Configure the attribute structure in the **Structure** section of the **Aspect Attributes** view.

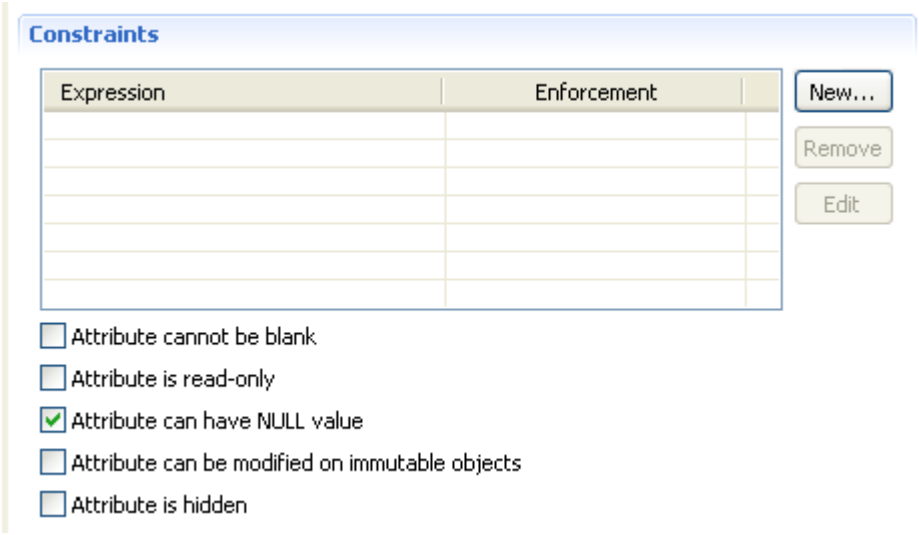


Enter the attribute structure properties, as described in the following table:

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 check box to allow more than one value for this attribute.
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.

7.3.2 Configuring the aspect attribute constraints

Configure the aspect attribute constraints in the **Structure** section of the **Aspect Attributes** view.



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

Enter or specify the aspect attribute constraint properties, as described in the following table:

Property	Description
Expression	The Docbasic expression that defines the constraint. Click **New** to create an expression. "Configuring constraint expressions" on page 83 provides information about how to create or modify an expression.
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.

Property	Description
Attribute cannot be blank	Select to specify that the aspect attribute must have a value. Users must enter a value for this aspect attribute when the application executes.
Attribute is read-only	Select to specify that the aspect attribute is read-only. Users cannot change the value of the aspect attribute when the application executes.
Attribute can have NULL value	Select to specify that the aspect attribute does not need an assigned value assigned. Users do not need to enter a value for this aspect attribute when the application executes.
Attribute can be modified on immutable objects	Select to enable users to modify the aspect attribute even though the object itself is immutable (unchangeable).

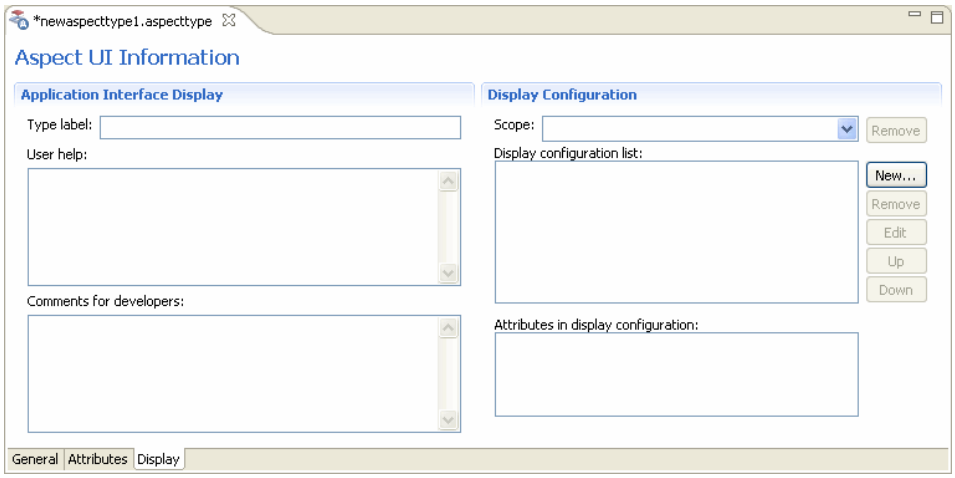
7.4 Configuring the Aspect UI information

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.

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.



2. Enter the Aspect UI information as described in the following table:

Property	Description
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.
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. "Adding a tab" on page 90 information about adding a tab to display an attribute in a client application. • 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.

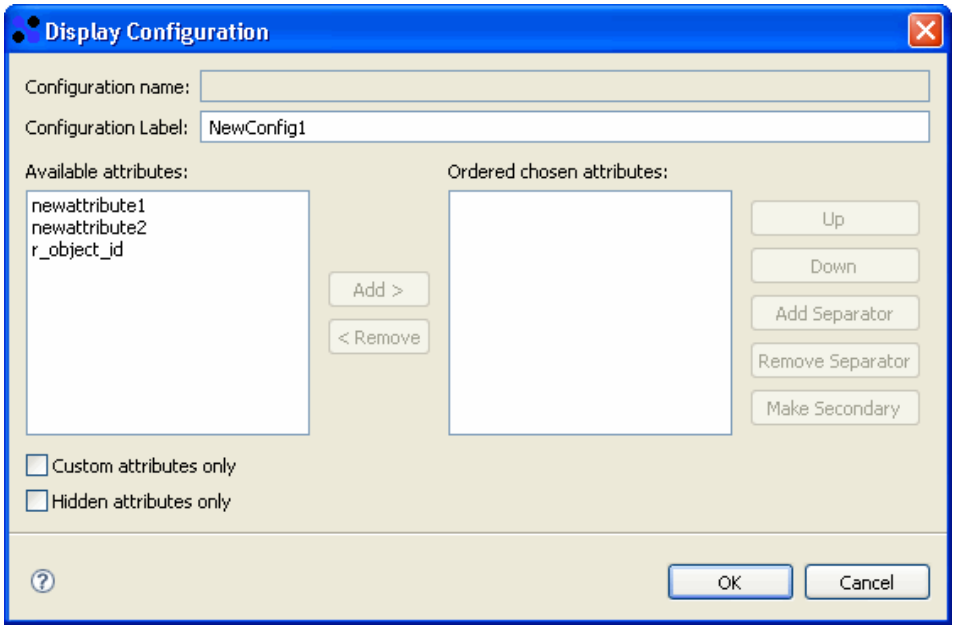
7.4.1 Adding a tab

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.

   The **Display Configuration** dialog appears.



2. Configure the tab properties, as described in the following table:

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.

Tab properties	Description
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. "Adding aspect attributes" on page 84 provides information about configuring attributes.
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.

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

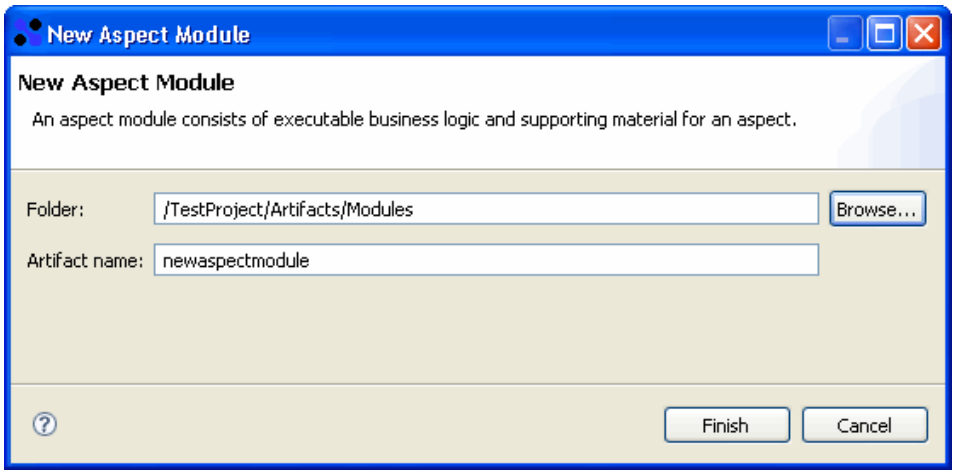
7.5 Creating an aspect module

Before you can create an aspect module, create a Documentum project. "Creating a project" on page 17 provides information on how to create a Documentum project.

To create an aspect module:

1. In your Documentum Composer project, expand the **Artifacts** folder and right-click **Modules**. Select **New > Aspect Module.**

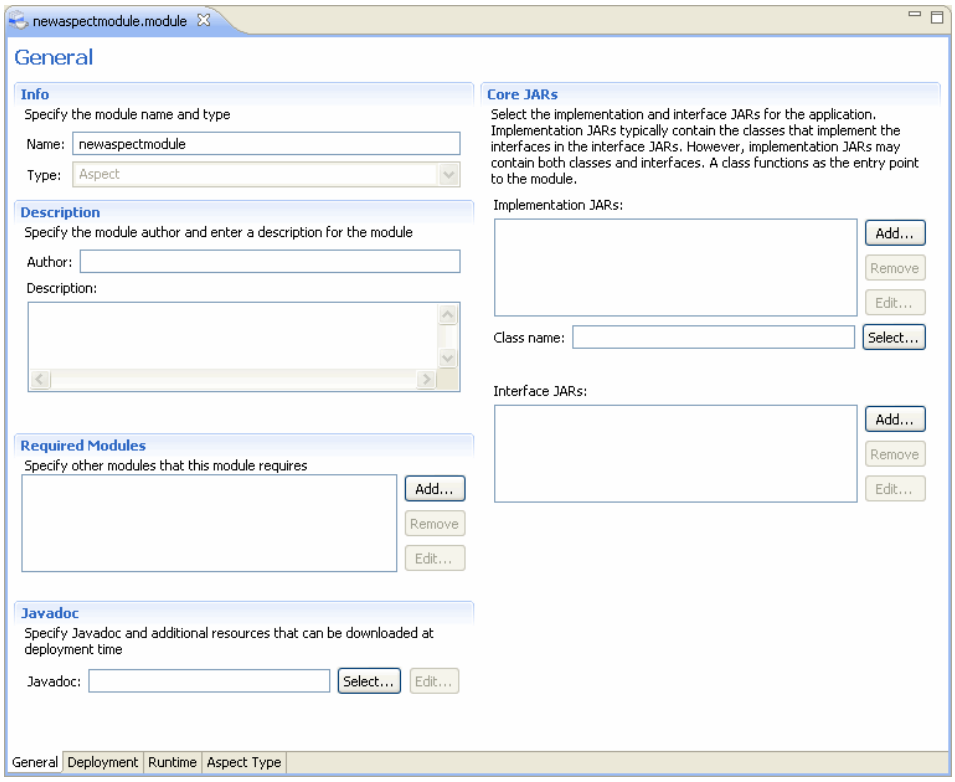
   The **New Aspect Module** dialog appears.



2. Enter a name for the new aspect module, then click **Finish**.

Note: 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.



3. Enter the required and optional properties in the **Info**, **Description**, **Required Modules**, **Javadoc**, and **Core JARs** sections, as described in the following table:

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 module.
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.

4. Click the *Deployment* tab to configure the module dependencies, as described in "Configuring aspect module deployment" on page 94.

5. Click the **Runtime** tab to configure the runtime environment for this module, as described in "Configuring the aspect module runtime environment" on page 96.

6. Click the **Aspect Type** tab to configure an aspect type for this module, as described in "Configuring the aspect type" on page 97.

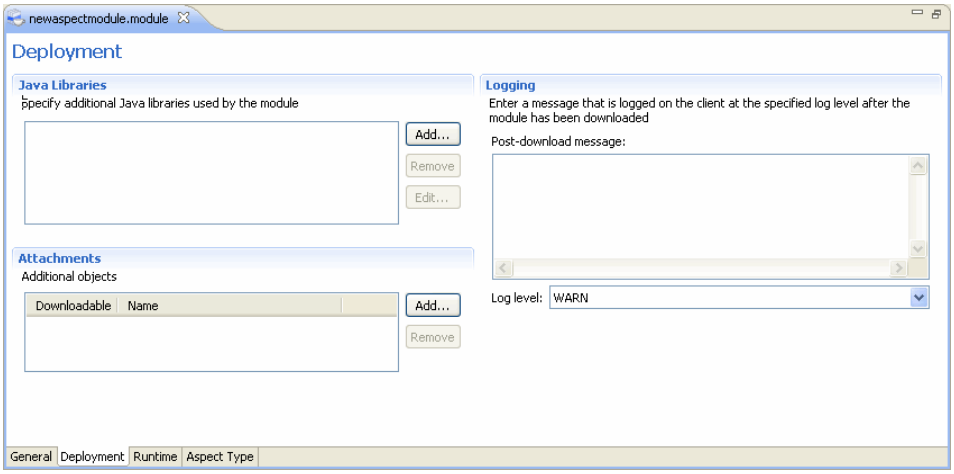
7.5.1 Configuring aspect module deployment

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

To configure module deployment:

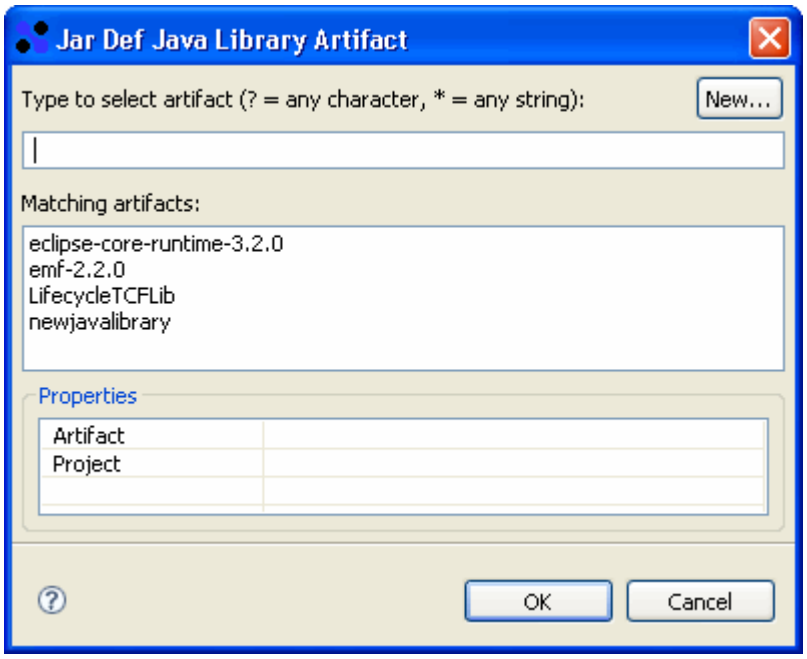
1. Click the **Deployment** tab in the Aspect Module editor.

   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 **Matching artifacts** listbox and click **OK** or click **New** to create a Java Library. You can only link existing Java libraries into this module. You cannot modify an existing library that is shared by multiple modules. "Linking and configuring a Java library" on page 107 provides information about how to create a Java library.

3. In the **Attachments** section, specify additional objects that are 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.

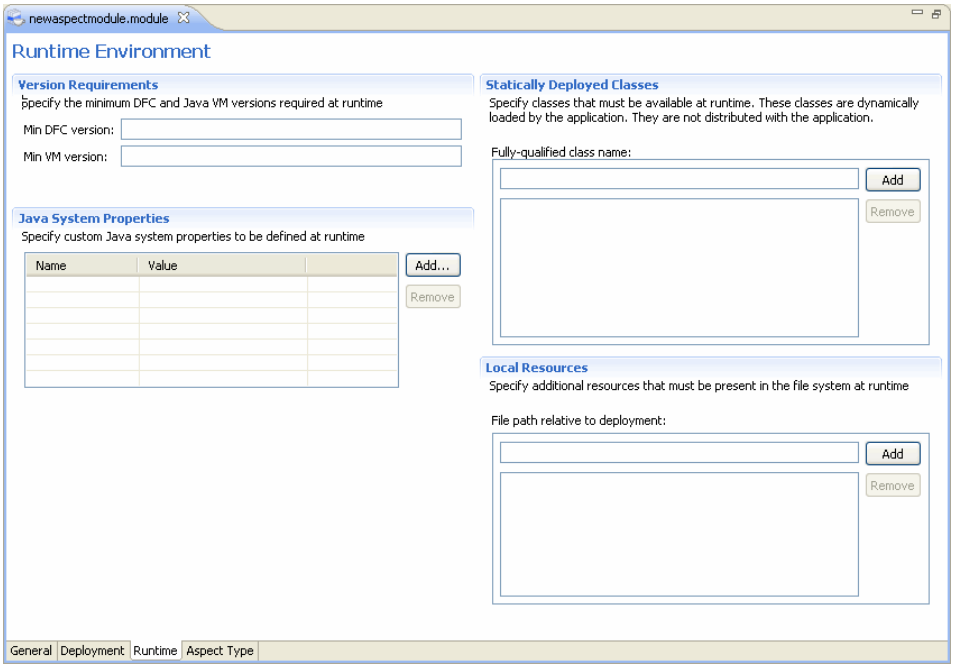
7.5.2 Configuring the aspect module runtime environment

The runtime environment lets you configure optional properties that are required in the runtime environment, 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 the following table:

Property	Description
Version Requirements	This section lets you specify the OpenText™ Documentum™ Content Management Foundation Java API and Java VM versions required on the client for the module to function properly.
Min OpenText Documentum Content Management (CM) Foundation Java API version	The minimum Foundation Java API 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.

Property	Description
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**.
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.

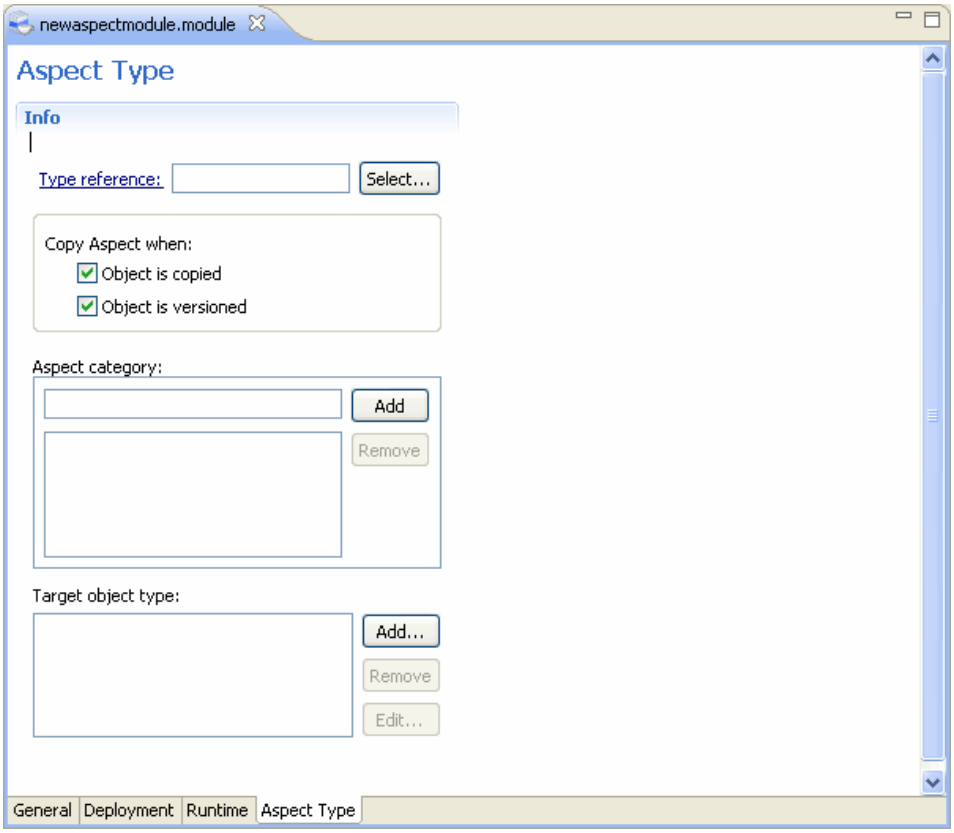
7.5.3 Configuring the aspect type

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 the following table:

Property	Description
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 an aspect type. "Creating an aspect type" on page 81 provides information about how to create an aspect type.
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.

Property	Description
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 type. "Object types" on page 177 provides information about how to create types.

Chapter 8

Managing Formats

8.1 Formats

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

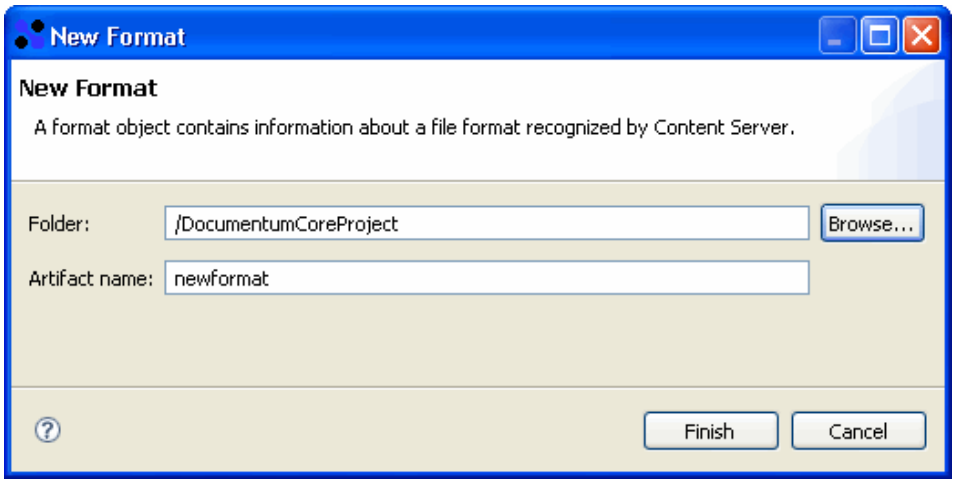
8.2 Creating a format artifact

Use the format editor to create a format artifact.

To create a format artifact:

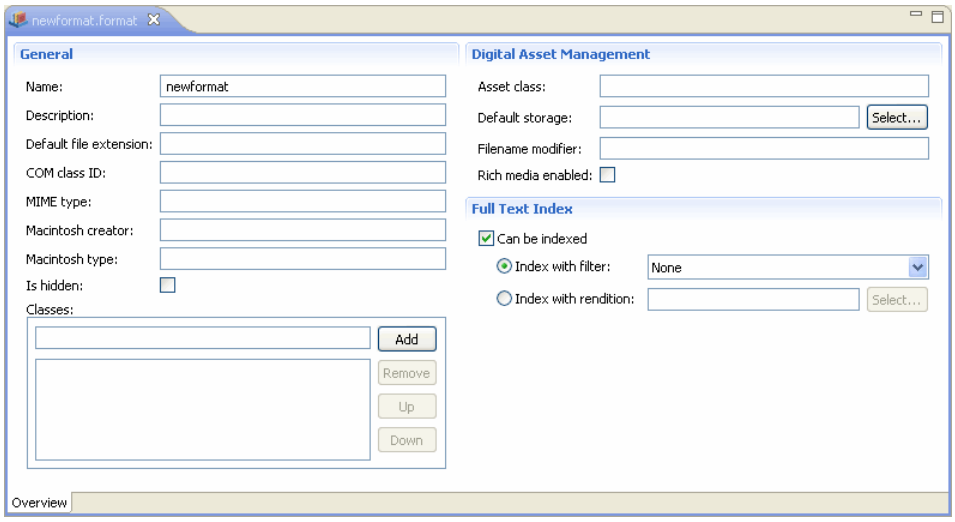
1. In your Documentum Composer project, expand the **Artifacts** folder and right-click **Formats**. Select **New > Format**.

   The **New Format** dialog appears.



2. 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.



3. Enter the format properties in the **General**, **Digital Asset Management**, and **Full Text Index** sections, as described in the following table:

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.

Parameter	Description
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	
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 Documentum CM 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

Parameter	Description
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 9

Managing JARs and Java Libraries

9.1 JAR definitions, JARs and Java libraries

A Java ARchive or JAR file is an archive file that aggregates many files into one. It is used to distribute Java classes and associated metadata, and can serve as a 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.

Documentum 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.

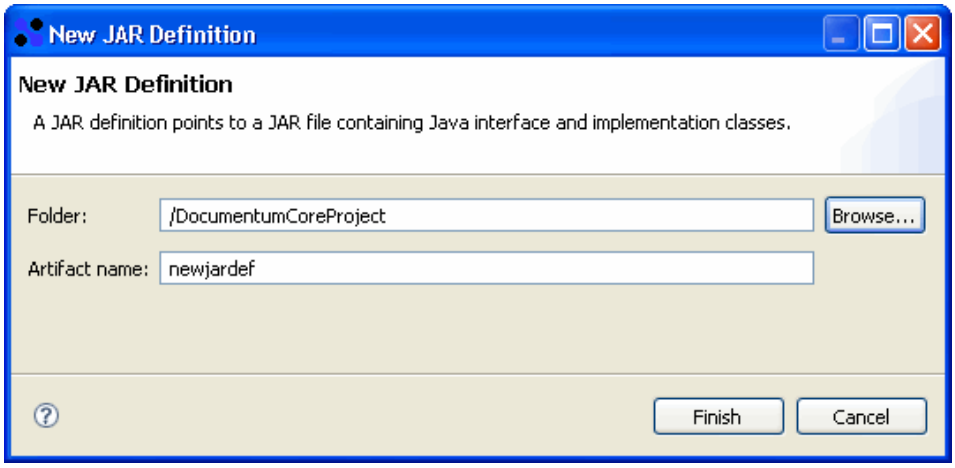
9.2 Creating a JAR definition

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

To create a JAR definition:

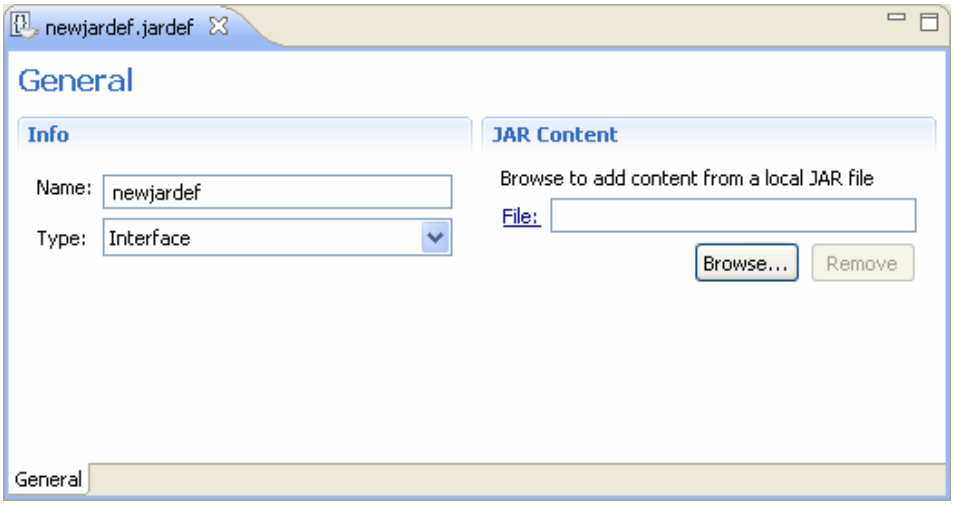
1. In your Documentum Composer project, expand the **Artifacts** folder and right-click **JAR Definitions**. Select **New > JAR Definition**.

   The **New Jar Definition** dialog appears.



2. 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.



3. Enter the JAR definition properties and add file content, as described in the following table:

Property	Description
Info	
Name	A string specifying the name of the 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.

4. Save your changes.

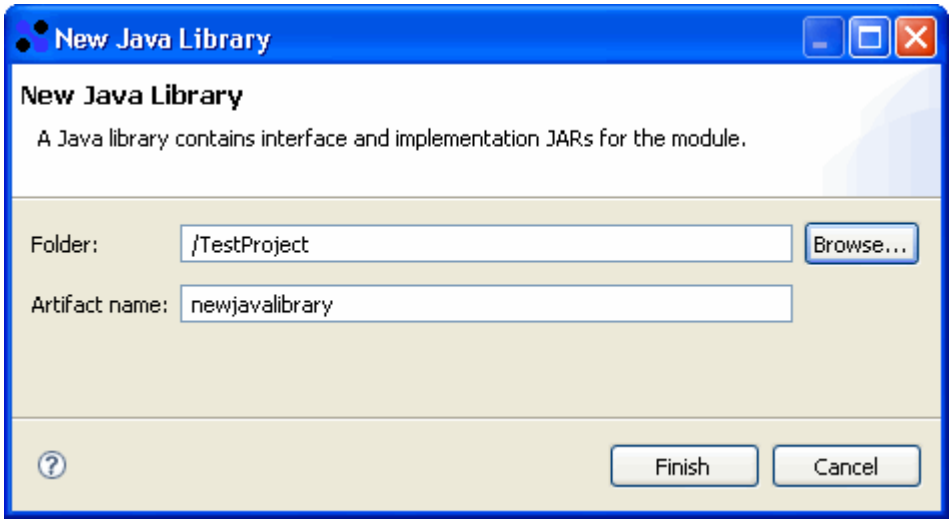
9.3 Linking and configuring a Java library

Documentum 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 from the module editor or from the Documentum Composer main menu.

To link a Java library:

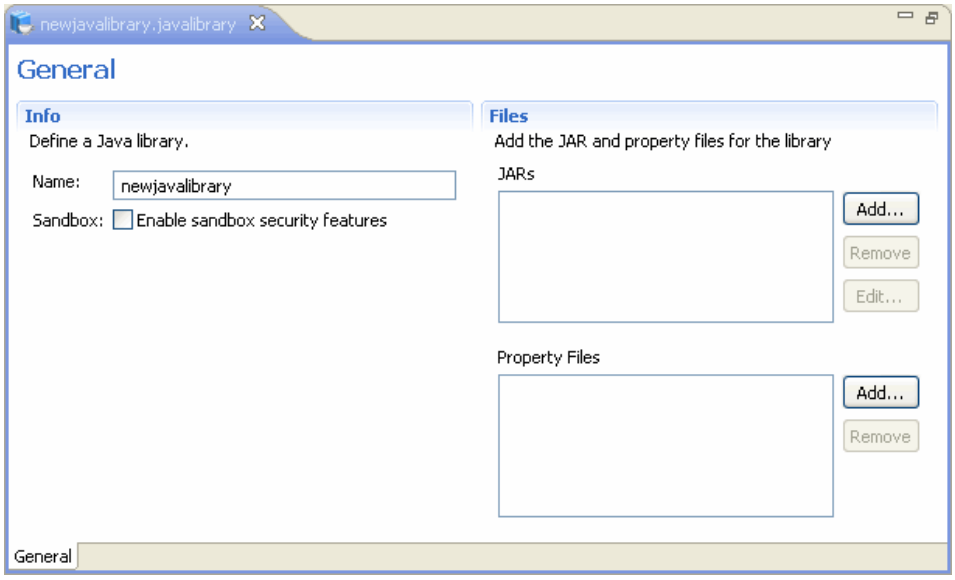
1. In your Documentum Composer project, expand the **Artifacts** folder and right-click **Java Libraries**. Select **New > Java Library**.

   The **New Java Library** dialog appears.



2. 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.



3. Configure the Java library, as described in the following table:

Property	Description
Info	
Name	A string specifying the name of the Java library.
Sandbox	Specifies whether this Java library uses a sandbox. If selected, the Java library uses a sandbox. By default, the sandbox is disabled. **Note:** If you do not sandbox your Java libraries, they will not be deployable.
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.

4. Save your changes.

Chapter 10

Managing Lifecycles

10.1 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 state, or a finalized state. A lifecycle does not define what activities happen to the document while it resides in a state.

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.

10.1.1 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, Documentum Composer provides the dm_sysobject and its subtypes in the lifecycle editor.

You can also create an object type. "Object types" on page 177 provides more information about object types, and "Creating a standard object type" on page 178 provides information on how to create an object type.

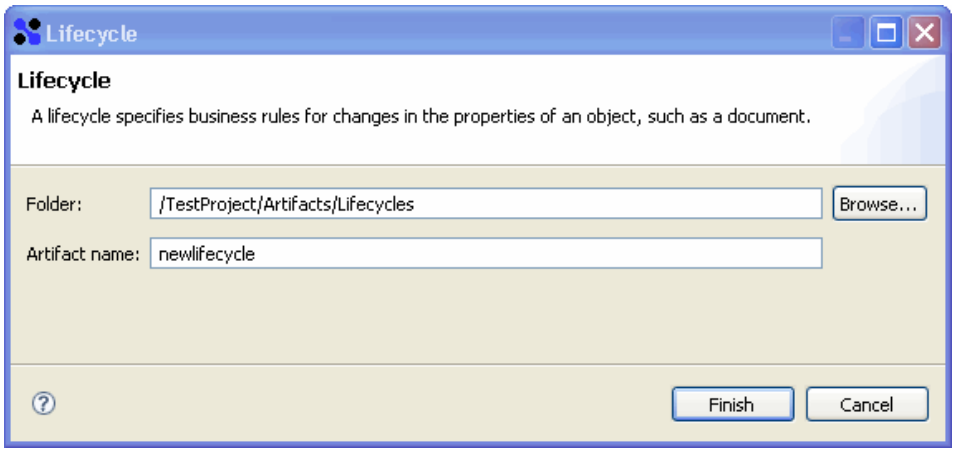
10.2 Creating a lifecycle

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

To create a lifecycle:

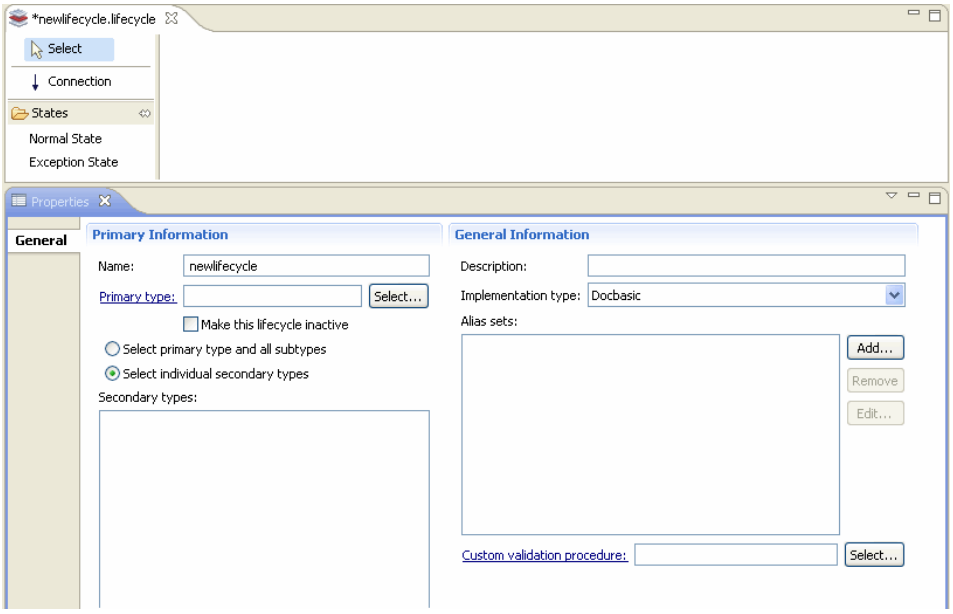
1. In your Documentum Composer project, expand the **Artifacts** folder and right-click **Lifecycles**. Select **New > Lifecycle**.

   The **Lifecycle** dialog appears.



2. Enter the full path name of the folder in which you want to create the lifecycle in the **Folder** field or click **Browse** to select a folder from drop-down list.
3. Enter the name of the lifecycle in the **Artifact name** field or accept the default name.
4. Click **Finish**.

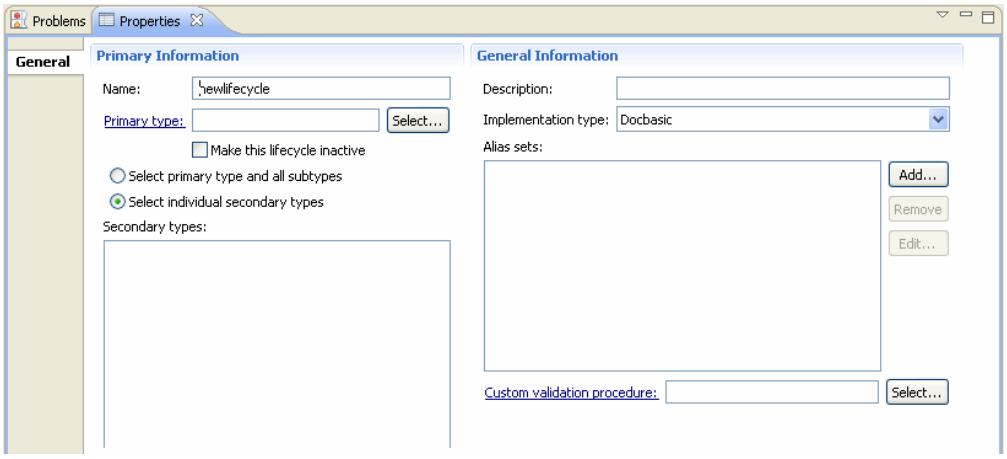
   The **Lifecycle** editor appears.



5. Configure the properties of the lifecycle, as described in "Configuring lifecycle properties" on page 111.
6. Configure lifecycle states, as described in "Adding and configuring lifecycle states" on page 113.
7. Save your changes.

10.3 Configuring lifecycle properties

Configure the properties of a lifecycle on the **General** tab of the **Lifecycle editor - Properties** page.



Lifecycle properties include the primary object type, secondary object types, a validation procedure, implementation type, and alias sets. In the **Primary**

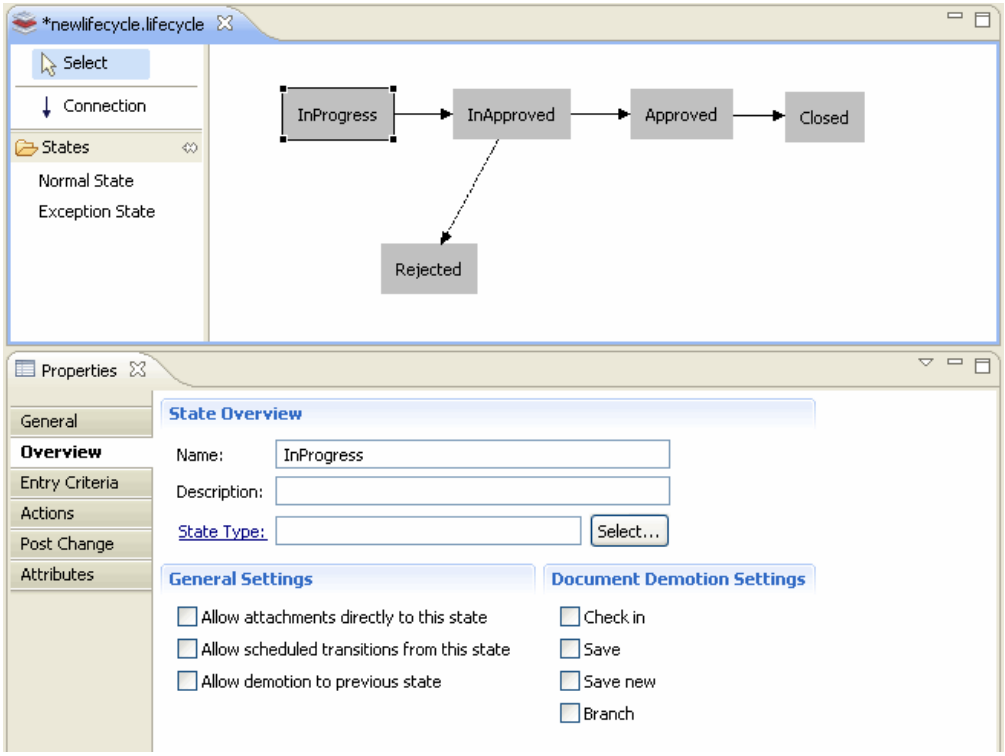
Information and **General Information** sections, configure the properties of a lifecycle as described in the following table:

Parameter	Description
Primary Information	
Name	A string specifying the name of the lifecycle.
Primary type	Assign a primary object type by 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 an object type. The **New Object Type Artifact** wizard appears. "Creating a standard object type" on page 178 provides more information about how to create a custom object type.
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 list box.
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

Parameter	Description
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. "Alias, alias values, and alias sets" on page 75 provides additional information about 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.

10.4 Adding and configuring lifecycle states

After you create the lifecycle, you can add states. Add lifecycle states in the main window of the lifecycle editor in the form of a state diagram. Each state appears as a rectangle and the arrows designate transitions from one state to the next as shown in the following graphic:



The lifecycle states are 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 the following table:

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 state type. "Creating a state type" on page 116 provides more information about how to create state types.
General Settings	
Allow attachments directly to this state	If this setting is selected, 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 selected, 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 selected, 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 selected, 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 selected, 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 selected, 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.

Properties	Description
Branch	If this setting is selected, a document is automatically returned to the base state when the document is branched.

10.4.1 Creating a state type

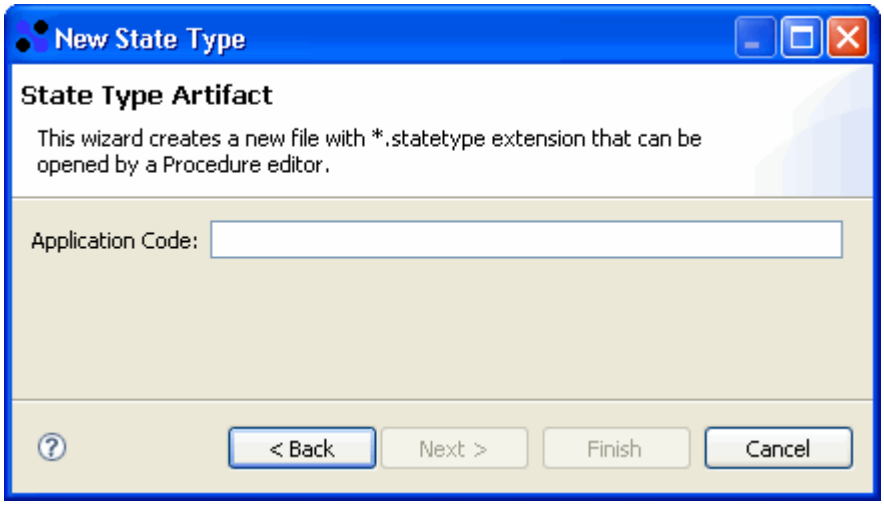
A state type identifies a state that a client application uses. Certain client applications identify a state by type instead of the name. You can create a state type artifact from the lifecycle state editor.

To create a state type:

1. In the **Lifecycle** editor, select the lifecycle state for which you want to create a state type.
2. Click the **Overview** tab on the **Properties** pane to display the **State Overview** page.
3. In the **State Overview** section, click the **State Type** link.

   The **New State Type** dialog appears.
4. Enter the full path name of the folder in which you want to create the state type in the **Folder** field or click **Browse** to select a folder from drop-down list.
5. Enter a name for the state type in the **Artifact name** field, or accept the default name. The state type name must be unique.
6. Click **Next**.

   The **State Type Artifact** dialog appears.



7. 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.

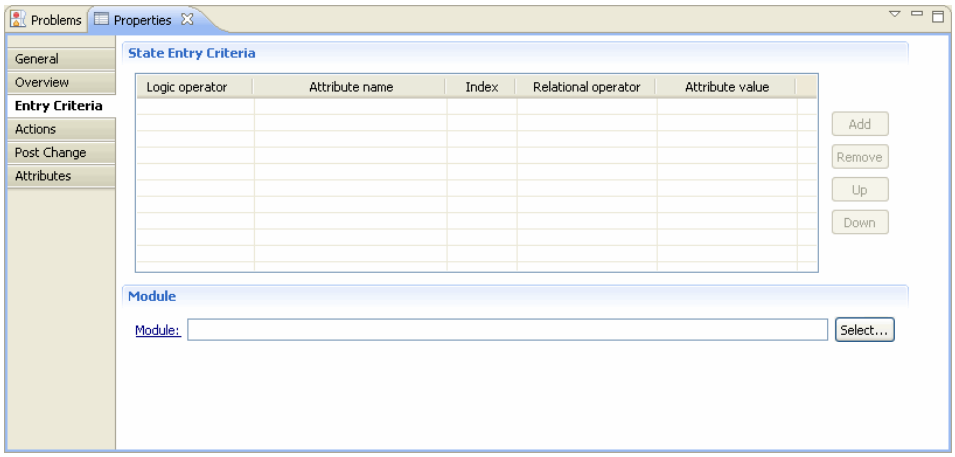
8. Click **Finish**.

10.5 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.



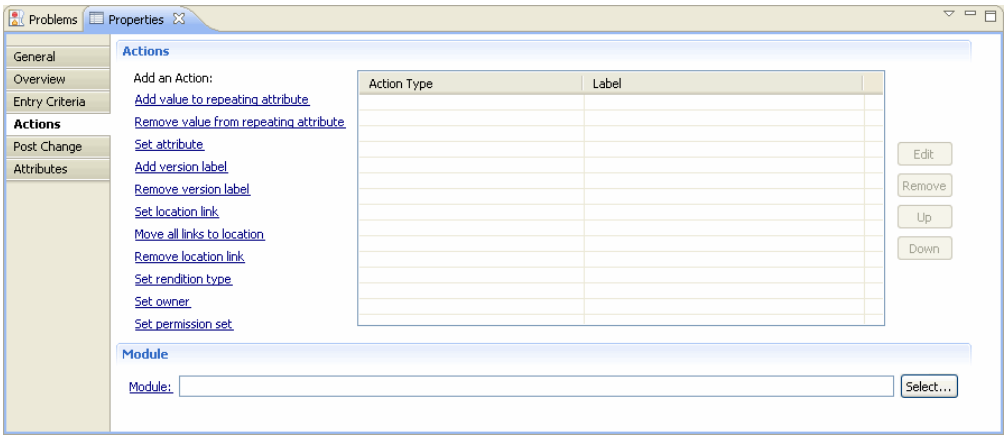
2. In the **State Entry Criteria** section click **Add** to add one or more state entry criteria rules. Click any field in a row to type or edit a value, as described in the following table:

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 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.

Property	Description
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.
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 procedure. • Click the **Procedure:** link to create a procedure. "Creating a procedure" on page 159 provides more information about how to create a procedure.

10.6 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.



10.6.1 Adding repeating attribute values

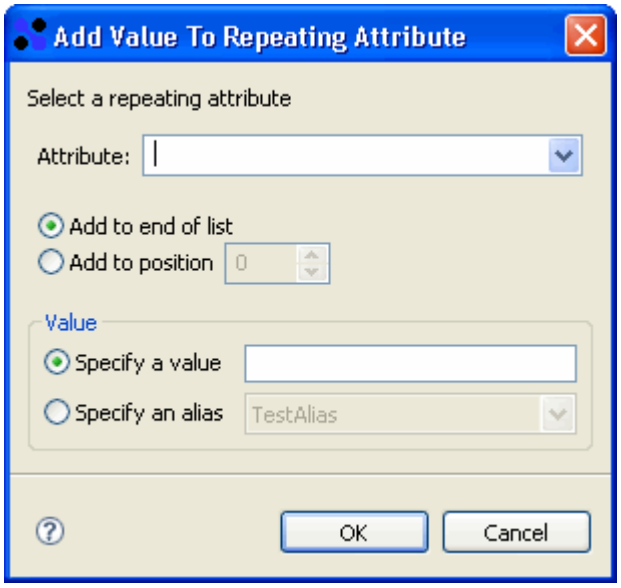
Repeating attributes are attributes that can have more than one value. For example, the document object's authors attribute is a repeating attribute since a document can have more than one author.

To add repeating attribute values:

1. In the lifecycle state diagram, click the state for which you want to add repeating attribute values.
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 the following table:

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	Select to add the value to the end of the existing list of values.

Property	Description
Add to position	Select to add the value at a specified position and replace the old value. Enter a specified position.
Value	The value of the repeating attribute.
Specify a value	Select this option to specify the actual value. Next, enter the value to store.
Specify an alias	Select to resolve the value from an alias at runtime. If the drop-down list does not show any aliases, no aliases have been configured for the project yet. "Alias, alias values, and alias sets" on page 75 provides information about managing alias sets.

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

10.6.2 Removing repeating attributes values

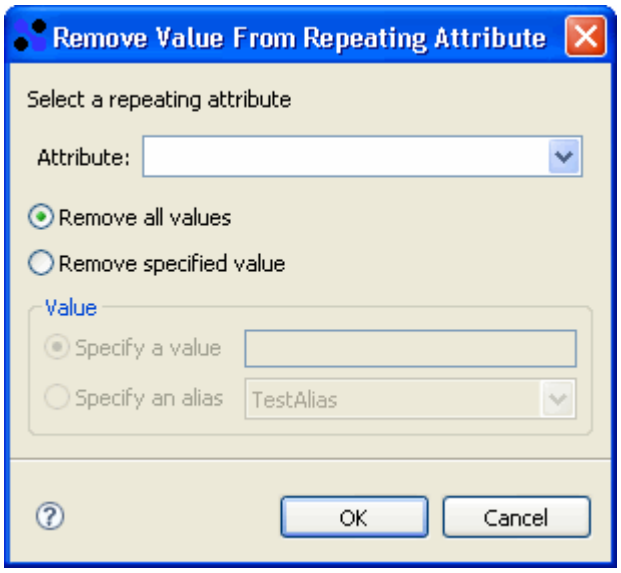
Repeating attributes are attributes that can have more than one value. For example, the document object's authors attribute is a repeating attribute since a document can have more than one author.

To remove one or more values from a repeating attribute:

1. In the lifecycle state diagram, click 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 the following table:

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 to specify an alias that is resolved at runtime that you wish to delete. Next, select the alias from the drop-down list. The alias is deleted from the values resolved at runtime.

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

10.6.3 Setting attributes

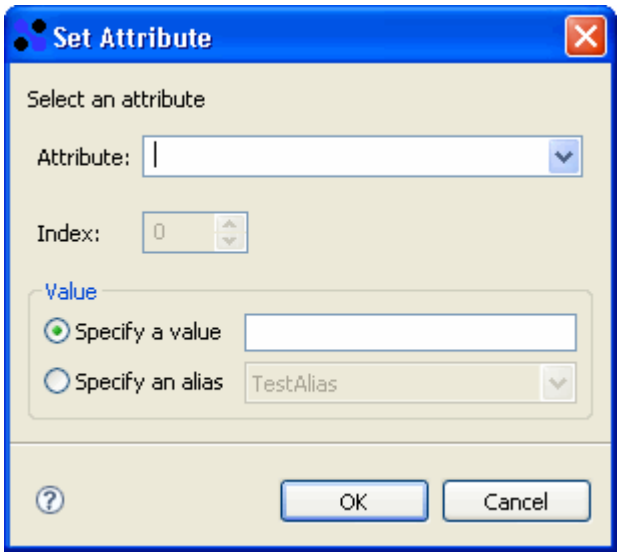
The Set Attribute action lets you specify attributes for a state, such as the title or version of a document.

To set an attribute for a lifecycle state:

1. In the lifecycle state diagram, click 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 the following table:

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.

Property	Description
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 to resolve the value from an alias at runtime and 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. "Alias, alias values, and alias sets" on page 75 provides information about managing alias sets.

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

10.6.4 Adding version labels

You can specify a version label to add to a document when it enters a particular state.

To specify a version label:

1. In the lifecycle state diagram, click the state for which you want to specify a version label for documents that enter this state.
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.

10.6.5 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 the state for which you want to remove a version label for documents entering this state.
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.

10.6.6 Setting location links

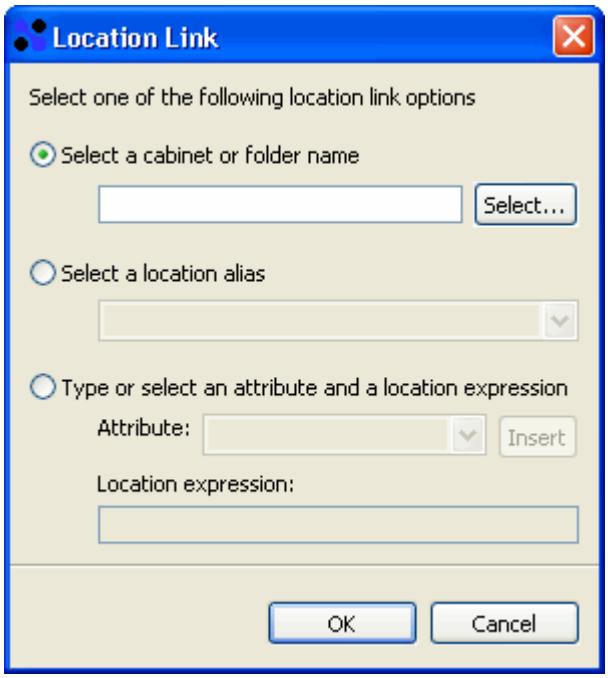
Location links let you link a document to a specific location, such as a folder, a cabinet, an alias, or a local expression. The link is created when the document enters the state.

To link a document to a specific location:

1. In the lifecycle state diagram, click 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 the following table:

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.

10.6.7 Moving all links

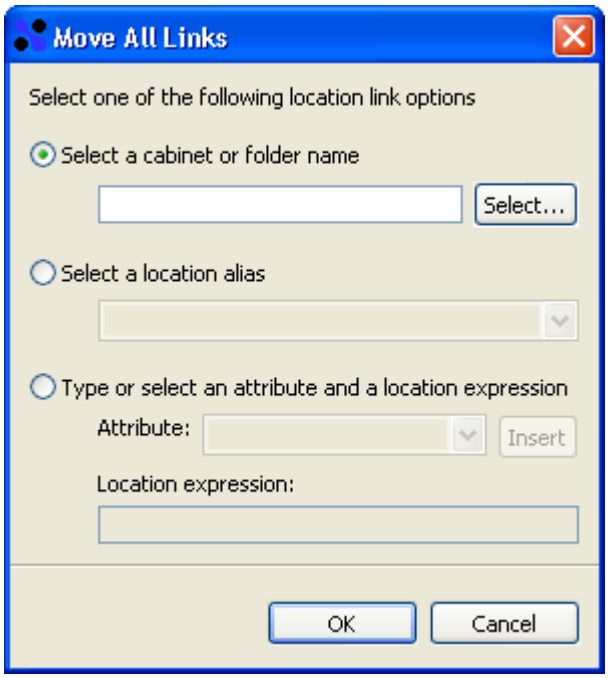
The Set location to move all links link lets you move all links to a specific location.

To move all location links:

1. In the lifecycle state diagram, click 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 **Move all links to location**.

   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 the following table:

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.

10.6.8 Removing location links

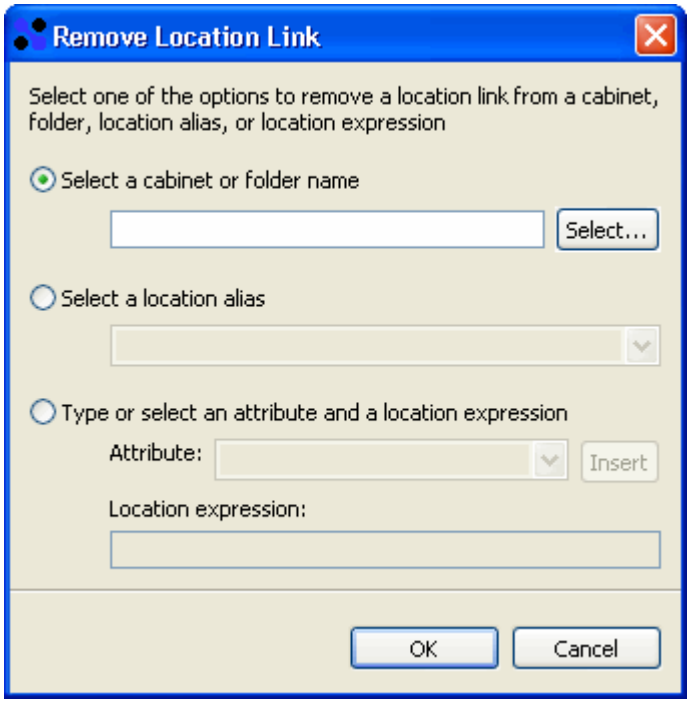
Use the **Remove Location Link** dialog to remove location links.

To remove location links:

1. In the lifecycle state diagram, click 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 the following table:

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.

Property	Description
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.

10.6.9 Assigning a document renderer

Documentum 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 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 Documentum Composer on a machine with a Windows operating system.
   - **Auto Render Pro for Macintosh**, if you are running Documentum Composer on a machine with a Macintosh operating system.
5. Click **OK** when you are finished.

10.6.10 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. Superusers can assign another user as the owner. To assign a group as the owner of an object, you must be a superuser or own the document and be a member of the group to which you are assigning ownership.

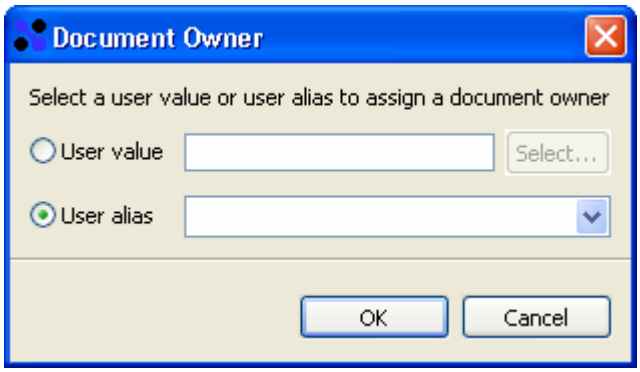
Note: You must have an alias set assigned to the lifecycle to assign an owner to a document. You can assign an alias set in the **General** tab of the lifecycle properties page. "Configuring lifecycle properties" on page 111 provides information on assigning alias sets to a lifecycle.

To assign an owner to a document:

1. In the lifecycle state diagram, click 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 following table:

Property	Description
User value	Select this option to 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 user value. An install parameter is resolved when the application is installed by the installing application and parameters supplied by the installing user.

Property	Description
User alias	Use this option to assign a user alias. Select an alias from the drop-down list. An alias is resolved at runtime by the Documentum CM Server.

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

10.6.11 Setting permission sets

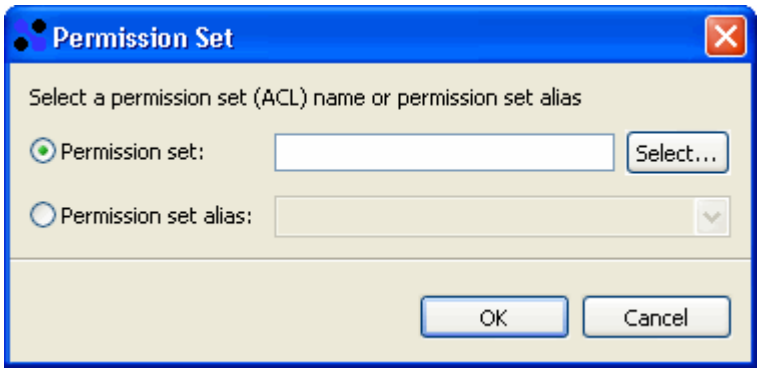
Permission sets (also known as ACLs, or access control lists) specify the operations (such as read, edit, create a version, or delete) users can perform on a document attached to a lifecycle.

To assign a permission set to a lifecycle state:

1. In the lifecycle state diagram, click 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 the following table:

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 permission set. "Permissions, permission sets, and permission set templates" on page 149 provides information about permission sets.

Property	Description
Permission set alias	Select this option to assign a permission set alias and select a permission set alias from the drop-down list. "Permissions, permission sets, and permission set templates" on page 149 provides information about permission sets.

10.7 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 the state to which you want to assign a post-change procedure.
2. Click the **Post Change** tab in the **Properties** pane.

   The **Module** page appears.
3. Click **Select**.

   The **Module Artifact** dialog appears.
4. Select a post-change procedure from the list or click **New** to create a post-change procedure.

   "Creating a procedure" on page 159 provides information on how to create a procedure.

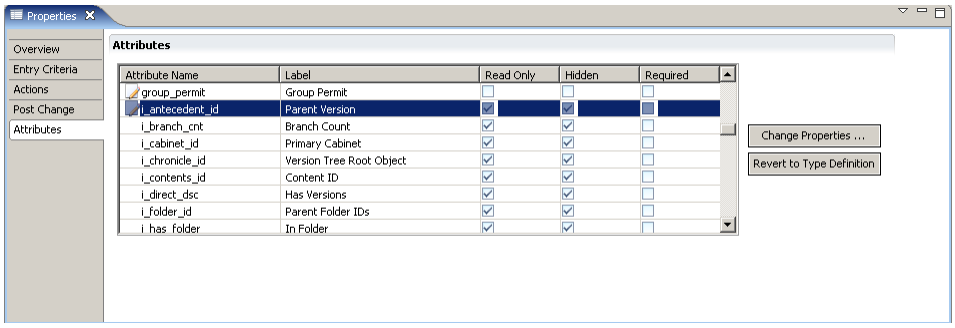
10.8 Configuring state attributes

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

To configure state attributes:

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

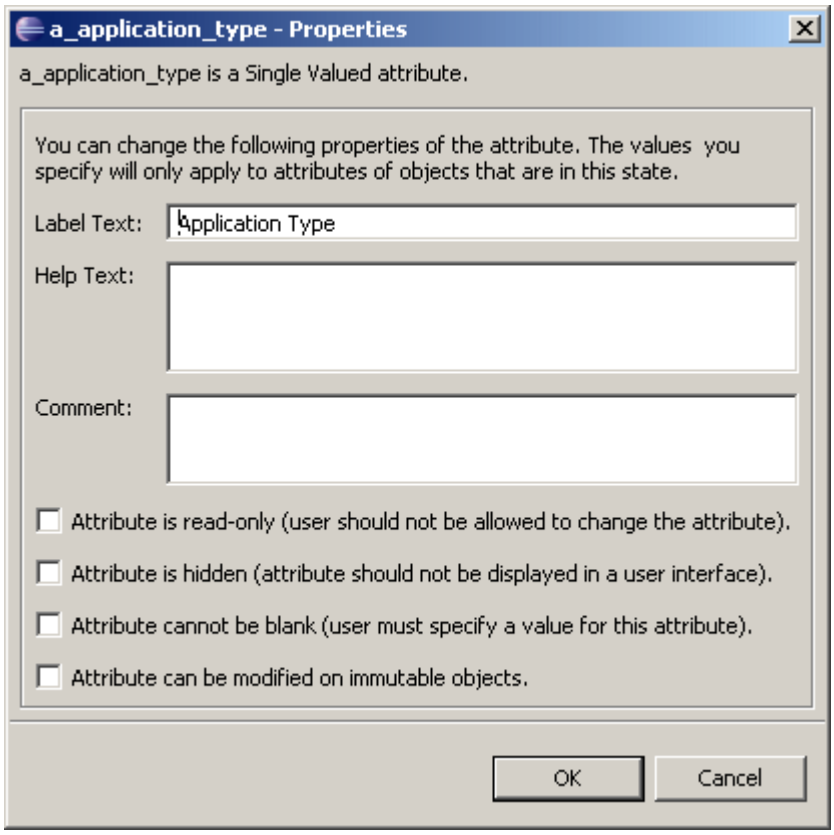
   The **Attributes** page appears.



The page displays only 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** on the **Attributes** page.

10.9 Deleting a lifecycle state

To delete a lifecycle state:

1. In the lifecycle state diagram, select the state to delete.
2. Right-click the state and select **Delete** from the drop-down menu.

10.10 Deleting a lifecycle

To delete a lifecycle:

1. In your Documentum Composer project in the **Documentum Navigator** view, expand the **Artifacts** folder.
2. Right-click **Lifecycles** and then select **Delete** from the drop-down menu.

   The **Delete Resources** dialog appears.
3. Click **OK** to delete the lifecycle.

Chapter 11

Managing Methods and Jobs

11.1 Methods and jobs

Methods are executable programs 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.

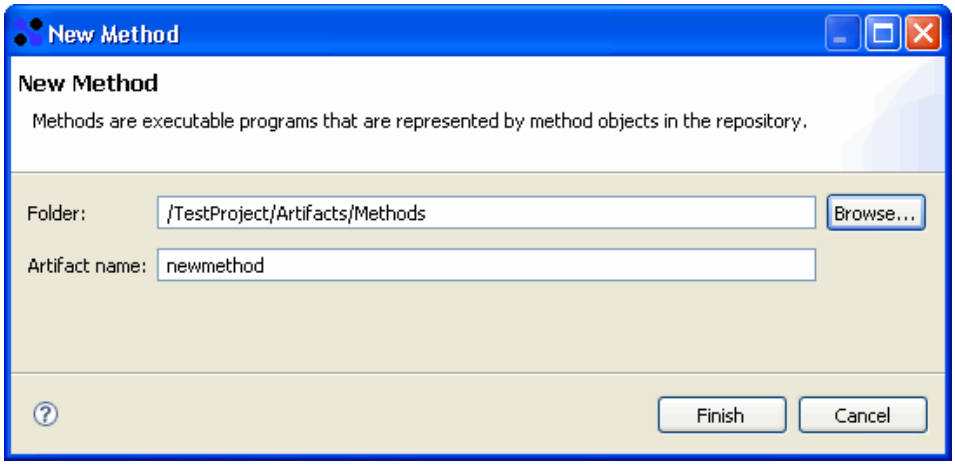
11.2 Creating a method

Use the Method editor to create a method.

To create a method:

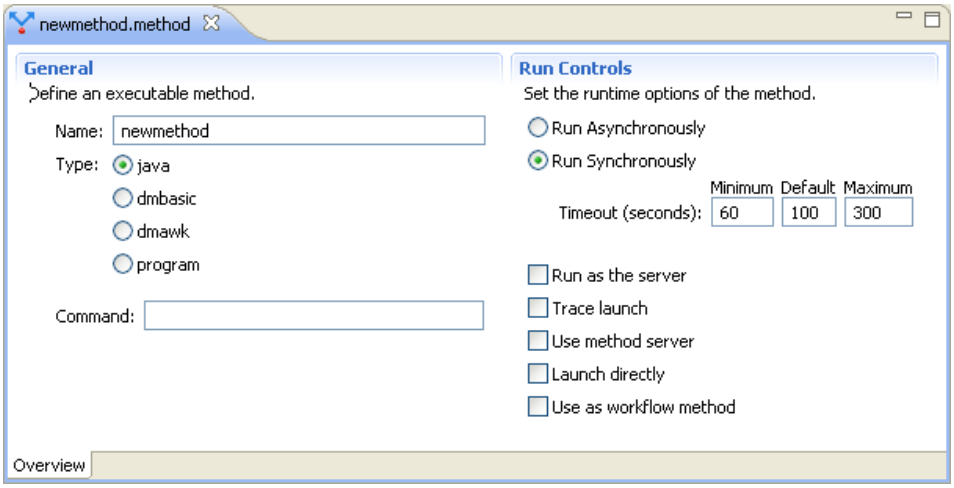
1. In your Documentum Composer project, expand the **Artifacts** folder and right-click **Methods**. Select **New > Method**.

   The **New Method** dialog appears.



2. Enter a name for the new method or accept the default name and then click **Finish**.

   The **Method** editor appears.



3. Enter the properties for the method, as described in the following table:

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 on of the following types: • **java**: The method is written in Java and the Java method server is executing the method. • **dmbasic**: 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 Documentum CM Server and SAVE_RESULTS is set to TRUE on the command line.
Run Synchronously	Specifies whether the method is run synchronously.

Parameter	Description
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.
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 Documentum CM 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 check box 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.

4. Save your changes.

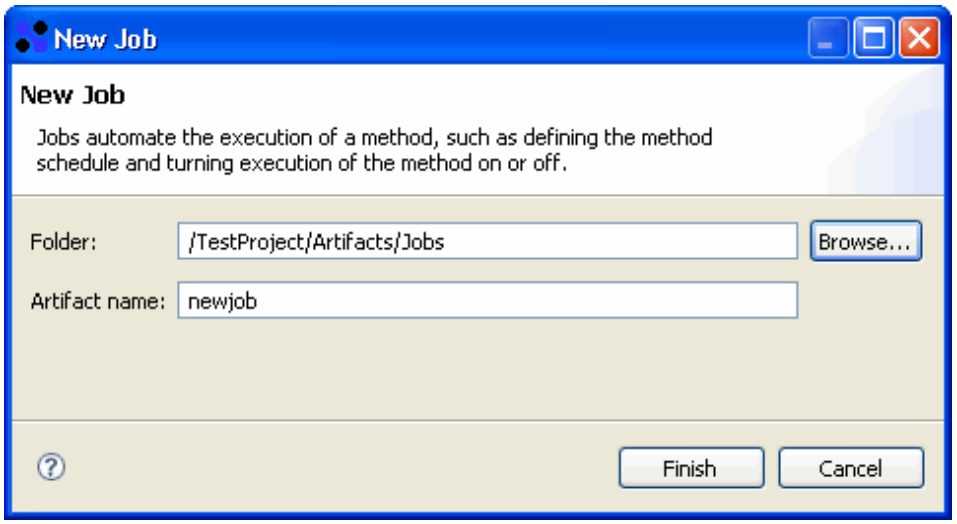
11.3 Creating a job

A job automates the execution of a method.

To create a job:

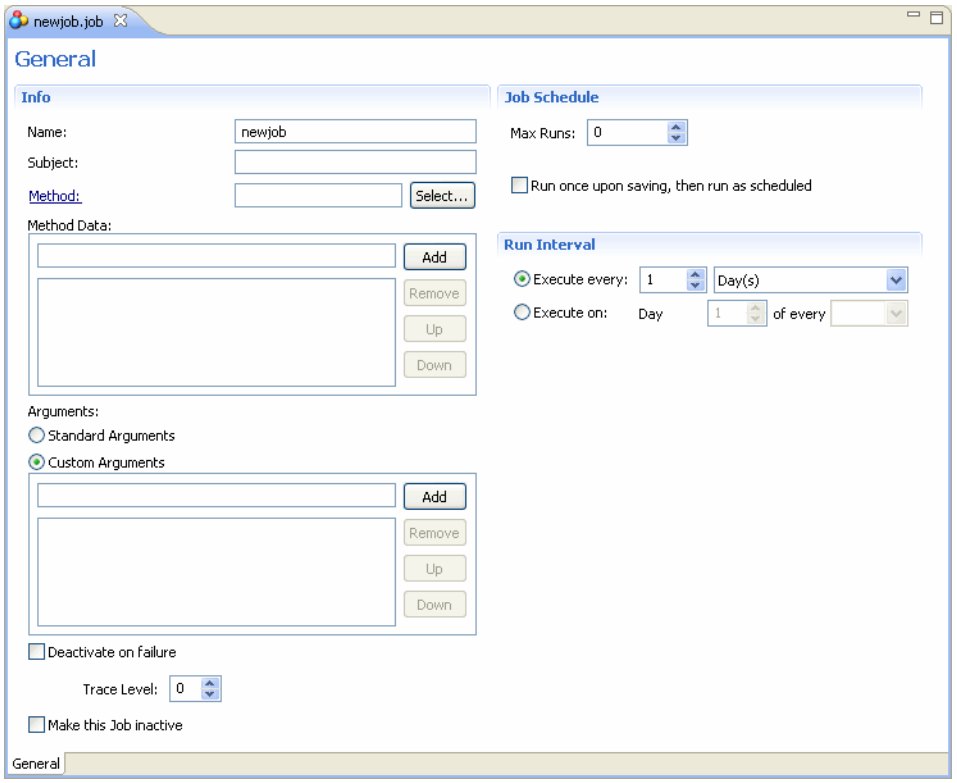
1. In your Documentum Composer project, expand the **Artifacts** folder and right-click **Jobs**. Select **New > Job**.

   The **New Job** dialog appears.



2. Enter a name for the new job or accept the default name, then click **Finish**.

   The **Job** editor appears.



3. Enter the properties for the job in the Info, Job Schedule, and Run Interval sections, as described in the following table:

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 method for this job.
Method Data	Data that is used by the method associated with the job. This property is available for the method to read and write 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
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 Inactive	Select to inactivate the job.
Job Schedule	
Max Runs	Specifies the maximum number of times the job runs.
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.

Parameter	Description
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.

4. Save your changes.

Chapter 12

Managing Modules

12.1 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 can also include Java libraries and documentation.

There are three types of modules:

- 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 the inbox of a user.

- 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, keywords, and subject 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" on page 81.

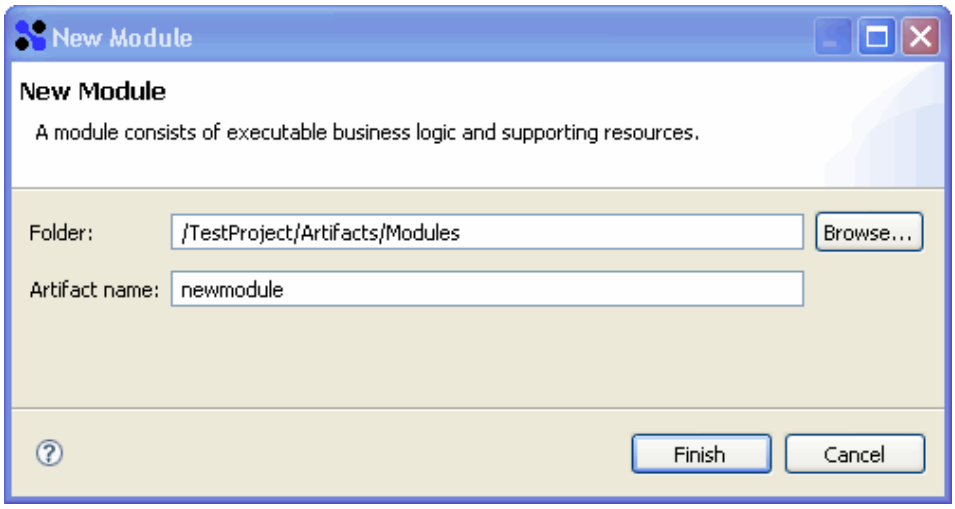
12.2 Creating a module

Use the Module editor to create a module.

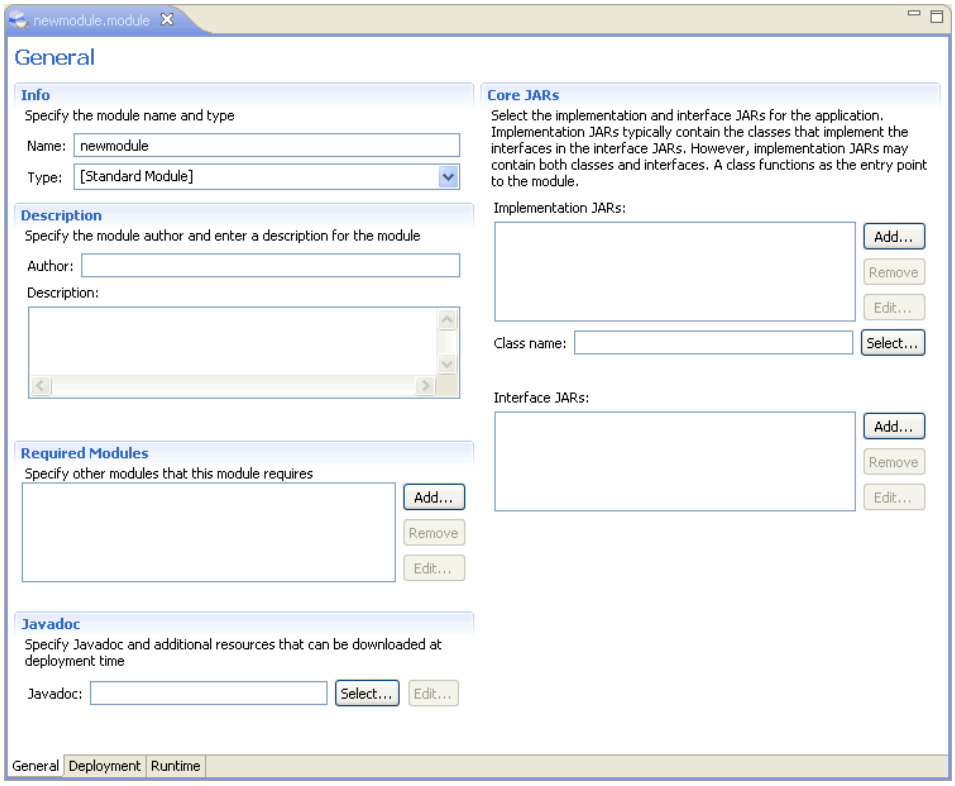
To create a module:

1. In your Documentum Composer project, expand the **Artifacts** folder and right-click **Modules**. Select **New > Module**.

   The **New Module** dialog appears.



2. Enter a name for the new module or accept the default name, then click **Finish**. The **Module** editor appears with the **General** tab selected.



3. Enter the required and optional properties in the **Info**, **Description**, **Required Modules**, **Javadoc**, and **Core JARs** sections, as described in the following table:

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.
Type	A string specifying the type of the module. Required parameter. Enter the module type or select the type from the drop-down list. Documentum 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 module.

Property	Description
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.
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.

4. Click the **Deployment** tab and configure the module deployment options as described in "Configuring module deployment" on page 144.
5. Click the **Runtime** tab to configure the runtime environment for this module, as described in "Configuring the module runtime environment" on page 146.

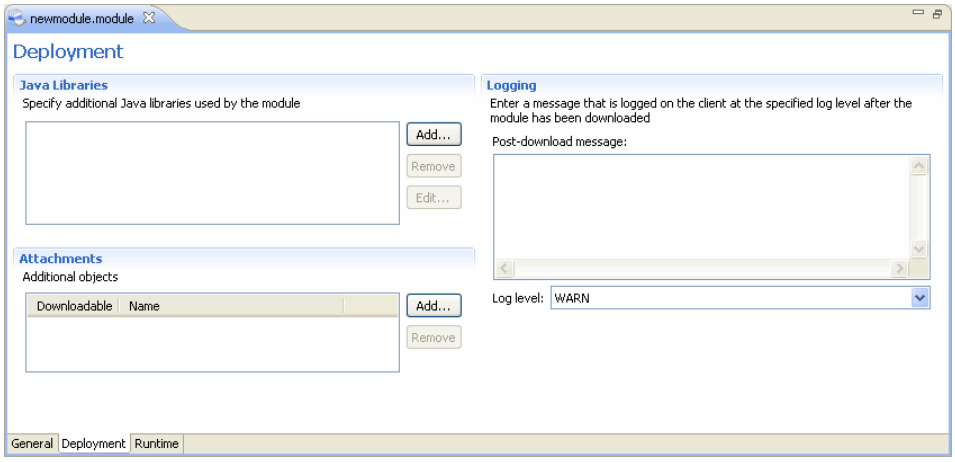
12.3 Configuring module deployment

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

To configure module deployment:

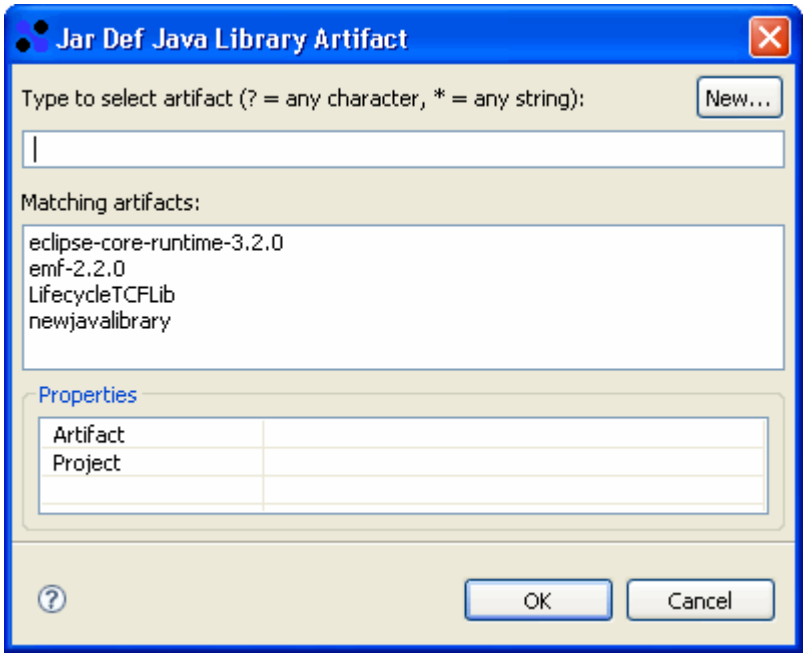
1. Click the **Deployment** tab in the **Module** editor.

   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 Java Library. You can only link existing Java libraries into this module. You cannot modify an existing library that multiple modules share. "Linking and configuring a Java library" on page 107 information about creating a Java Library.

3. In the **Attachments** section, specify Javadocs and other resources that must 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.

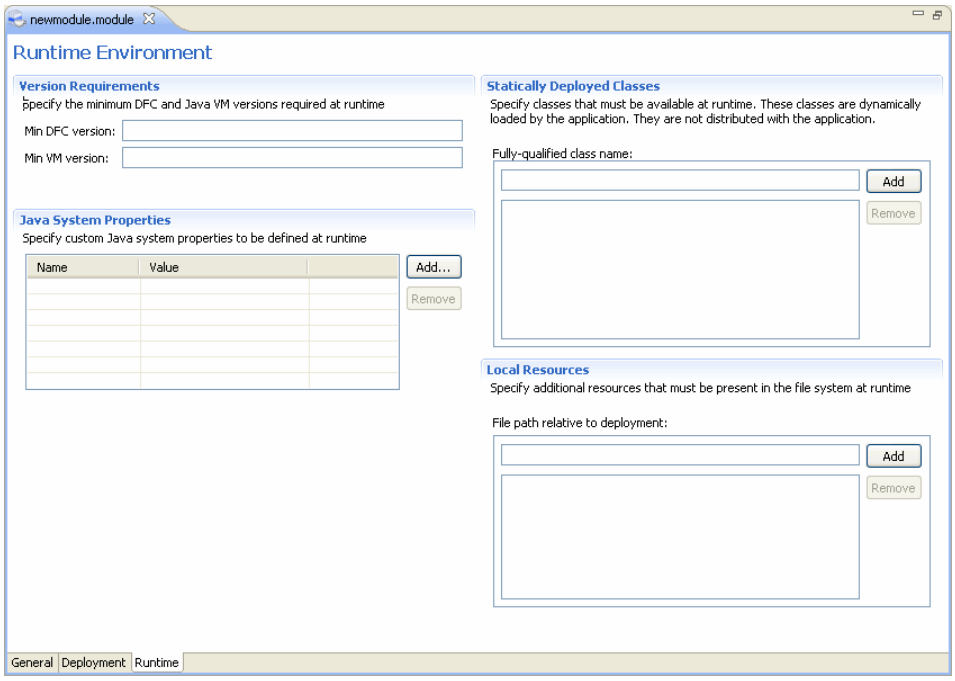
12.4 Configuring the module runtime environment

The runtime environment lets you configure optional properties that must be present in the runtime environment, 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 Environment** view appears.



2. Specify the version requirements, Java system properties, statically deployed classes, and local resources, as described in the following table:

Property	Description
Version Requirements	This section lets you specify the Foundation Java API and Java VM versions required on the client for the module to function properly.
Min Foundation Java API version	The minimum Foundation Java API 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**.
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 13

Managing Permissions Sets (ACLs)

13.1 Permissions, permission sets, and permission set templates

Access to folders and documents in a repository is subject to security restrictions of an organization. All content in the repository is associated with object permissions. Object permissions determine the access users have to each object in the repository such as a file, folder, or cabinet. The object permissions also govern the ability of users 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. Permission sets are 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 objects, perform file-management actions such as importing, copying, or linking files, and start processes.

ACLs are the mechanism that Documentum CM 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. Documentum Composer lets you create permission set templates, public, and regular ACLs, as follows:

- Template: Creates a permission set template. Template ACLs are used to make applications and lifecycles portable. For example, an application that uses a template ACL could be used by various 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 anyone in a repository can use. Public ACLs are available for use by any user in the repository.
- Regular: Creates a regular ACL that only the user or group that creates it can use it.

13.1.1 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 permissions are described in the following table:

Basic Permission	Description
None	No access to the object is permitted.
Browse	Users can view the properties of an object but not the content of the object.
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 content of an object and check in a new version of the item (with a new version number). Users cannot overwrite an existing version or edit the properties of an item.
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.

13.1.2 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 the following table:

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 must also have Browse permission.
Change State	Users can change the state of an item with a lifecycle applied to it.

Extended Permission	Description
Change Permissions	Users can modify the basic permissions of an object.
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 might want a user to delete documents but not read them. This extended permission is useful for Records Management applications where discrete permissions are common.

13.2 Creating a permission set template

Use permission set templates to make applications and lifecycles portable. For example, an application that uses a permission set template could be used by various 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.

You can add a specific user or group to the permission set template. A permission set template can also serve as a template for a permission set. Suppose that you have a permission set but you do not know who is assigned to it yet. For example, set up a permission set template named Template_1 with the following permissions:

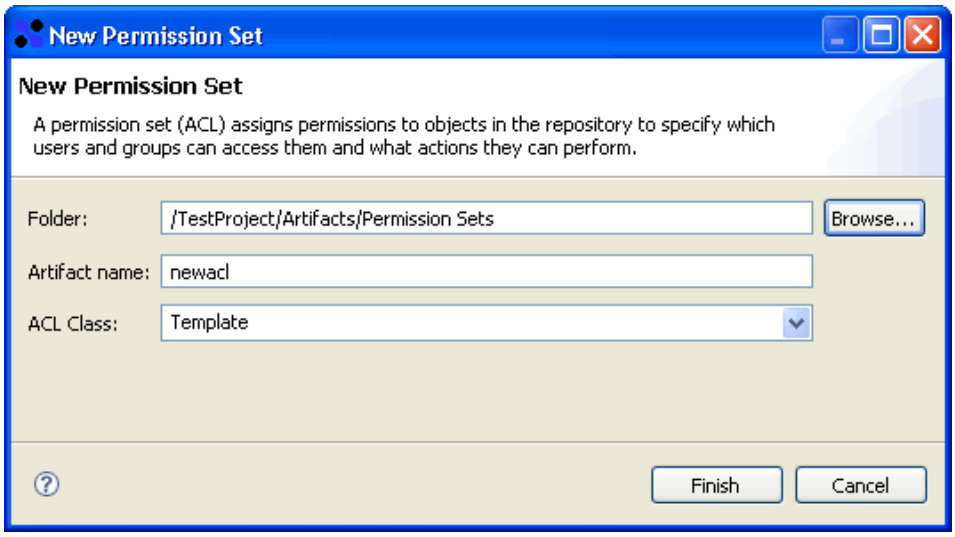
- dm_owner: VERSION and some extended permissions
- dm_world: READ and some extended permissions
- admingroup: DELETE and some extended permissions
- %superuser: DELETE and some extended permissions
- %otheruser: BROWSE and some extended permissions

In this example, %superuser and %otheruser must be set up in one alias set so that it can be resolved when the permission set template is used.

To create a permission set template:

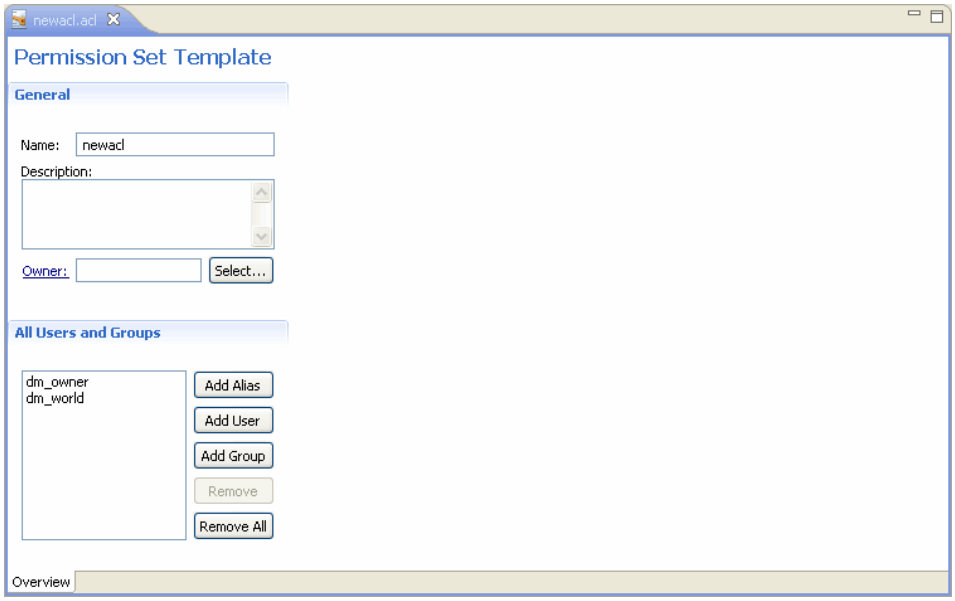
1. In your Documentum Composer project, expand the **Artifacts** folder and right-click **Permission Sets**. Select **New > Permission Set**.

   The **New Permission Set** dialog appears.



2. 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.
3. Enter a name for the permission set in the **Artifact name** field.
4. In the **ACL Class** field, select **Template** and 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 alias entries from a permission set template.

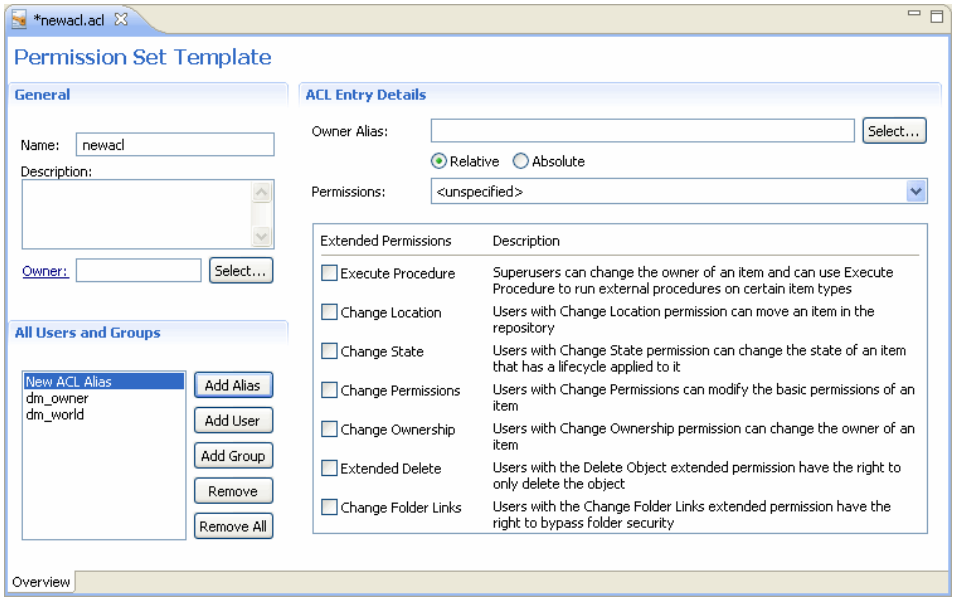
5. In the **General** section, specify general information as described in the following table:

Parameter	Description
Name	Type a name for the permission set template.
Description	Optional. Type a description for the permission set template.
Owner	Click **Select** to select the owner of the permission set template.

6. In the **All Users and Groups** section, select the **dm_owner** or **dm_world** alias, or click one of the following:

   - **Add Alias**
   - **Add User**
   - **Add Group**

   The **ACL Entry Details** section appears.



7. In the **ACL Entry Detail** section, specify the name and permissions for the alias, user, or group that you selected, as described in the following table:

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, create an alias first. "Creating an alias set" on page 75 provides more information about creating an alias set. For the Owner Alias, also select one of the following: • **Relative**: Select to specify a relative alias name (%alias_name). • **Absolute**: Select to specify aliases from more than one alias set (%alias_set_name.alias_name).
Owner User or Owner 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, create an owner in the form of a user or group installation parameter. You cannot add users or groups directly to the ACL. 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. "Creating an ACL entry owner" on page 157 provides information about creating an ACL entry owner.
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. "Basic permissions" on page 150 and "Extended permissions" on page 150 provides information about permissions.

8. Save your changes.

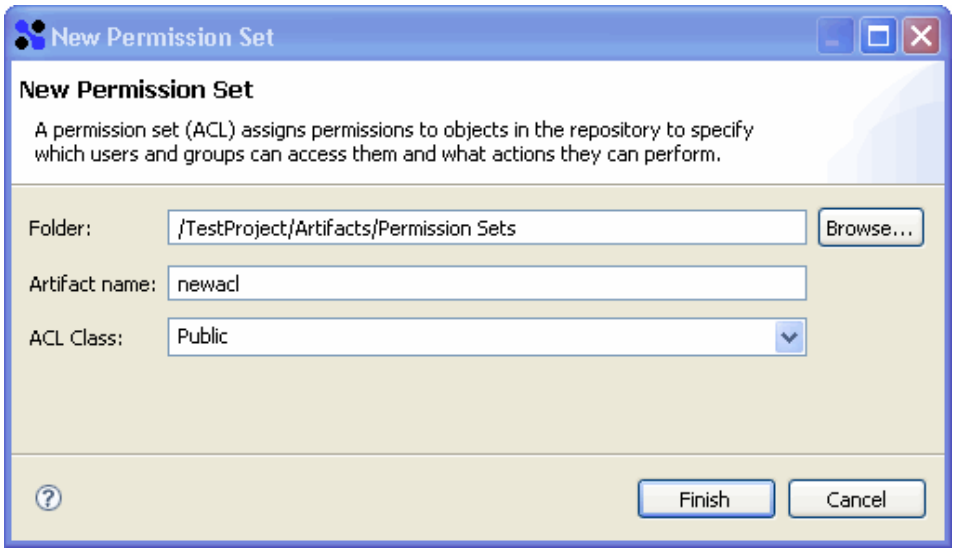
13.3 Creating a regular or public permission set

Regular ACLs can be used only 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:

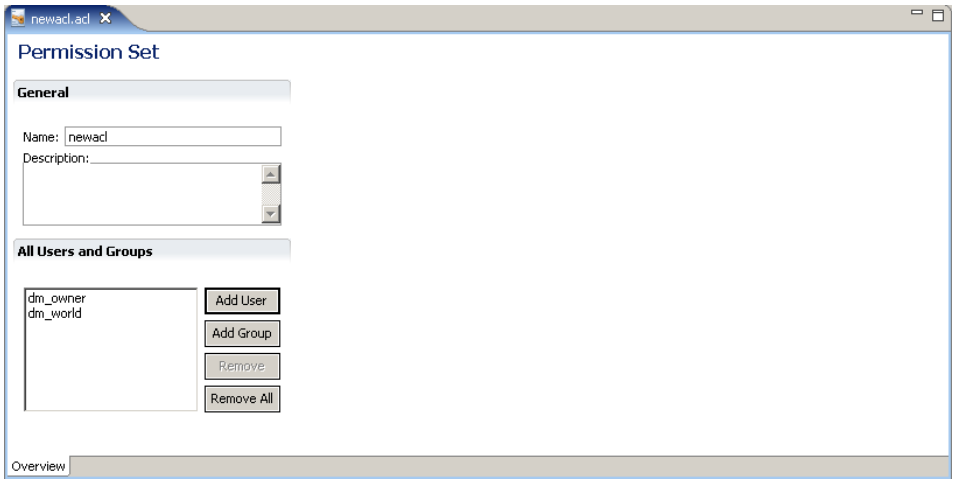
1. In your Documentum project, expand the **Artifacts** folder and right-click **Permission Sets**. Select **New > Permission Set**.

   The **New Permission Set** dialog appears.



2. 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.
3. Enter a name for the permission set in the **Artifact name** field.
4. Depending on what type of ACL you want to create, select **Regular** or **Public** from the *ACL class* drop-down list, 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:

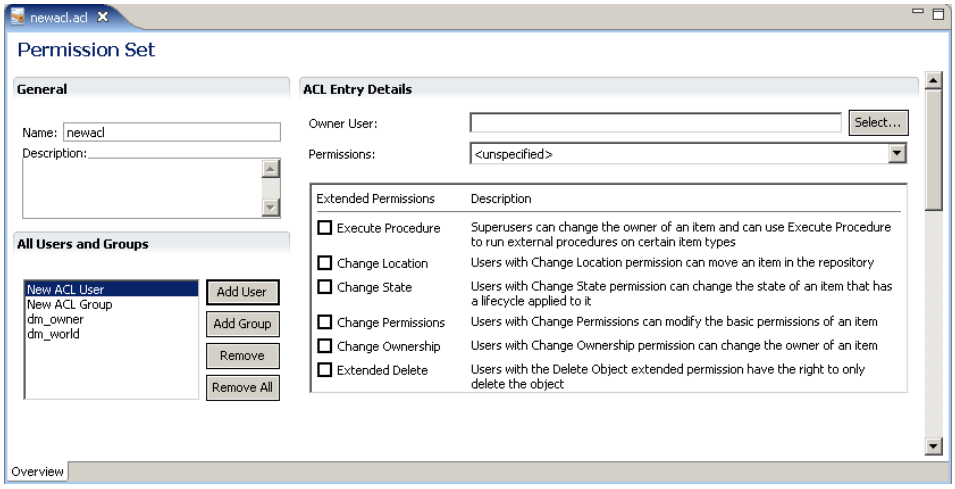
- **dm_owner**: The owner of the permission set.
- **dm_world**: All repository users.

You cannot delete these default entries from a permission set.

5. Enter a name and an optional description for the permission set in the **General** section.
6. Select the dm_owner or dm_world ACL entry in the **All Users and Groups** section or click one of the following:

   - **Add User** to add a new user ACL.
   - **Add Group** to add a new group ACL.

   The **ACL Entry Details** section appears.



7. Specify the name and permissions for the ACL entry that you selected, as described in the following table:

Parameter	Description
Owner User or Owner 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, create an owner in the form of a user or group installation parameter. You cannot add users or groups directly to the ACL. 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. "Creating an ACL entry owner" on page 157 provides information about creating an ACL entry owner.
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. "Basic permissions" on page 150 and "Extended permissions" on page 150 provides information about permissions.

8. Save your changes.

13.3.1 Creating an ACL entry owner

Every ACL entry requires an owner, which can be a user or a group of users. Documentum Composer lets you create an owner parameter in the 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. In the **ACL Entry Details** section of the **Permission Set** editor, click **Select**.

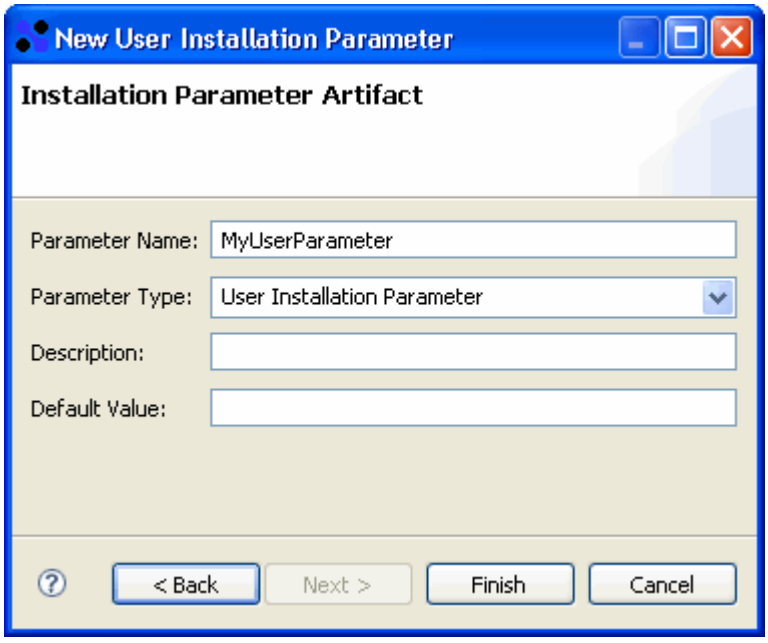
   The **User Installation Parameter** or **Group Installation Parameter** dialog appears.

2. Click **New**.

   The **New User Installation Parameter** or **New Group Installation Parameter** dialog appears.

3. Accept the default folder location and artifact name, and click **Next**.

   The **Installation Parameter Artifact** dialog appears.



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

Chapter 14

Managing Procedures

14.1 Procedures

Procedures are applications that extend or customize the behavior of Documentum clients or Documentum CM 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.

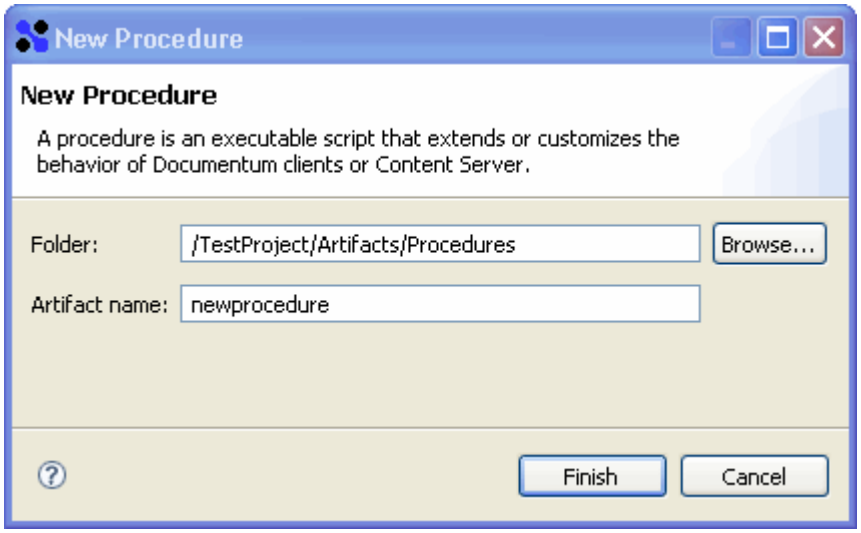
14.2 Creating a procedure

Use the Procedure editor to create a procedure.

To create a procedure:

1. In your Documentum project, expand the **Artifacts** folder and right-click **Procedures**. Select **New > Procedure**.

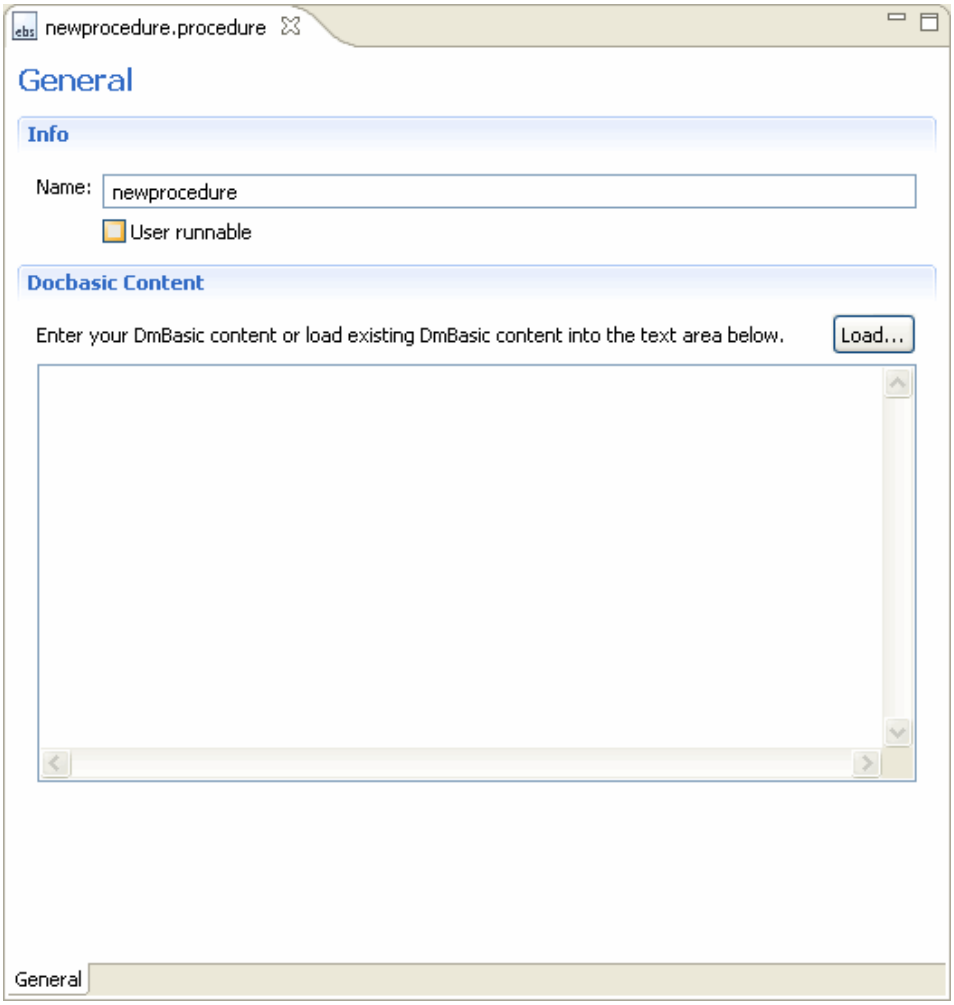
   The **New Procedure** dialog appears.



2. 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.

3. Enter a file name for the procedure in the **Artifact name** field, then click **Finish**.

   The **Procedure** editor appears.



4. Enter a name for the procedure in the **Name** field or accept the default name.
5. Select the **User runnable** check box to enable a user to execute the procedure in the associated client application.
6. 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 15

Managing Relation Types

15.1 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 object 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.

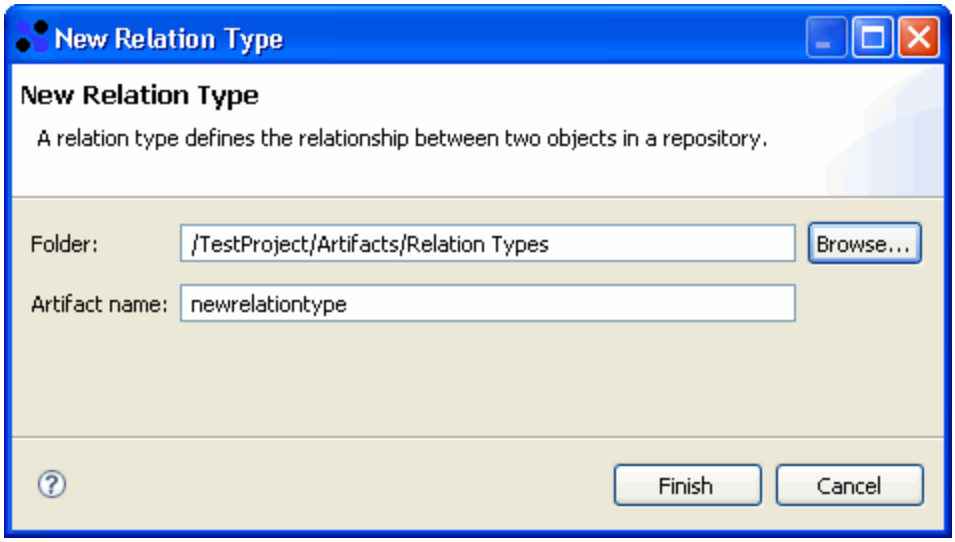
15.2 Creating a relation type

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

To create a relation type:

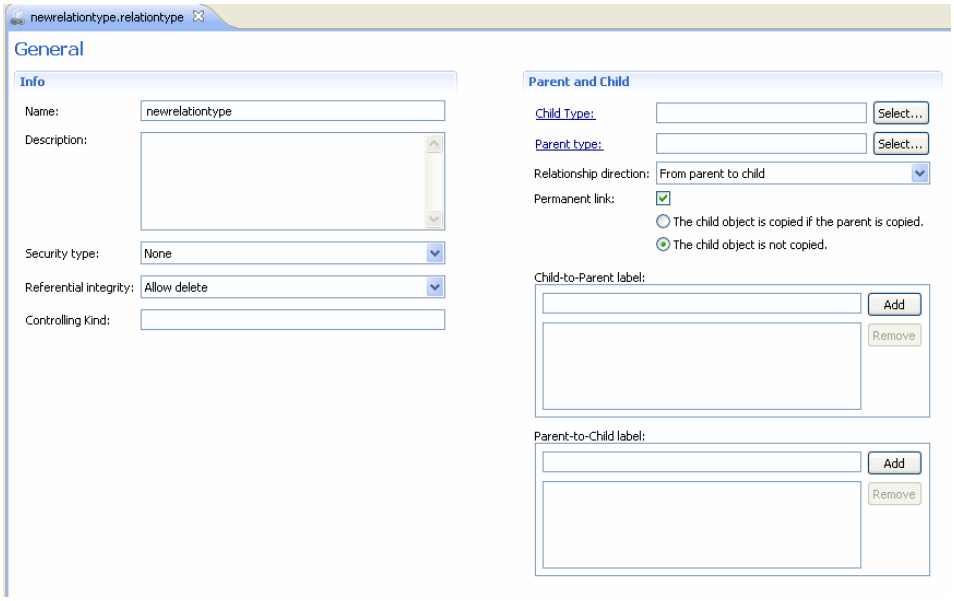
1. In your Documentum project, expand the **Artifacts** folder and right-click **Relation Types**. Select **New > Relation Type**.

   The **New Relation Type** dialog appears.



2. 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.
3. Enter a file name for the relation type in the **Artifact name** field, then click **Next**.
4. Enter a name for the relation type in the **Relation type name** field, then click **Finish**.

   The **Relation Type** editor appears.



5. Enter the relation type properties in the **General** and **Parent and Child** sections, as described in the following table:

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.

Property	Description
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.
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 list box. • Click the **Child type** link to create a child type. The **Documentum Artifact - Name and Location** wizard appears. "Creating an artifact" on page 26 provides information about creating an artifact.

Property	Description
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 parent 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 list box. • Click the **Parent type** link to create a parent type. The **New Documentum Artifact - Name and Location** wizard appears. "Creating an artifact" on page 26 provides information about creating an artifact.
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**.
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 16

Managing Smart Containers

16.1 Smart containers

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

Because a smart container template is intended for repeated construction of a modeled composite object, each new instance of the composite object must be different. You accomplished this 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.

16.2 Constructing a smart container

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

To construct a smart container:

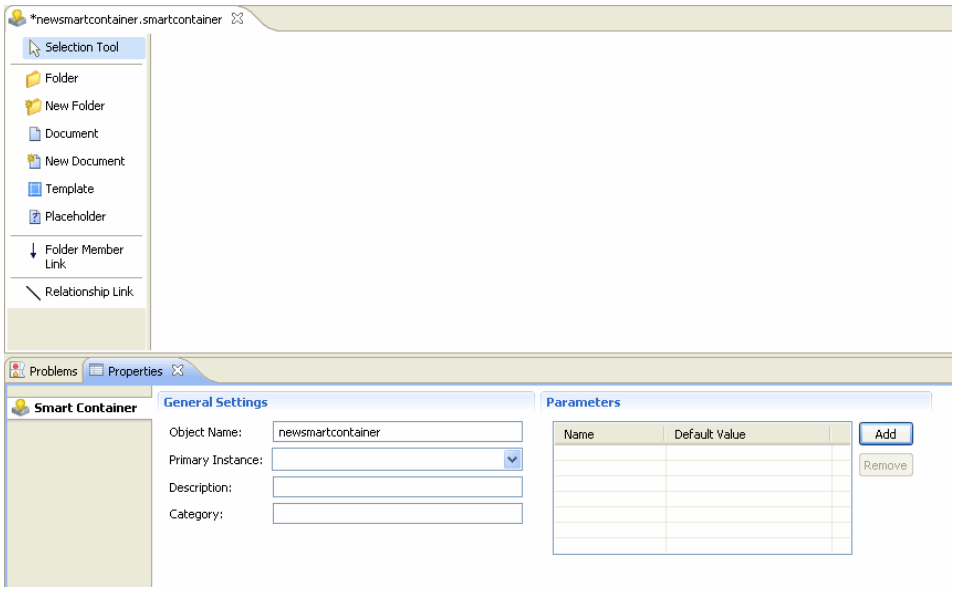
1. In your Documentum project, expand the **Artifacts** folder and right-click **Smart Containers**. Select **New > Smart Container**.

   The **New Smart Container** dialog appears.

2. 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.

3. Enter a file name for smart container in the **Artifact name** field, then click **Next**.

   The **Smart Container** editor appears.



4. Configure the smart container properties, as described in following table:

Parameter	Description
General Settings	
Object Name	A string specifying the name of the smart container. You can accept the default name that is assigned by Documentum 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.
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	
Name	The name of a parameter that the smart container uses at runtime.
Default Value	The default value of the runtime parameter.

5. Add one or more artifacts to your smart container, as follows:

   - **Folder**, as described in "Adding a folder" on page 167.
   - **New folder**, as described in "Adding a new folder" on page 168.
   - **Document**, as described in "Adding a document" on page 169.

- **New document**, as described in "Adding a new document" on page 169.
- **Template**, as described in "Adding a template" on page 170.
- **Placeholder**, as described in "Adding a placeholder" on page 171.

6. Add relationships to your artifacts, as described in "Adding smart container relationships" on page 172.
7. Save your changes.
8. Configure the artifact installation parameters for the smart container, as described in "Configuring artifact install options" on page 211.

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.

16.3 Adding smart container elements

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

16.3.1 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 exists in a repository, define the folder by adding the Documentum object ID for the folder or its path.

To add a folder:

1. Open the **Smart Container** editor, as described in "Constructing a smart container" on page 165.
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 the following table:

Parameter	Description
Display Name	The name of the folder that appears in the workspace. This name is 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.

Parameter	Description
Object Id	The 16-character Documentum object id of the folder.
Path	The relative path for a location to which the folder is linked.

16.3.2 Adding a new folder

A new smart container folder is like 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" on page 165.
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 the following table:

Parameter	Description
Display Name	The name of the folder that appears in the workspace. This name is 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.
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 an object type. "Object types" on page 177 provides information about 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 permission set. "Permissions, permission sets, and permission set templates" on page 149 provides information about managing permission sets.

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 an aspect.

   "Aspect modules and aspect types" on page 81 provides information about 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.

16.3.3 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 exists in a repository, define the document by adding the Documentum object ID for the document or its path.

To add a document:

1. Open the **Smart Container** editor, as described in "Constructing a smart container" on page 165.

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 the following table:

Parameter	Description
Display Name	The name of the document instance that appears in the workspace. This name is 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.

16.3.4 Adding a new document

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

To add a document:

1. Open the **Smart Container** editor, as described in "Constructing a smart container" on page 165.

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 the following table:

Parameter	Description
Display Name	The name of the new document instance that appears in the workspace. This name is 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 an object type. "Object types" on page 177 provides information about 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 permission set. "Permissions, permission sets, and permission set templates" on page 149 provides information about managing permission sets.

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 an aspect.

   "Aspect modules and aspect types" on page 81 provides information about 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.

16.3.5 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" on page 165.

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 the following table:

Parameter	Description
Display Name	The name of the template that appears in the workspace. This name is 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 permission set. "Permissions, permission sets, and permission set templates" on page 149 provides information about managing permission sets.
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 an aspect.

   "Aspect modules and aspect types" on page 81 provides information about 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.

16.3.6 Adding a placeholder

A placeholder object is like 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" on page 165.

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 the following table:

Parameter	Description
Display Name	The name of the placeholder that appears in the workspace. This name is 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 an object type. "Object types" on page 177 provides information about types.
Required	Specifies whether the placeholder must be assigned an object type.

4. Save your changes.

16.4 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 distinguish a relationship visually:

- **Folder Member Link**: Use this relation when modeling folders and their members.
- **Relationship Link**: 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 **Folder Member Link** (arrow) or **Relationship Link** (straight line) to activate the linking tool.
2. Click the first smart container object.
3. Click the second smart container object.

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

Chapter 17

Managing SysObjects

17.1 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.

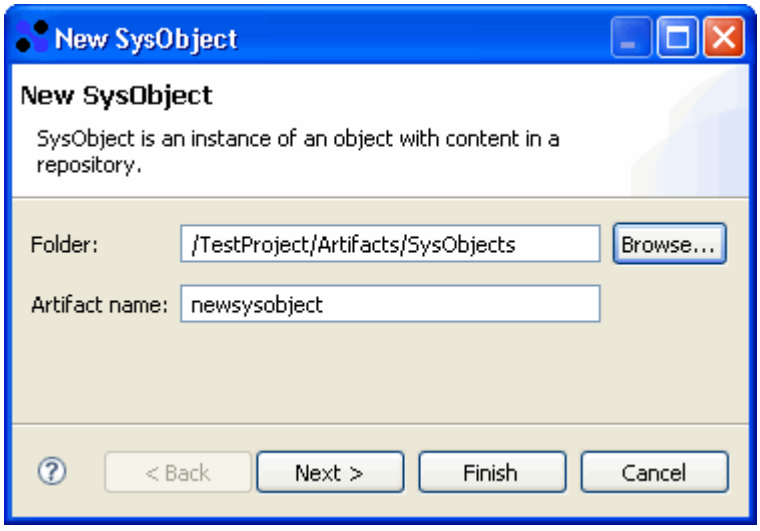
17.2 Creating a SysObject

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

To create a SysObject:

1. In your Documentum Composer project, expand the **Artifacts** folder and right-click **SysObjects**. Select **New > SysObject**.

   The **New SysObject** dialog appears.



2. 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.

3. Enter a file name for the SysObject in the **Artifact name** field, then click **Next**.

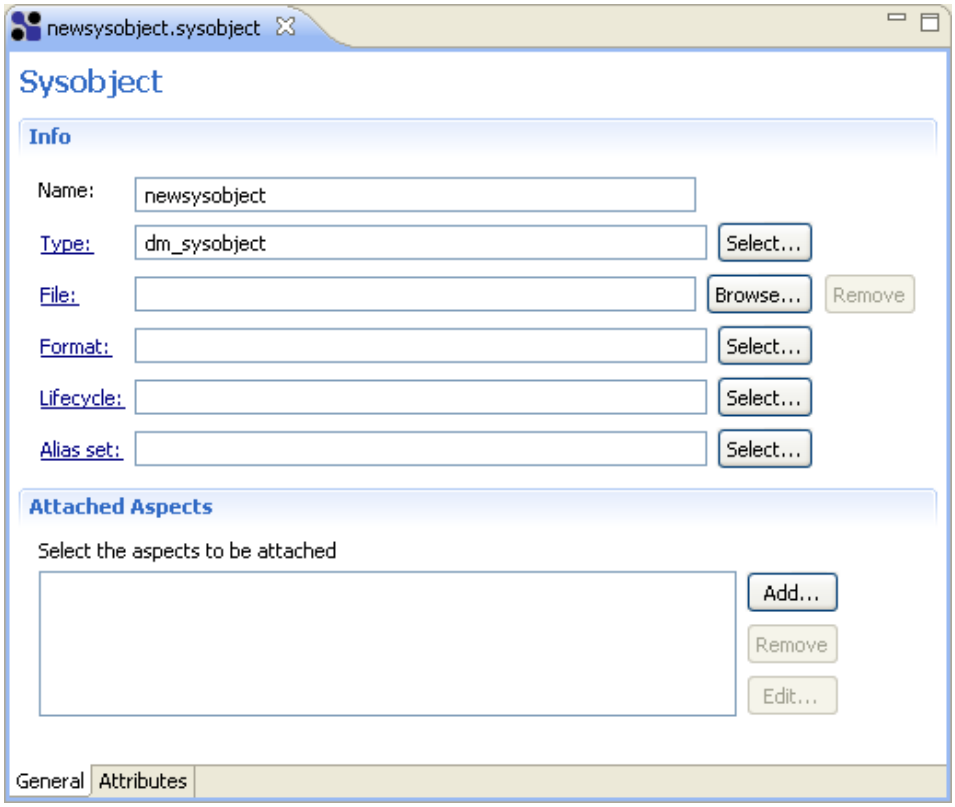
The **Type, File, and Format** dialog appears.

4. Do one of the following:

   - If you do not want to enter a file name and format for the SysObject, click **Finish**.
   - Enter the type, file, and format information for the SysObject, if applicable, as described in the following table, then click **Next**.

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.



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

   The **Aspect Module Artifact** dialog appears.

6. Select an aspect from the list or click **New** to create an aspect.

   "Aspect modules and aspect types" on page 81 provides information about aspects.

7. View or modify the SysObject attributes, as described in "Viewing and modifying SysObject attributes" on page 176.

8. Save your changes.

17.3 Viewing and modifying SysObject attributes

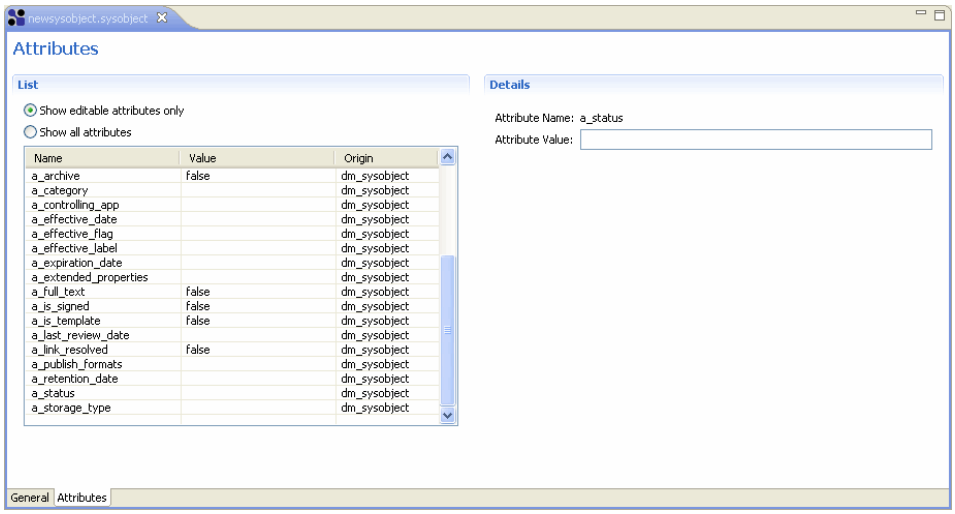
The **Attributes** tab in the **SysObject** editor lets you view attributes and modify the attribute values associated with the specified SysObject.

To view or modify SysObject attributes:

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

   The **Attributes** view appears.

2. Select an attribute 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. Save your changes.

Chapter 18

Managing Types

18.1 Object types

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

- Standard objects
- Lightweight objects

A set of attributes defines every object type. When an object is created, its attributes are set to values that describe that 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. The *OpenText Documentum Server Fundamentals Guide* provides information about objects and object types.

Lightweight objects are part of an object model enhancement introduced to share system managed metadata among objects which hold only 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 High-Volume Server can make proper use of lightweight objects. High-Volume Server is an extension of Documentum CM 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 installing Documentum CM Server. The *OpenText Documentum Content Management - High-Volume Server Development Guide (EDCCS-DGD)* provides information about lightweight object types and High-Volume Server.

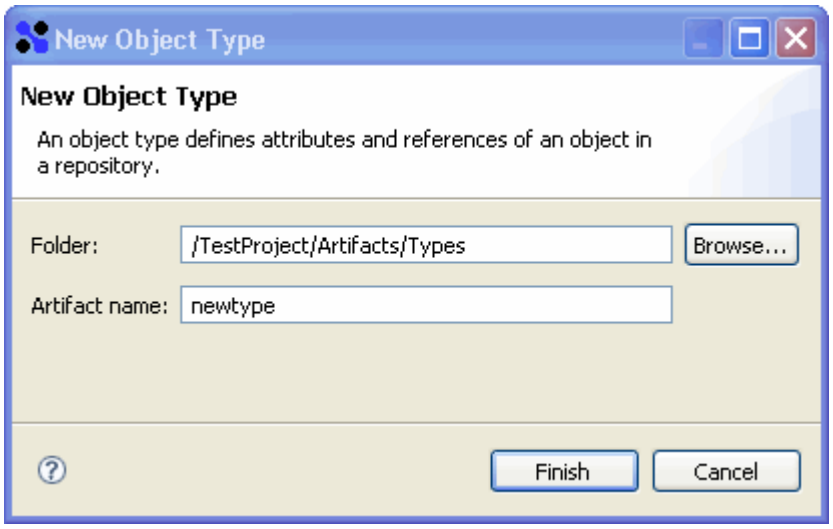
18.2 Creating a standard object type

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

To create a standard object type:

1. In your Documentum Composer project, expand the **Artifacts** folder and right-click **Types**. Select **New > Type**.

   The **New Object Type** dialog appears.



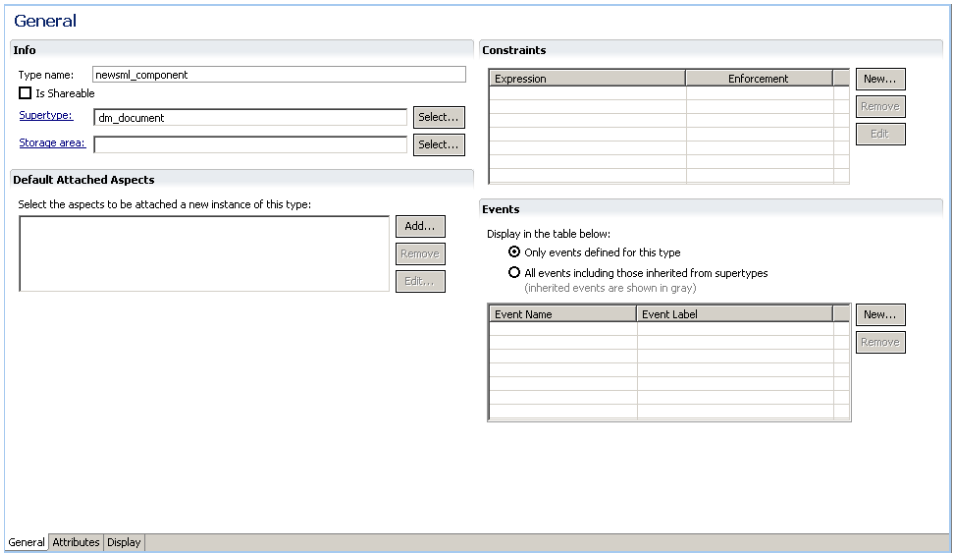
2. Enter the folder path and name of the project for which you want to create an object type in the **Folder** field, or click **Browse** to select the project from a folder list.

3. Enter a file name for the object type in the **Artifact name** field.

4. If the lightweight SysObject plugin is installed, select **Standard object type**, then click **Next**.

   "Installing the lightweight SysObject plug-in" on page 11 provides instructions on how to install the lightweight SysObject plugin.

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



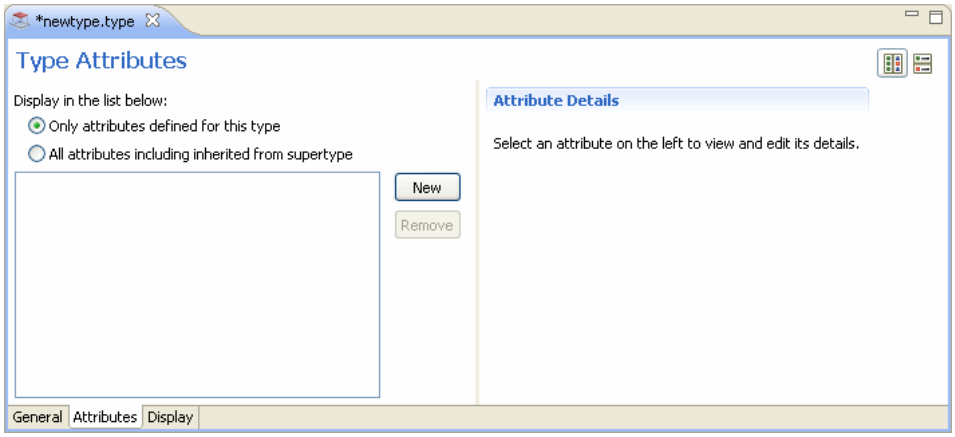
5. Enter the object type information in the **Info**, **Default Attached Aspects**, **Constraints**, and **Events** sections, as described in the following table:

Property	Description
General	
Type name	A string that specifies the name of the type. The following rules apply to all type names: • A maximum of 27 characters, all lower-case. The Documentum CM 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 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, Documentum 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. "Attaching aspects" on page 181 provides information about attaching aspects.

Property	Description
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. "Configuring constraint expressions" on page 83 provides information about creating or modifying an expression.
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. "Adding, deleting, or modifying events" on page 187 provides information about creating an event.
Only events defined for this type	Select this option to display 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.

6. Click the **Attributes** tab.

   The **Type Attributes** view appears.



7. Click **New** to add a type attribute. Enter information in the **Structure** and **Constraints** sections, as described in "Adding type attributes" on page 187.

8. Click the **Display** tab.

   The **Type UI Information** view appears.

9. Enter the type UI information in the **Application Interface Display** and **Display Configuration** sections, as described in "Configuring the type UI information" on page 197.

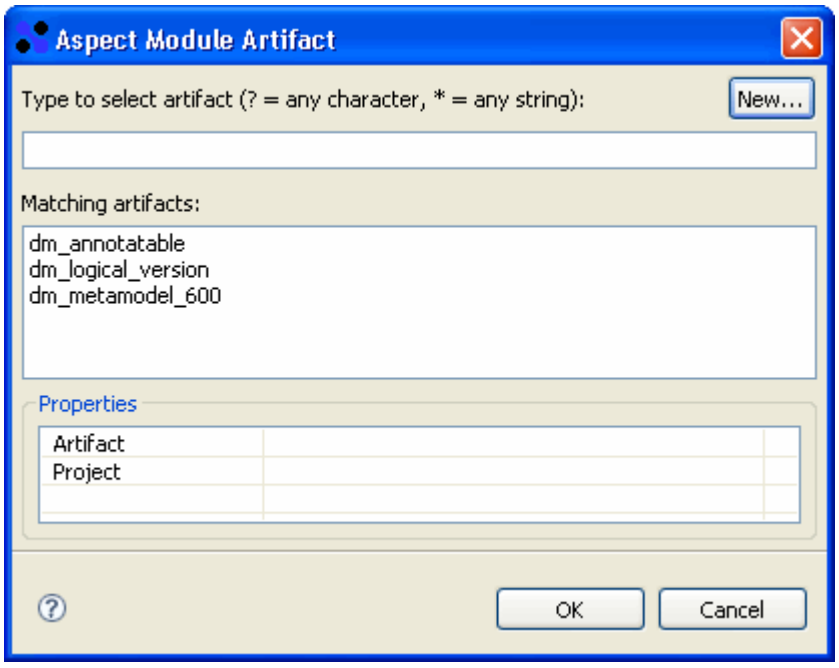
18.2.1 Attaching aspects

When you create an 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** list box, then click **OK**.

Note: If there are no aspects listed in the **Matching Artifacts** list box, no aspects have been created yet. "Creating an aspect type" on page 81 information about creating an aspect.

18.3 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 High-Volume Server have a use for lightweight object types. High-Volume Server is an extension of Documentum CM 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 Documentum CM Server is installed. Also, verify that the lightweight SysObject plugin is installed. "Installing the lightweight SysObject plug-in" on page 11 provides instructions on how to install the lightweight SysObject plugin.

To create a lightweight object type:

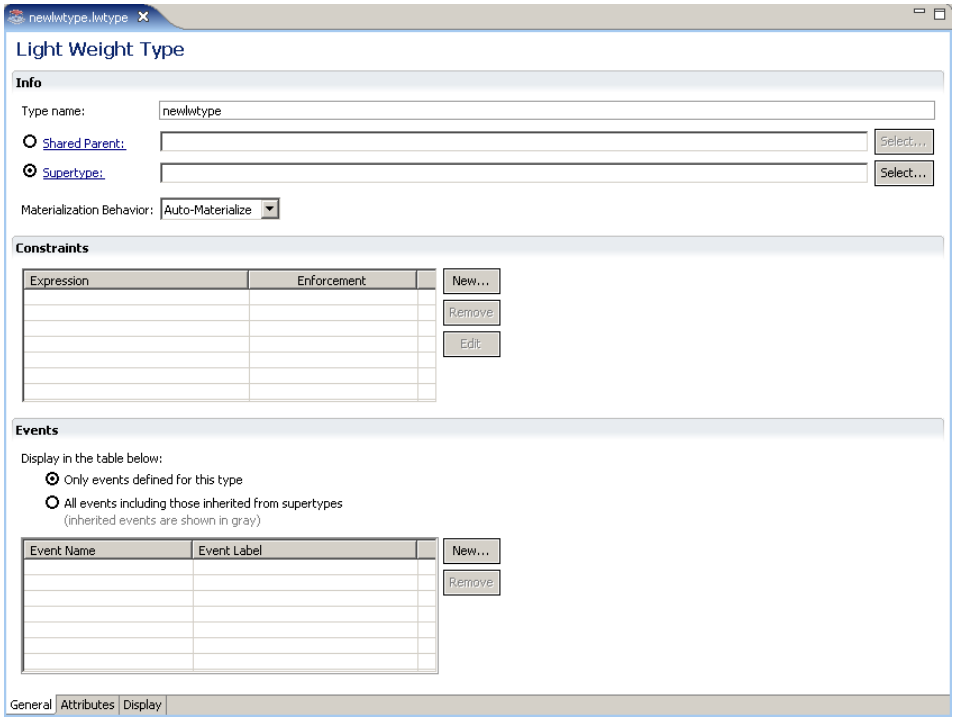
1. In your Documentum Composer project, expand the **Artifacts** folder and right-click **Types**. Select **New > Type**.

   The **New Object Type** dialog appears.

2. 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.

3. Enter a file name for the object type in the **Artifact name** field.
4. Select **Lightweight object type**, then click **Next**.

   The **Lightweight Type** editor appears with the **General** tab selected.



5. Enter the object type information in the **Info**, **Constraints**, and **Events** sections, as described in the following table:

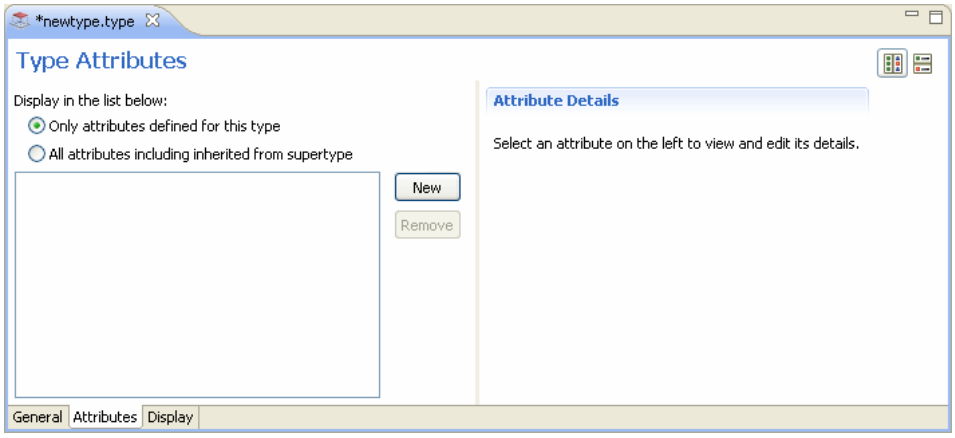
Property	Description
General	
Type name	A string specifying the name of this lightweight object type. The following rules apply to all type names: • A maximum of 27 characters, all lower-case. The Documentum CM Server is case-insensitive and stores all type names in lower-case. • The first character 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.

Property	Description
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 can drastically increase the size of database tables. The materialization behavior can be set to the following values: • **Auto-Materialize**: The lightweight object is materialized automatically when certain operations occur, for example a checkout or checkin operation or a branch operation. • **On-Request**: The lightweight object is materialized only when requested by an explicit API call. • **Disallow**: The lightweight object is never materialized. 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 an expression. "Configuring constraint expressions" on page 83 provides information about creating or modifying an expression.
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. "Adding, deleting, or modifying events" on page 187 provides information about creating an event.

Property	Description
Only events defined for this type	Select to display events that are defined for this type in the events table.
All events including inherited from supertype	Select to display all events for this type in the events table, including events that are inherited from the supertype.
Event name	A string that specifies the name of the event that is associated with instances of this type.
Event label	A string that specifies the label for the event.

6. Click the **Attributes** tab.

   The **Type Attributes** view appears.



7. Click **Add** to add a new type attribute. "Adding type attributes" on page 187 provides information about adding type attributes.

8. Click the **Display** tab.

   The **Type UI Information** view appears.

9. Enter the type UI information in the Display Configuration and Application Interface Display sections, as described in "Configuring the type UI information" on page 197.

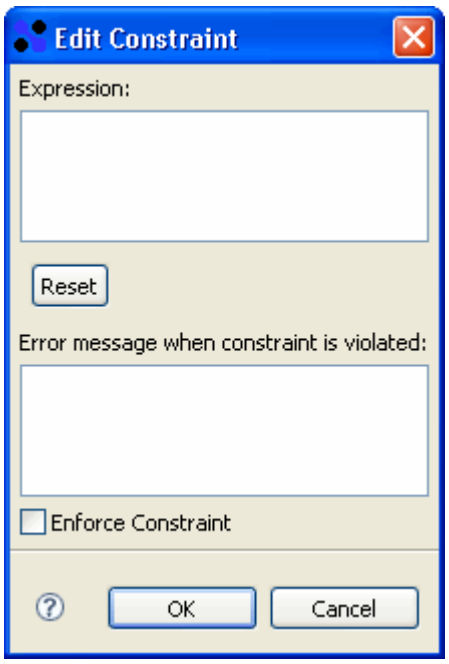
18.4 Configuring constraint expressions for a type

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 resolves 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. Select **Enforce Constraint** to instruct applications to enforce this constraint or clear the check box to not enforce the constraint.

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

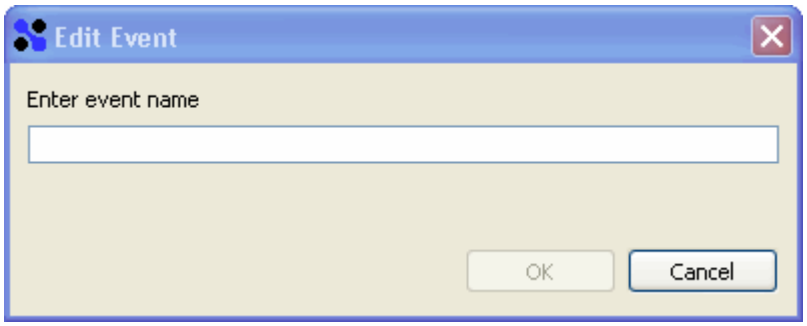
18.5 Adding, deleting, or modifying events

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

To create an event:

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

   The **Edit Event** dialog appears.



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

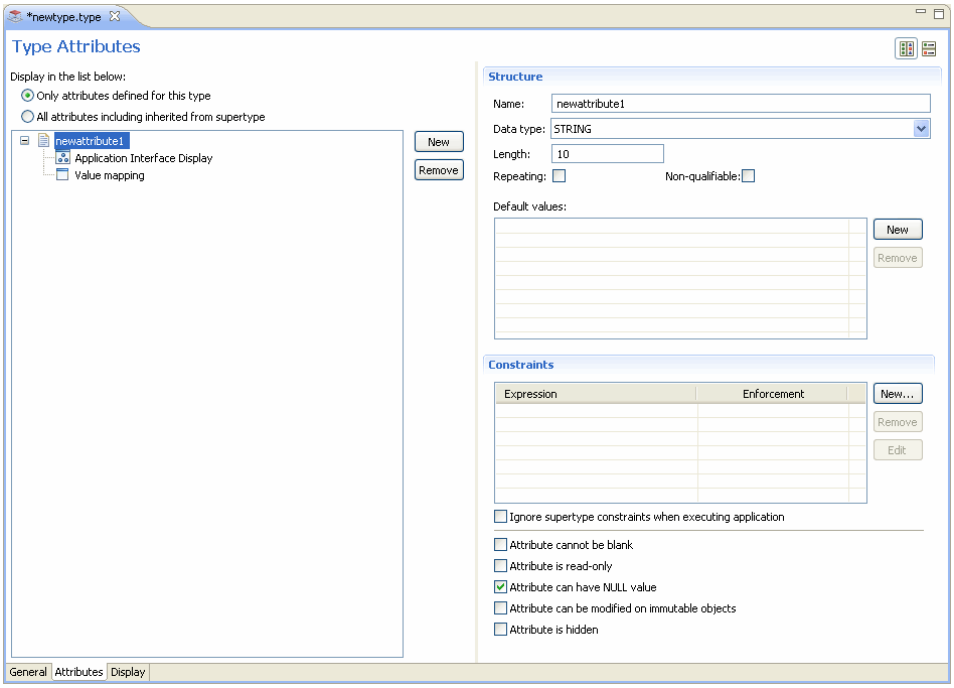
18.6 Adding type attributes

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 an attribute entry, then click the + sign to display the configuration options.

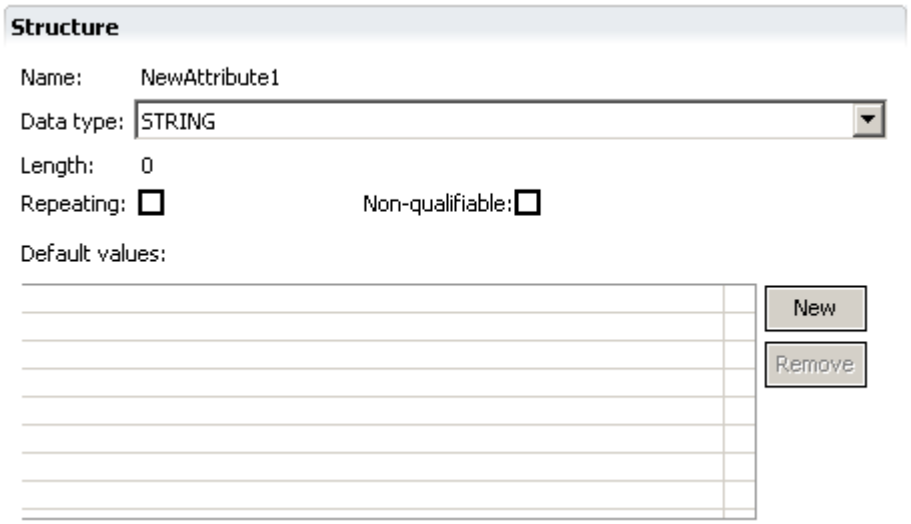
   The **Type Attributes** view expands and displays the **Structure** and **Constraints** section.



3. Configure the attribute structure, as described in "Configuring attribute structure" on page 189.
4. Configure the attribute constraints, as described in "Configuring attribute constraints" on page 190.
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 type attribute UI" on page 192.
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" on page 194.
9. Configure the attribute value mapping, as described in "Configuring attribute value mapping" on page 196.

18.6.1 Configuring attribute structure

The attribute structure is configured in the Structure section of the Type Attributes view:



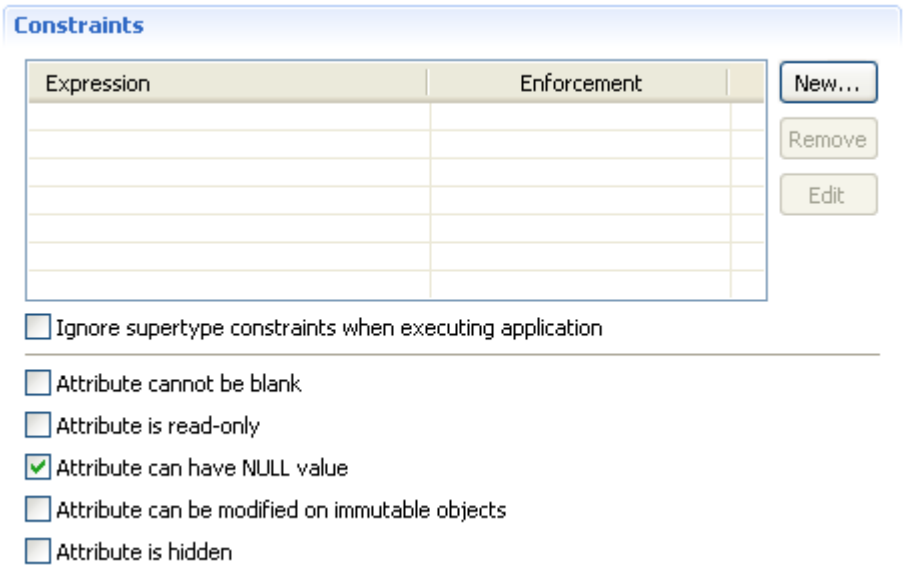
Enter the attribute structure properties, as described in the following table:

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 install the application.

Property	Description
Repeating	Specifies whether this attribute can have more than one value. Select the check box 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. They 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.

18.6.2 Configuring attribute constraints

Attribute constraints are configured in the Constraint section of the Type Attributes view.



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

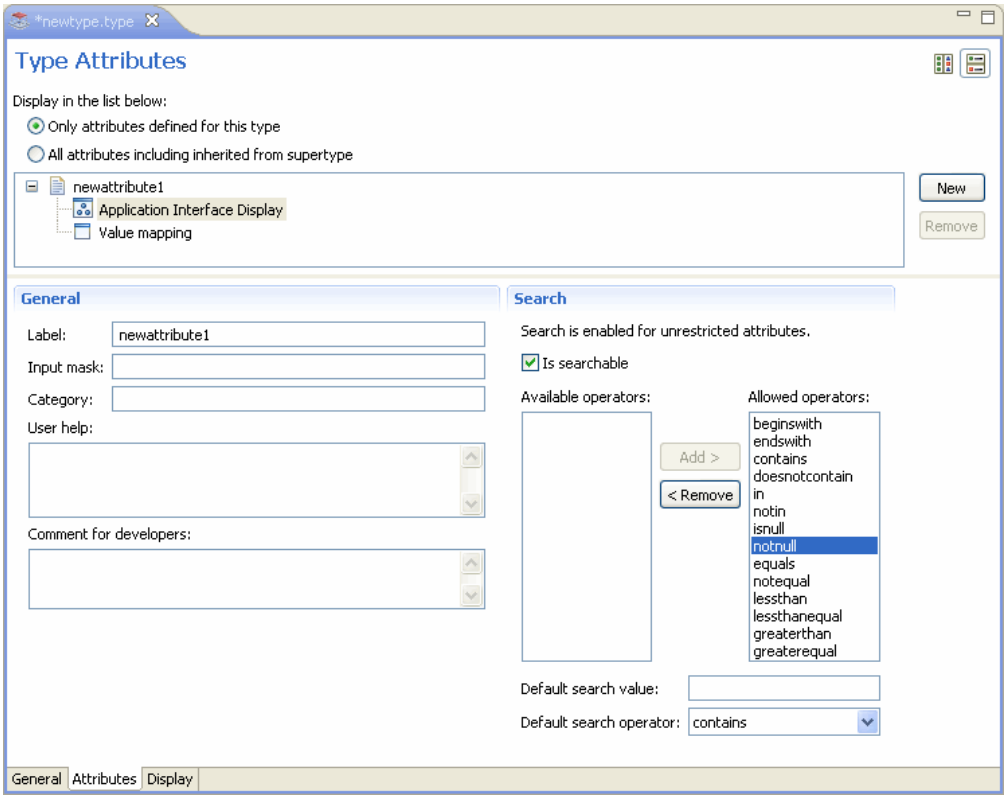
Enter or specify the attribute constraint properties, as described in the following table:

Property	Description
Expression	The Docbasic expression that defines the constraint. Click **New** to create an expression. "Configuring constraint expressions" on page 83 provides information about how to create or modify an expression.
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 to specify that applications should ignore supertype constraints for attributes of this type.
Attribute cannot be blank	Select 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 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 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 if you want users to be able to modify the attribute even though the object itself is immutable (unchangeable).

18.6.3 Configuring 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 Application Interface Display in the attribute directory tree.

The Type Attributes UI properties view displays the **General** and **Search** sections.



Enter or specify the attribute UI properties in the **General** and **Search** sections, as described in the following table:

Property	Description
General	
Label	The name that is displayed for this attribute in the client application.

Property	Description
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 • **L**: Alphabetic, gets converted to lowercase For input mask examples, see "Input mask" on page 193.
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.

Input mask

The following table shows a mask value and an example of valid user input:

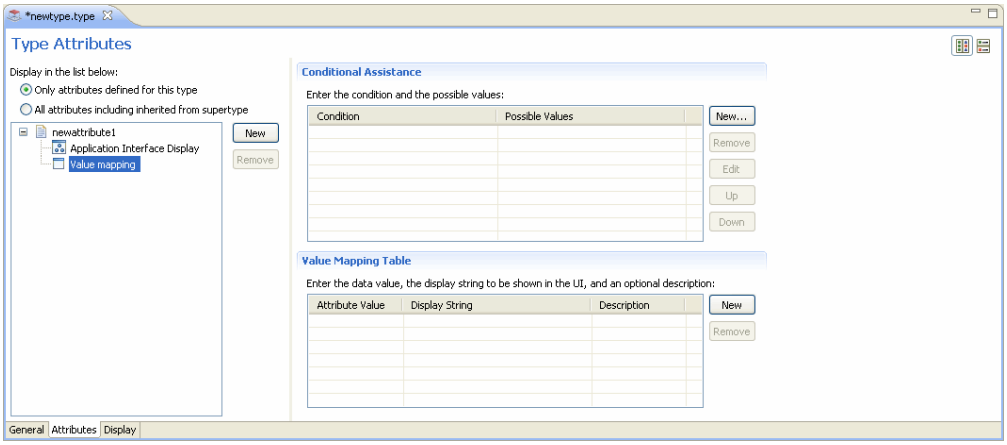
Mask Value	Description	Valid User Input
##/##/####	Specifies a date entry.	08/23/1968
##:## UU	Specifies a time entry.	2:42 PM
###-##-####	Specifies a format for entering a numerical sequence, such a US social security number.	111223333
??????????????	Specifies a sequence for entering up to 15 letters,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

18.6.4 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.

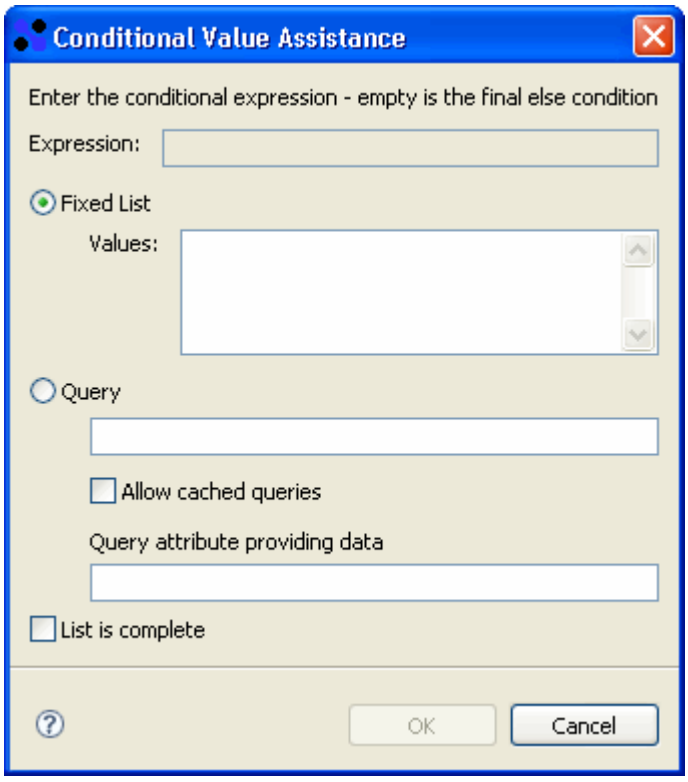
Conditional value mappings are configured in the **Conditional Assistance** section of the **Value Mapping** view:



To create a conditional value mapping:

1. In the **Conditional Assistance** section of the **Value mapping** view, click **New**.

   The **Conditional Value Assistance** dialog appears.



2. Specify the conditional value properties, as described in the following table:

Property	Description
Expression	A Docbasic expression that specifies the condition. The Docbasic expression must resolve to true or false.
Fixed list	Specifies that the values are associated with the condition in the form of a fixed list. Select this option to use a fixed list of values and type 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. String values must be entered on separate lines.
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.

Property	Description
Allow cached queries	Select 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.

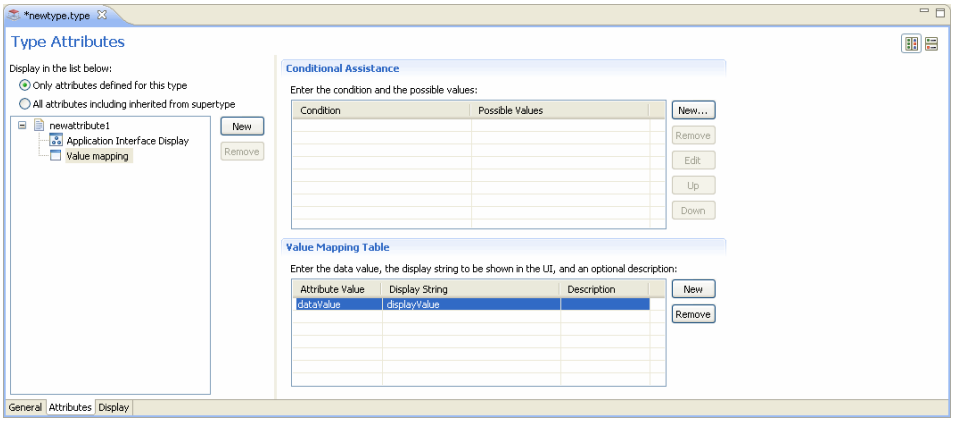
18.6.5 Configuring attribute value mapping

Value mapping associates or maps attribute values to strings that display 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.

To create or modify a value mapping:

1. In the **Value Mapping Table** section of the **Value mapping** view, click **New**.

   The system displays **dataValue** and **displayValue** 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.

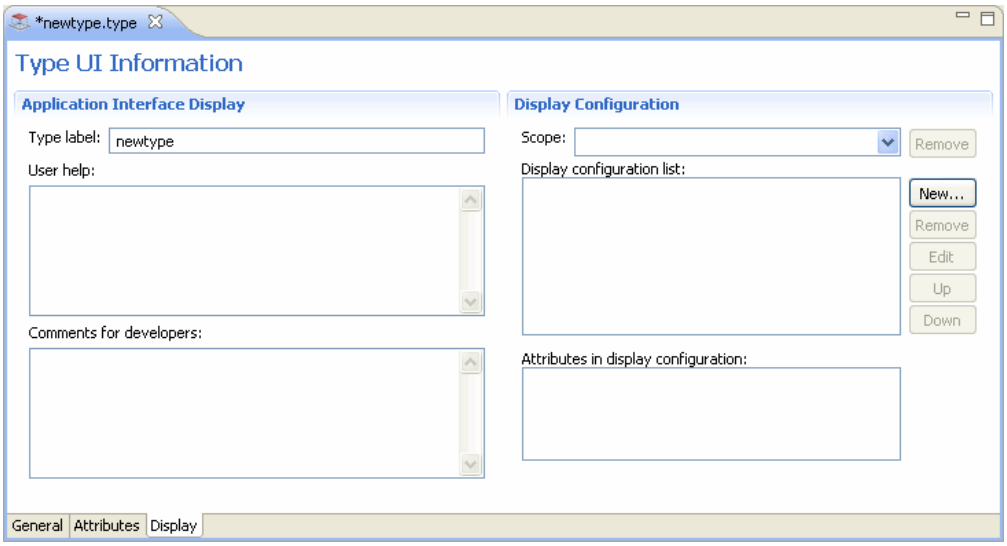
3. Click the **displayValue** field in the **Display String** column and enter the string that is displayed as the mapped value in the client application.

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**.

18.7 Configuring the type UI information

The type UI information view lets you specify which attributes display in Documentum clients and custom applications. Click the **Type** tab in the type editor to display the **Type UI Information** view.



To configure one or more attributes to display in clients, enter the type UI information as described in the following table:

Property	Description
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.
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.

Property	Description
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: • To add a tab, click **New**. The Display Configuration dialog appears. "Adding a tab" on page 198 provides information about how to add a tab to display an attribute in a client application. • 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 display, 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.

18.8 Adding a tab

Use the Display Configuration dialog to add a tab.

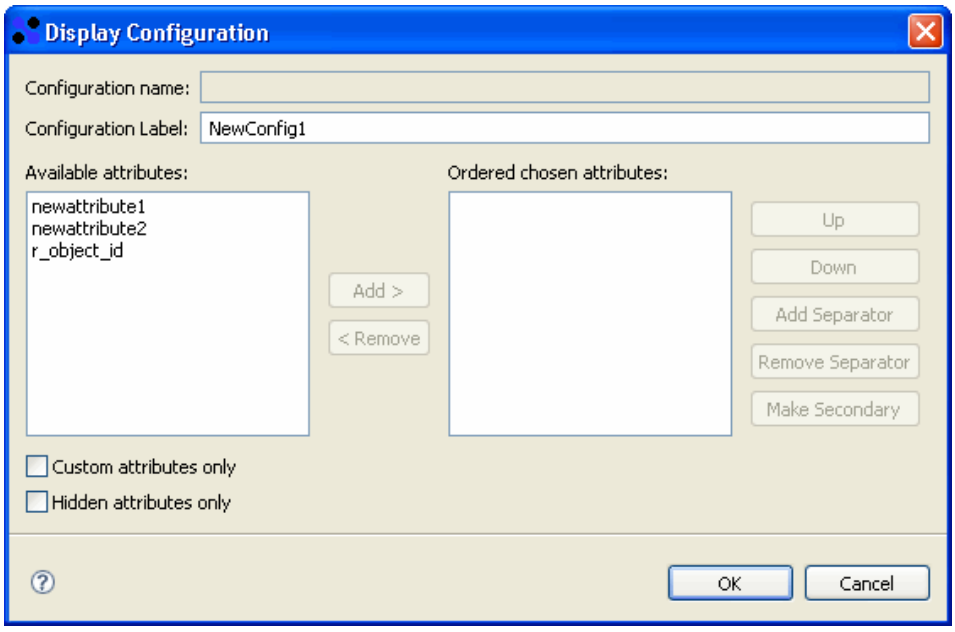
To add a tab:

1. Click **New** in the **Display Configuration List** section of the **Type UI Information** view. "Configuring the type UI information" on page 197 shows the Type UI Information view.

   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 the following table:

Tab properties	Description
Configuration name	A string that specifies the tab name. You can 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. "Adding type attributes" on page 187 provides information about configuring attributes.

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

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

Chapter 19

Managing XML Applications

19.1 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.

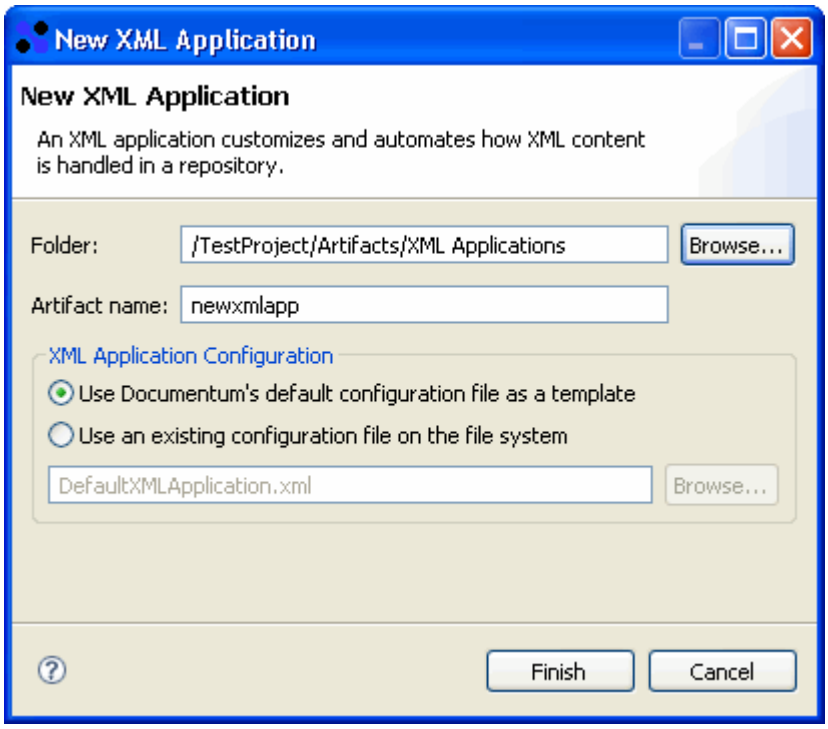
19.2 Creating an XML application artifact

Documentum Composer has a built-in editor that allows you to define XML application artifacts.

To create an XML Application artifact:

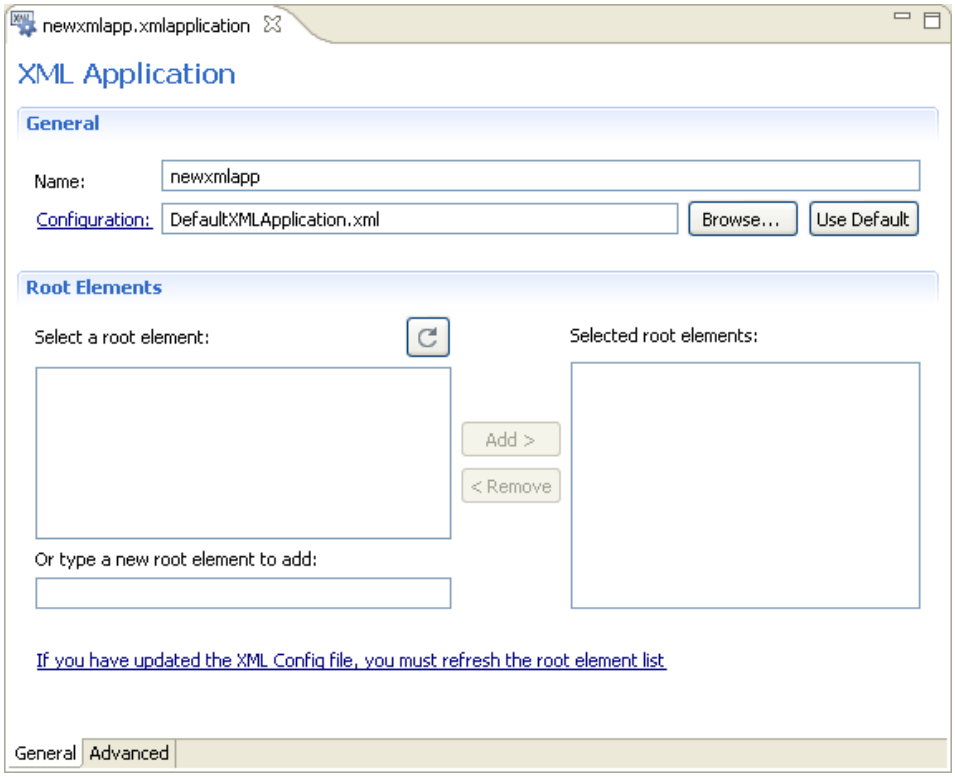
1. Right-click the **XML Applications** folder of your project in the **Documentum Navigator** view and select **New > XML Application**.

   The **New XML Application Artifact Wizard** appears.



2. In the **Artifact name** field, enter the name that you want to give to the XML Application artifact.
3. Do one of the following:
   - If you have an existing XML configuration file that you want to use, select **Use an existing configuration file on the file system** and then click **Browse** to find the file.
   - Select **Use Documentum's default configuration file as a template**.
4. Click **Finish**.

   The **XML Application** editor appears with the **General** tab selected.



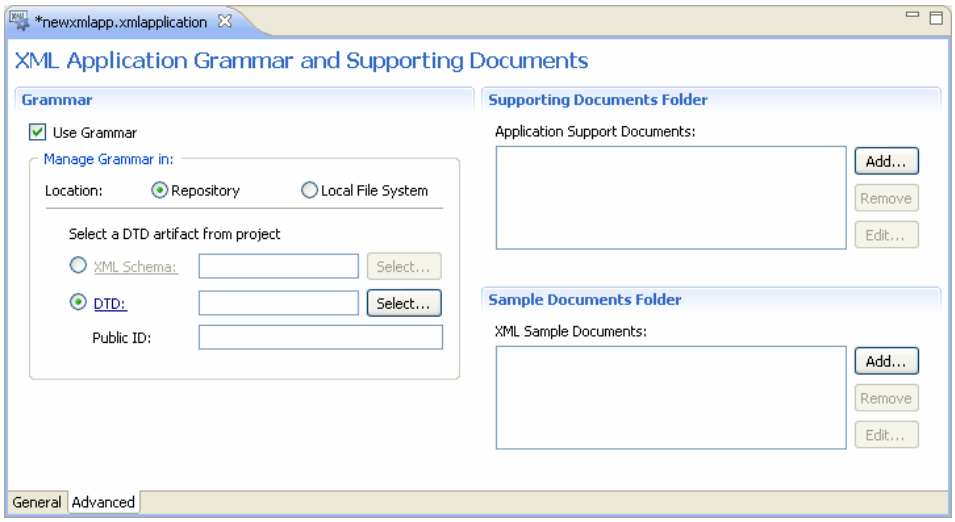
5. In the **Root Elements** section, add root elements that you want to process by doing one of the following:

   - Select a root element from the **Select a root element** list.
   - Enter a root element in the **Or type a new root element to add** field.

   When done, click **Add**.

6. Click the **Advanced** tab to specify a DTD or Schema to validate XML applications if you want to use one.

   The **XML Application Grammar and Supporting Documents** view appears.



7. In the **Grammar** section, do the following:

   a. Select **Use Grammar**.

   b. Select one of the following:

      - **Repository** to manage the DTD or Schema in the repository.
      - **Local File System** to manage the DTD or Schema on your local file system.

   c. Depending on what you want to use, select **XML Schema** or **DTD** and specify the XML Schema or DTD. To manage the XML Schema or DTD in the repository, the artifact must be in your project so that you can select it.

8. In the **Supporting Documents Folder** section, add any artifacts that your XML application needs.

9. In the **XML Sample Documents** section, add any sample XML documents.

19.3 Viewing or modifying an XML application configuration file

Documentum 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. However, the rule 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 to treat the linked file as a child or a peer of the XML virtual document, and if it 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

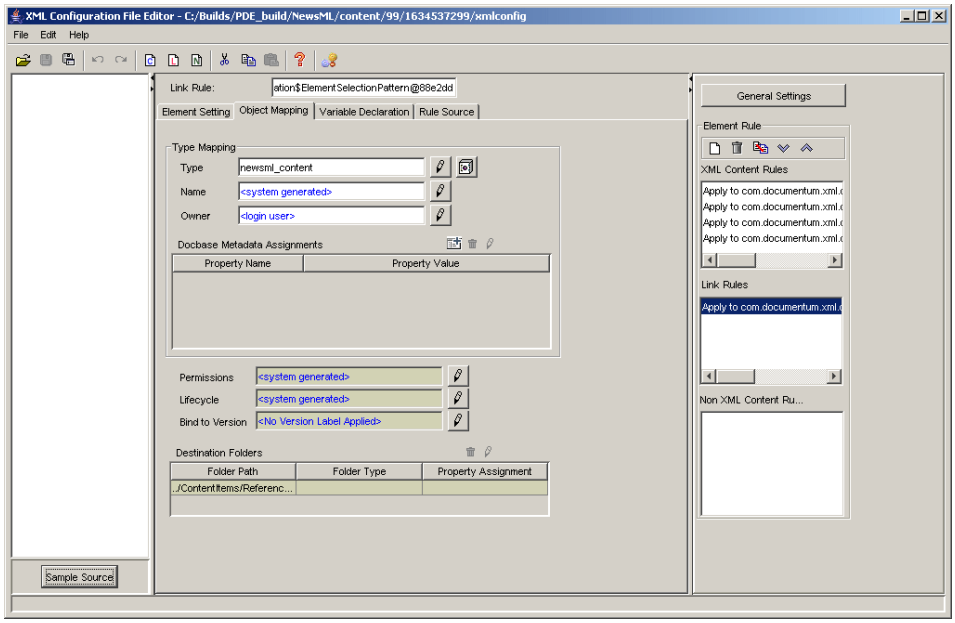
  An element rule or entity rule instructs the XML application to preserve all external parsed (XML) entities as separate components of the XML virtual document when imported or checked in, and to maintain their status as external entities when the main XML document is exported or checked out.

To view or modify an XML application configuration file:

1. In your project, expand the **XML Applications** folder.
2. Double-click the **.xmlapplications** file that you want to modify.

   The **XML Application** editor appears.
3. Click the **Configuration** link in the **General** section.

   The **XML Configuration File Editor** appears.



4. 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, make the same modifications in your project.

Chapter 20

Building and installing a project

20.1 Understanding the build and installation process

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

- Documentum Composer user interface

  The Documentum 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" on page 207 and "Configuring artifact install options" on page 211.

  You also have the option to build the application and generate a DAR archive file (.dar), as described in "Generating a DAR file" on page 213 for use with the DAR Installer or Headless Composer.

- DAR Installer

  The DAR Installer cannot build projects, but can install pre-built DAR files that were built with Documentum Composer or Headless Composer.

- 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 that is useful for automation. "Creating a Headless Composer build" on page 225 provides information about how to use Ant tasks and DAR files.

20.2 Configuring the project installation options

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

To configure project installation options:

1. In the Documentum Composer main menu right-click the name of the project to install and then select **Properties** from the drop-down menu.

   The **Documentum Project** dialog appears.

2. Specify the installation options for the project, as described in the following table:

Install Option	Description
Owner	Specifies the owner installation parameter to install 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. "Adding an owner installation parameter" on page 209 provides information about adding a new owner installation parameter.
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.

20.2.1 Adding an owner installation parameter

Use the Principal (User or Group) dialog to specify the owner installation parameter to install a project. The owner must be a valid user or group in the repository where the project is installed.

To add an owner installation parameter:

1. In the **Documentum Projects** dialog, click **Select** next to the **Owner** field.

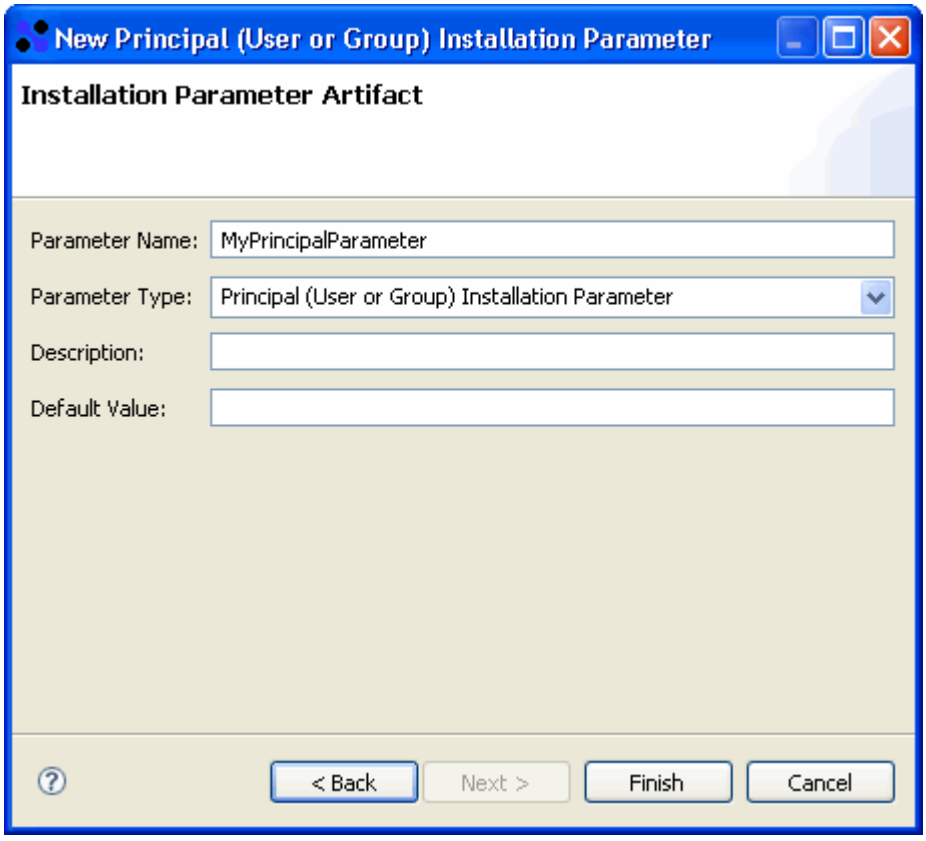
   The **Principal (User or Group) Installation Parameter** dialog appears.

2. Click **New**.

   The **New Principal (User or Group) Installation Parameter** 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:

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 **Principal (User or Group) Installation Parameter** 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**.

20.3 Configuring pre-installation 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.

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 procedure.

5. Click **OK**.

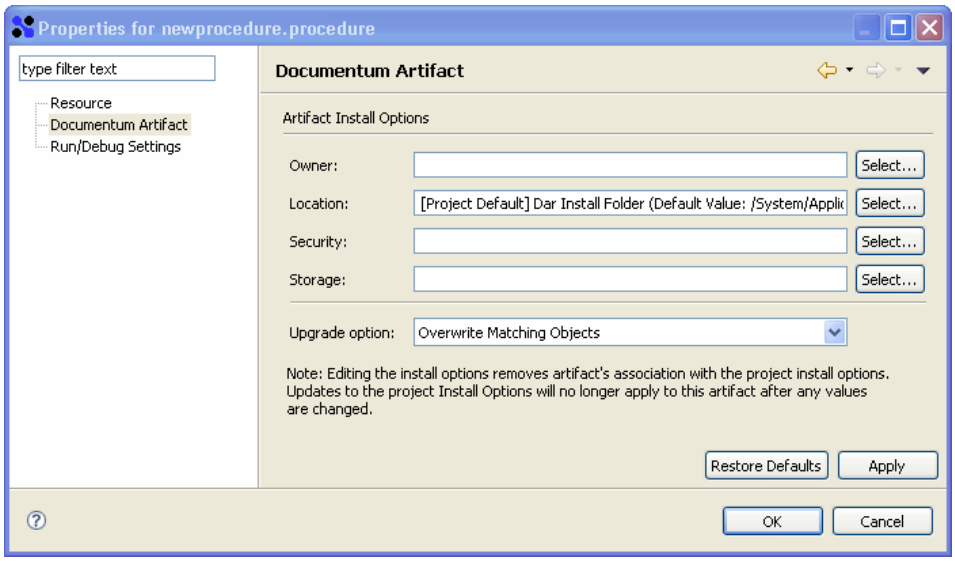
20.4 Configuring artifact install options

Documentum Composer lets you configure installation options at a project level and 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 the following table:

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. "Adding an owner installation parameter" on page 209 provides more information about adding a new owner installation parameter.
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.

Install Option	Description

Storage	Specifies the storage installation parameter. Click **Select** to select a storage installation parameter from the listbox.
	
	The storage parameter is available for a SysObject artifact and other artifacts that are a subtype of SysObject, such as Procedure, JAR Definition, and so on.
	
	The Storage field is not available for an artifact that is not a subtype of SysObject, such as Type. Documentum Composer sets the storage type for a type object during installation.
	
	By default, the project or DAR installation does not set the storage attribute. During installation, Documentum CM Server sets this value based on the storage policy stored in the repository. To overwrite the default behavior and set the storage type specified in the Properties page for each artifact, specify a system property: com.emc.ide.installer.enableStorage. Modify the composer.ini or darinstaller.ini to add a line for the Java VM argument. The following illustrates adding the system property to the composer.ini file:
	
	`-vmargs`
	`-Xms64m`
	`-Xmx512m`
	`-Dcom.emc.ide.installer.enableStorage`

Install Option	Description
Upgrade option	Specifies the upgrade option used when installing this artifact. There are three upgrade options: • **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 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:** To 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.

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

20.5 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 Documentum 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 Documentum Composer workspace. It is recommended that you leave this on in most situations. 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.
2. Select **Build Project** from the drop-down list.

   Documentum 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.

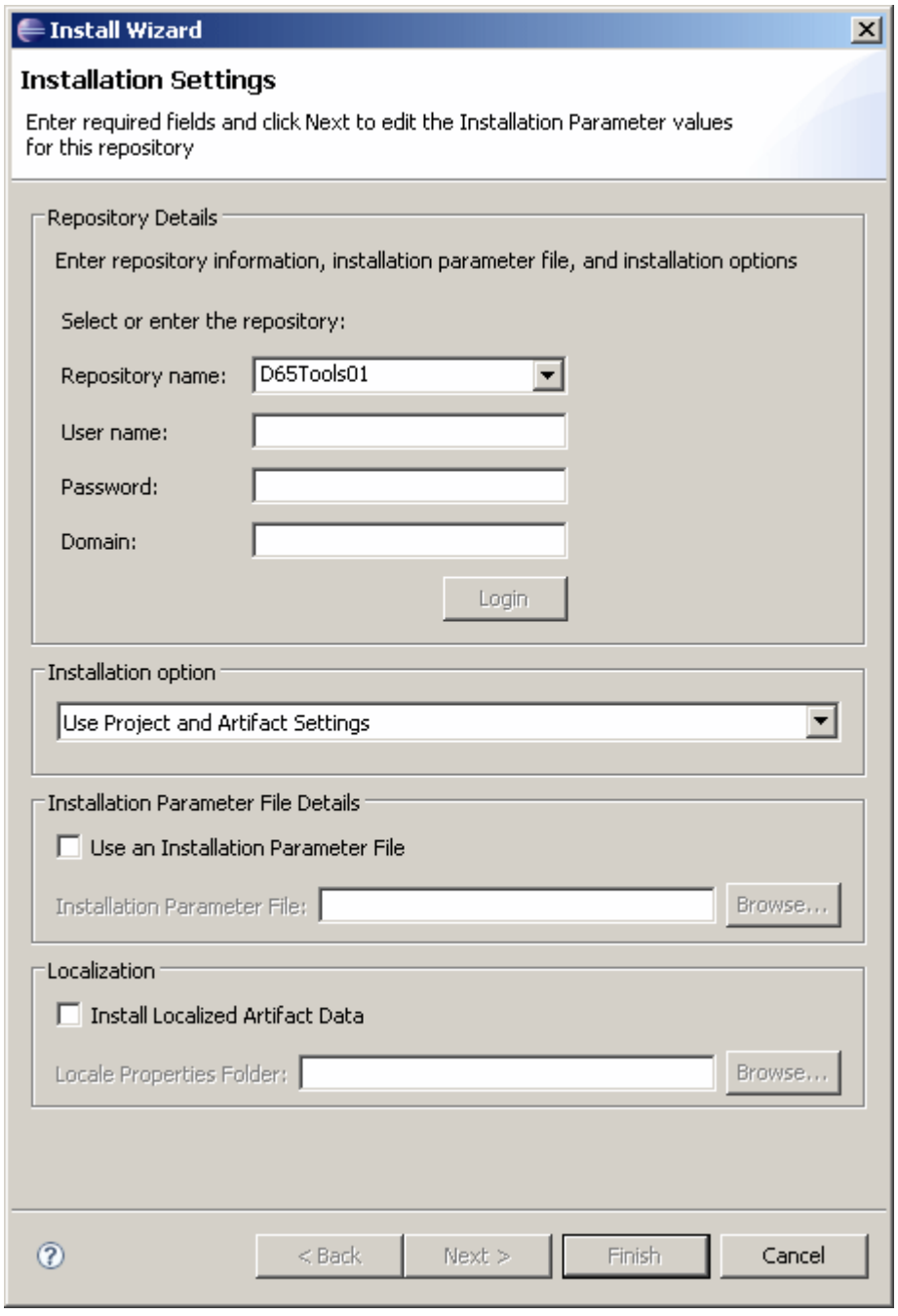
20.6 Installing a project

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

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. Documentum 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.

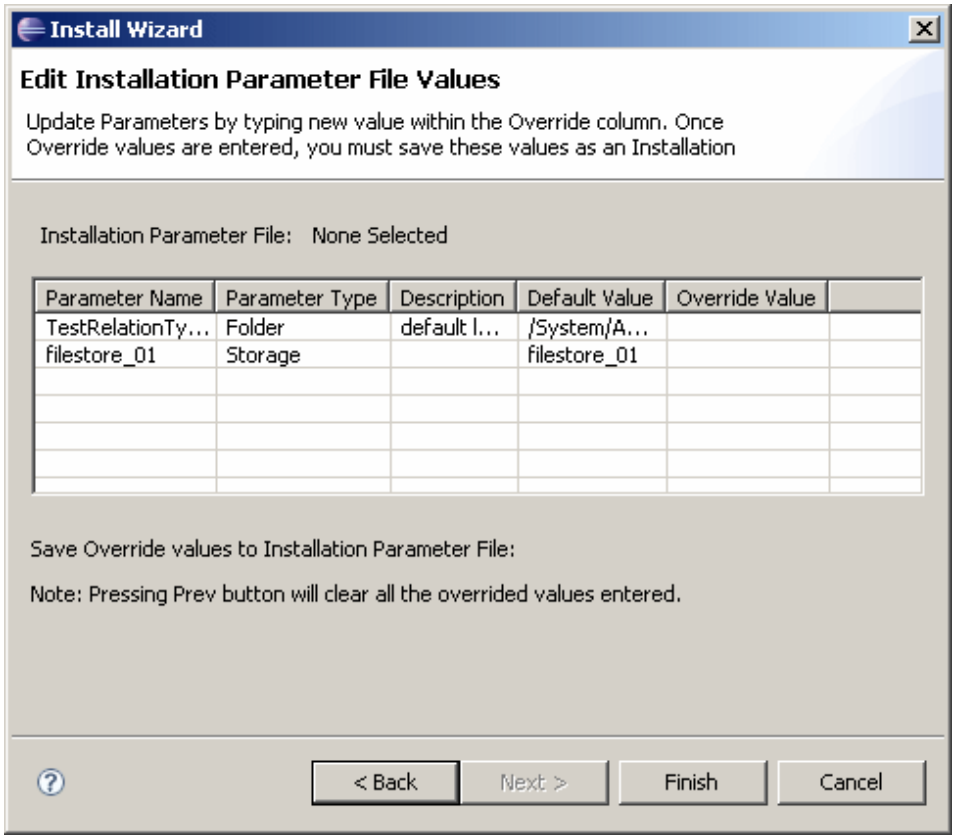


2. Enter the installation information, as described in the following table:

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. If the repository resides in a different domain than the client from which the repository is accessed, specify the domain name.
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 exists in the repository, all objects are overwritten when a modified version of the project is installed. • **Version**: If the project 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, create the input parameter file first, as described in "Creating an installation parameter file" on page 219.
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. "Localizing a Documentum Composer project" on page 30 provides information about localizing a project.

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. "Creating an installation parameter file" on page 219 provides information about creating an input parameter file.

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

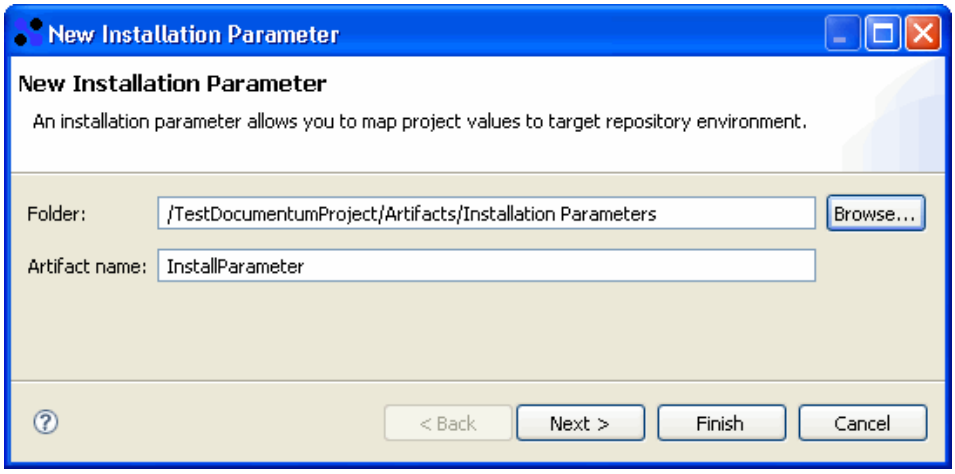
Note: If you install a project in the repository, then edit the project in Documentum Composer and remove one or more objects and re-install the project, the old objects remain in the repository. Documentum Composer does not delete objects from the repository during installation, even if the objects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

20.7 Creating an installation parameter

To create an installation parameter:

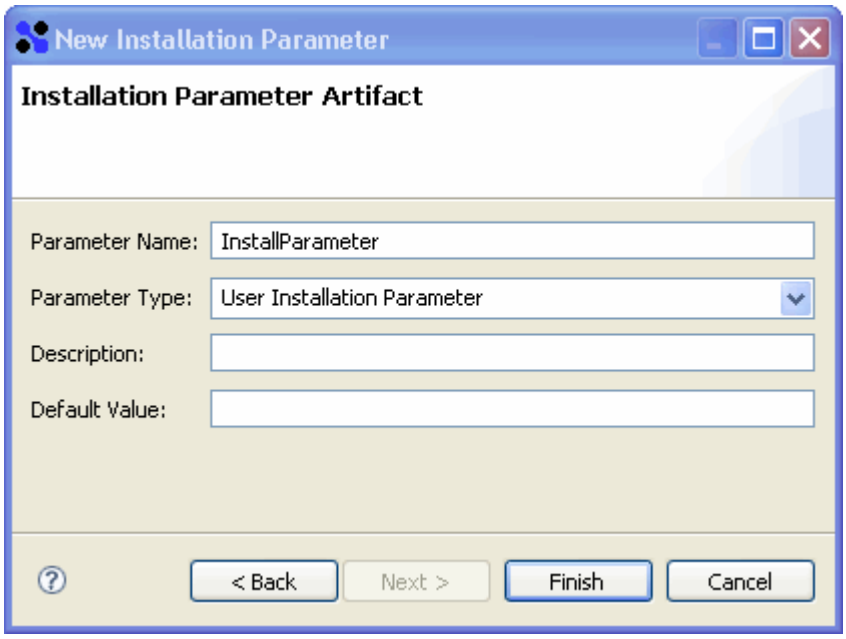
1. In your Documentum Composer project, expand the **Artifacts** folder and right-click **Installation Parameters**. Select **New > Installation Parameter**.

   The **New Installation Parameter** dialog appears:



2. Enter a name for the new installation parameter, then click **Next**.

   The **Installation Parameter Artifact** dialog appears:



3. Enter the properties as described in the following table:

Install Option	Description
Parameter Name	Displays the name of the installation parameter.
Parameter Type	Select an installation parameter type: • **User Installation Parameter**: Select to create an owner installation parameter to install a project. • **ACL Installation Parameter**: Select to create a permission set (ACL) installation parameter. • **Folder Installation Parameter**: Select to create a folder installation parameter. • **Storage Installation Parameter**: Select to create a storage installation parameter. "Configuring artifact install options" on page 211 provides more information about the storage installation parameter. • **Principal (User or Group) Installation Parameter**: Select to create the owner installation parameter to install a project. "Adding an owner installation parameter" on page 209 provides more information about adding a new owner installation parameter. • **Group Installation Parameter**: Select to create the owner installation parameter to install a project.
Description	Type a description of the installation parameter.
Default Value	Type a default value for the installation parameter.

20.8 Creating an installation parameter file

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

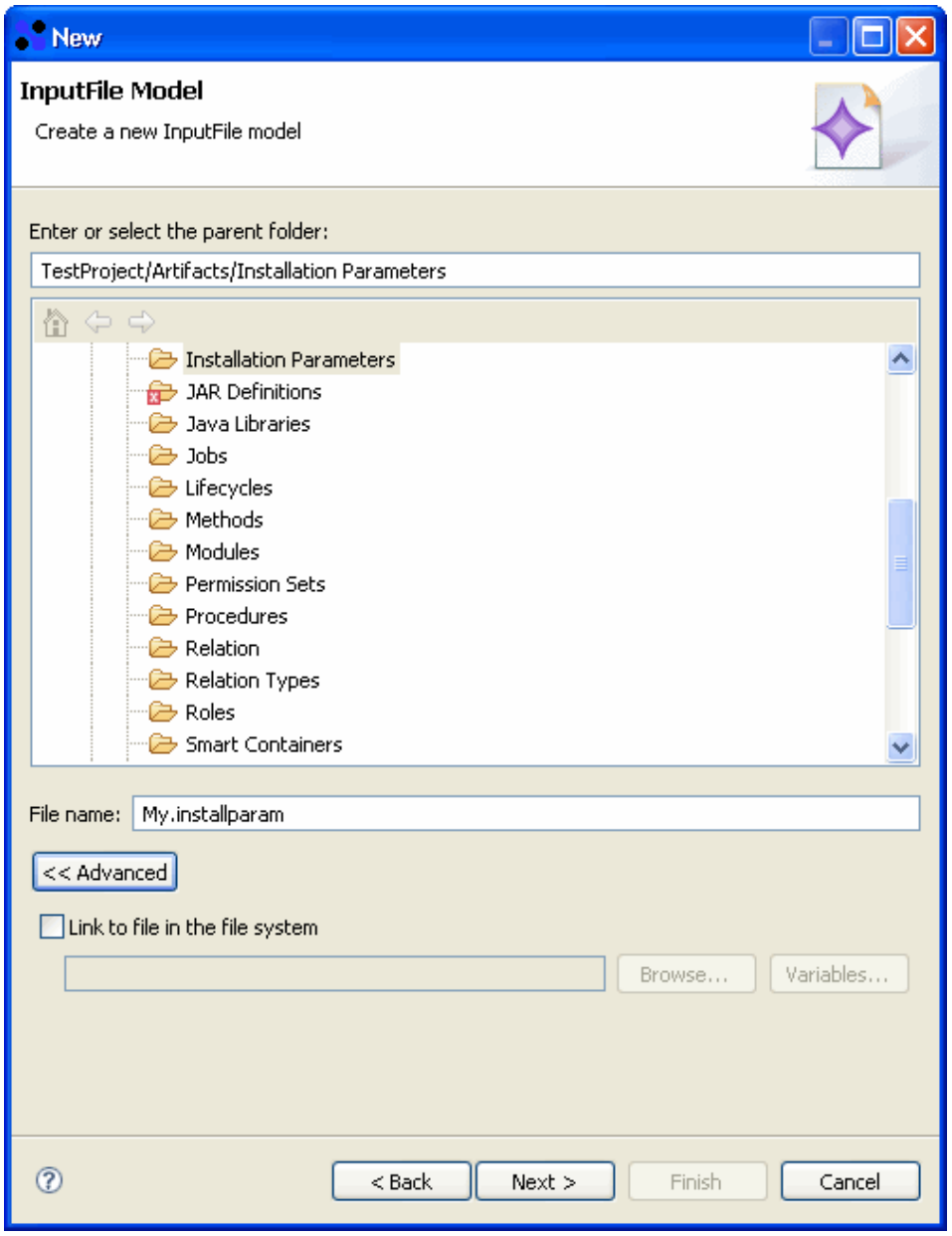
To create an installation parameter file:

1. In your Documentum Composer project, right-click **Installation Parameter Files**. Select **New > Other** from the drop-down menu.

   The **Select a wizard** dialog appears.

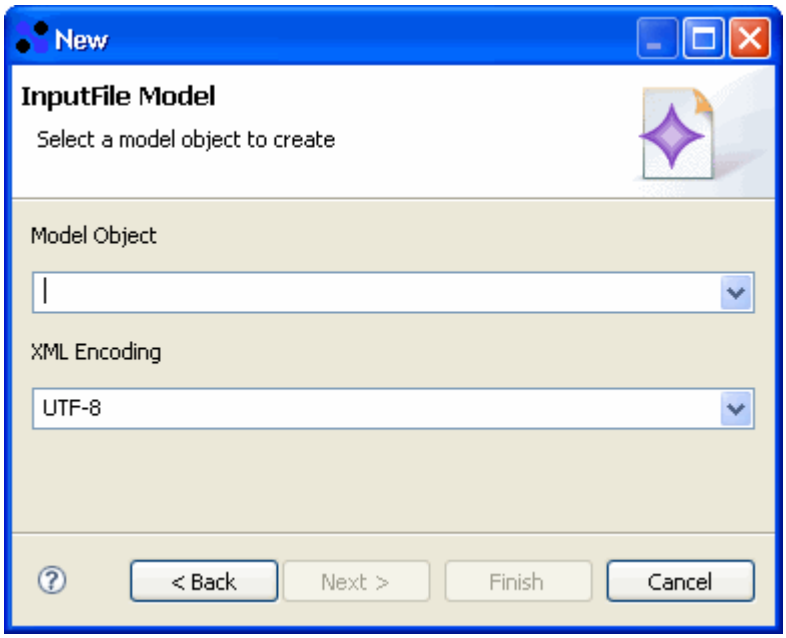
2. Double-click **Installation Parameter File** to expand it, select **Installation Parameter File New Wizard**, then click **Next**.

   The **InputFile Model** 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 secondary **InputFile Model** dialog appears.



4. In the **Model Object** drop-down list, select **Installation Parameter File**. In the **XML encoding** drop-down list, select **UTF-8**. Next, click **Finish**.

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

20.9 Installing a DAR file with the DAR installer

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

You can also install a DAR file with Headless Composer. "Creating a Headless Composer build" on page 225 provides more information on how to install DAR files with Headless Composer.

When you open the DAR Installer, it creates three folders in your Documentum 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 can 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.

To install a DAR file:

1. To start the DAR Installer, run dardeployer.exe, which is located in the Documentum Composer root directory.
2. In the **DAR Details** section, specify values for the fields.
3. In the **Connection Broker Details** section, specify values for **Connection Broker Host** and **Connection Broker Port** and click **Connect**.
4. In the **Repository Details** section, specify values for the fields.
5. In the **Logger** section, specify the required logging level 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**.

The following table describes the fields for the DAR Installer plugin:

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, 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.
Connection Broker Host	Yes	The address of the connection broker.
Connection Broker Port	Yes	The port of the connection broker repository.

Parameter	Required	Description
Repository	Yes	The name of the repository that you want to install the DAR file to. Click the Connect button after entering the connection broker host and port to retrieve the available repositories.
User Name	Yes	The login name for the repository.
Password	Yes	The password for logging in to the repository.
Domain	No	The domain of the user.
Error messages	No	The check box provides an option to add error messages that occurred during the DAR installation to a log file.
Warning messages	No	The check box provides an option to add warning messages that occurred during the DAR installation to a log file.
Debug messages	No	The check box provides an option to add debug messages that occurred during the DAR installation to a log file.
Trace messages	No	The check box provides an option to add trace messages that occurred during the DAR installation to a log file.

Chapter 21

Managing projects and DAR files using Ant tasks and Headless Composer

21.1 Creating a Headless Composer build

Headless Composer is the non-UI command line version of Documentum Composer that includes a set of Ant tasks for common build and deployment features of Documentum Composer, such as import, build, and install.

A Headless Composer build allows you to automate the build and installation of Documentum Composer projects. A Headless Composer build consists mainly of two parts: Ant scripts that define the build and a batch file that sets up the build environment and runs the Ant scripts.

21.1.1 Creating Ant scripts to build, modify, and install Documentum Composer projects

Ant scripts are XML files that define your build. Documentum Composer provides Ant tasks that allow you to call certain Documentum Composer functionality from an automated build.

In general, create two separate Ant build files that build your projects and install your projects. The Ant scripts should be encoded in UTF-8 to ensure proper functionality.

To create the Ant scripts:

1. Create a file named build.xml.
2. Create a target to import the projects that you want to work with into the Headless Composer workspace with the emc.importProject task. You can also create new projects with the emc.createArtifactProject task. You either have to create or import a project into the workspace before you call any other Ant task.
3. Make modifications that you want to the project with the emc.importArtifacts or emc.importContent tasks.
4. Create a target to build the project with the emc.build task. Call this task before the emc.dar task to ensure that the DAR file contains the latest built code.
5. Create a target to generate the DAR file with the emc.dar task. Call this task before the emc.install task to ensure that the code built from emc.build task makes it into a new DAR file that you can install later.
6. Create a file named install.xml.

7. Create a target to install the DAR file with the emc.install task.

build.xml file

```
<?xml version="1.0"?>
<project name="HelloWorldBuild" default="package-project">

    <target name ="import-project" description="
        Must import a project before updating, building, or installing it">
    <emc.importProject dmproject="HelloWorldArtifacts" failonerror="true"/>
    </target>

    <target name="update-jardef" depends="import-project" description="
        Update JARDef with most current JAR file">
    <emc.importContent dmproject="HelloWorldArtifacts"
        artifactpath="Artifacts/JAR Definitions/hello.jardef"
    contentfile="build_workspace/HelloWorldBOFModule/bin-impl/hello.jar" />
    </target>

    <target name="build-project" depends="update-jardef"
        description="Build the project">
        <emc.build dmproject="HelloWorldArtifacts" failonerror="true"/>
    </target>

    <target name="package-project" depends="build-project"
        description="Package the project into a DAR for installation">
        <delete file="HelloWorld.dar" />
        <emc.dar
            dmproject="HelloWorldArtifacts"
            manifest="bin/dar/default.dardef.artifact"
            dar="HelloWorld.dar" />
    </target>
</project>
```

```
<?xml version="1.0"?>
<project name="headless-install" default="install-project">
    <target name="install-project"
        description="Install the project to the specified repository.
        dfc.properties must be configured">
        <emc.install
            dar="HelloWorld.dar"
            docbase="GlobalRegistry"
            username="Administrator"
            password="emc"
            domain="" />
    </target>
    </project>
```

Note: If you are installing a DAR file that depends on other reference DAR files, install the reference DAR files first and in the same Ant script as the DAR file that you want to install. You must do this even if you previously installed the reference DAR files.

21.1.2 Creating a batch file to setup and run the build

The batch file is used to configure the environment variables and workspaces on your local machine, and calls the Ant scripts that contain the import, build, or install instructions for your build.

The following example batch file performs these actions:

- Defines the Eclipse directory path to be at `C:\ComposerHeadless`.
- Creates two workspaces, one for importing and building a project (BUILDWORKSPACE) and one for installing the resulting DAR file (INSTALLWORKSPACE).
- Specifies the location of the build and installation scripts: build.xml and install.xml.
- Cleans the workspaces.
- Copies the Documentum Composer projects into the build workspace.
- Runs the build and installation scripts. Ensure that you run the build using the supported version of JDK. The *OpenText Documentum Composer Release Notes* contains the supported Java JDK version.

Example batch file

```
REM Set environment variables to apply to this command prompt only
SETLOCAL

REM Sets the root location of headless Composer
SET ECLIPSE="C:\ComposerHeadless"

REM Sets the location of your source projects.
REM This location gets copied into your build workspace directory
SET PROJECTSDIR="C:\Documents and Settings\Administrator\composer-workspace"

REM Sets the workspace directory where Composer builds the projects
REM that you want to install to a repository
SET BUILDWORKSPACE="C:\ComposerHeadless\example build\build_workspace"

REM Sets the workspace directory where Composer extracts built DAR files
REM before installing them to a repository
SET INSTALLWORKSPACE="C:\ComposerHeadless\example build\install_workspace"

REM Sets the Ant script that builds your projects
SET BUILDFILE="C:\ComposerHeadless\example build\build.xml"

REM Sets the Ant script that installs your projects
set INSTALLFILE="C:\ComposerHeadless\example build\install.xml"

REM Delete old build and installation workspaces
RMDIR /S /Q %BUILDWORKSPACE%
RMDIR /S /Q %INSTALLWORKSPACE%

REM Copy source projects into build workspace
XCOPY %PROJECTSDIR% %BUILDWORKSPACE% /E

REM Run Ant scripts to build and install the projects
%JAVA_HOME%\bin\java.exe -cp %ECLIPSE%\startup.jar org.eclipse.equinox.launcher.Main -
data
%BUILDWORKSPACE% -application org.eclipse.ant.core.antRunner -buildfile %BUILDFILE%
%JAVA_HOME%\bin\java.exe -cp %ECLIPSE%\startup.jar org.eclipse.equinox.launcher.Main -
```

```
data
%INSTALLWORKSPACE% -application org.eclipse.ant.core.antRunner -buildfile %INSTALLFILE%
```

21.2 emc.importProject

The emc.importProject task imports a Documentum project into the Headless Composer workspace. A project must be imported into the workspace before you can build or install it.

Syntax

```
<emc.importProject
    dmproject="project_name" />
```

The following table describes the emc.importProject parameter:

Parameter	Description
dmproject	Specify the name of the project that you want to import.

21.3 emc.createArtifactProject

The emc.createArtifactProject task creates a Documentum Composer project.

Syntax

Elements and attributes in brackets are optional.

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

The following table describes the emc.createArtifactProject parameters:

Parameter	Description
emc.createArtifactProject task	The emc.createArtifactProject task specifies the project to create. It can also contain the projectReferences element that specifies reference projects for the project that you are creating.
name	Specify the name of the project that you want to create.
overwrite	Specify `true` if you want existing projects overwritten, `false` otherwise. The default is `false`.

Parameter	Description
projectReferences element (optional)	Specify this element to assign reference projects to the project. Reference projects must be imported into the workspace by using the emc.importProject command. The projectReferences element contains project elements that specify reference projects for the project that you are creating. A projectReferences element can contain multiple project elements.
failOnMissingReferences (optional)	Specify `true` if you want the build to fail if the reference projects are not available, `false` otherwise. The default behavior is set to `false`.
project element	The project element specifies the name of the project that you want to designate as a reference project. You can specify multiple project elements.
name	Specify the name of the project that you want to designate as a reference project. The project must already be imported into the Headless Composer build workspace with the emc.importProject task.

The following target creates a project named *Test*. It sets the overwrite attribute to true, which overwrites any existing project with that name. It defines *MyReferenceProject* 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.

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

21.4 emc.createTaskspaceApplicationProject

The emc.createTaskspaceApplicationProject task creates a Documentum Composer project from a TaskSpace application.

Syntax

Elements and attributes in brackets are optional.

```
<emc.createTaskSpaceApplicationProject name="project_name" docbase="repo_name"
username="username" password="password">
    [<projectReferences>
        <project name="reference_project_name"/>
    </projectReferences>]
</emc.createTaskSpaceApplicationProject>
```

The following table describes the emc.createTaskspaceApplicationProject parameters:

Parameter	Description
emc.createArtifactProject task	The emc.createTaskSpaceApplicationProject task specifies a project to create from an existing TaskSpace application in a repository. It gives the Documentum Composer project the same name as the TaskSpace application. You can also declare the projectReferences element to specify reference projects for the project that you are creating. Reference projects must be imported into the workspace before you can use them.
name	Specify the name of the project that you want to create.
docbase	The name of the repository where the TaskSpace application resides.
username	The username to login to the repository.
password	The password for the username.
projectReferences element (optional)	The projectReferences element contains project elements that specify reference projects for the project that you are creating. A projectReferences element can contain multiple project elements.Specify this element if you want to assign reference projects to the project. Reference projects must be imported into the workspace using the emc.importProject command.
project element	The project element specifies the name of the project that you want to designate as a reference project. You can specify multiple project elements.
name	Specify the name of the project that you want to designate as a reference project. The project must already be imported into the Headless Composer build workspace with the emc.importProject task.

```
<emc.createTaskSpaceApplicationProject name="TaskSpaceApp" docbase="repository"
username="dmadmin" password="n0lif3">
    <projectReferences>
        <project name="TSWorkflows"/>
    </projectReferences>
</emc.createTaskSpaceApplicationProject>
```

21.5 emc.importArtifacts

The emc.importArtifacts task imports artifacts from a repository into a specified project.

Syntax

Elements and attributes in brackets 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>
```

The following table describes the emc.importArtifacts parameters:

Parameter	Description
project	Specify the name of the project that you want to import the artifact into.
docbase	Specify the repository name where the artifact resides.
username	Specify the user to login to the repository with.
password	Specify the password to login to the repository with.
domain	Specify the domain of the user.
objectIdentities element	Contains id, path, or qualification elements that specify the identities of the artifacts to import. The objectIdentities element contains the identities of the object in the repository that you want to import. Each identity element has one attribute, value, that specifies the value of the identity.
id element	Used to specify the object ID of the artifact in the repository.
path element	Used to specify the path of the artifact in the repository.
qualification element	Used to specify the DQL qualifier for the artifact in the repository.

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>
```

21.6 emc.importContent

The emc.importContent task imports or updates a content file in a specified artifact and project. Currently, the emc.importContent command only supports importing content into procedure artifacts and JAR definitions.

Syntax

```
<emc.importContent
    dmproject="project_name"
    artifactpath="path/to/artifact"
    contentfile="path/to/content"
    contentid="content_id" />
```

The following table describes the emc.importContent parameters:

Parameter	Description
dmproject	Specify the name of the project containing the artifact into which the content is imported.
artifactpath	Specify the path name of the artifact relative to the project, including the name of the artifact itself, for example: Artifacts/JAR Definitions/myjardef.
contentfile	Specify the absolute file path of the content file to import. The file must exist in a readable format. If specifying a relative path, the path is relative to the location of the Ant script.
contentid (optional)	Specify 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, Documentum Composer generates and assigns a random name.

21.7 emc.build

The emc.build task builds a Documentum Composer project, which can then be passed into the emc.dar task to generate a DAR file.

Syntax

Elements and attributes in brackets are optional.

```
<emc.build
dmproject="project_name"
[failonerror="true|false]"/>
```

The following table describes the emc.build parameters:

Parameter	Description
dmproject	Specify the name of the project that you want to build.
failonerror (optional)	Specify true if you want the build to fail if the project does not build correctly, false otherwise.

21.8 emc.dar

The emc.dar task generates a DAR file from a Documentum project. The emc.dar task depends on the output from the emc.build task and must run in the same Java process. Define both tasks in the same Ant script and call them within the same build to ensure proper functionality.

Syntax

```
<emc.dar
  dmproject="project_name"
  manifest="bin/dar/project_name.dardef.artifact|default.dardef.artifact"
  dar="path/to/dar/dar_name.dar" />
```

The following table describes the emc.dar parameters:

Parameter	Description
dmproject	Specify the name of the project that you want to build a DAR file from.

Parameter	Description
manifest	Specify 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 case.
dar	The absolute file path to the target .dar file. The .dar file must not exist yet.

21.9 emc.install

The emc.install task installs a project's DAR file into a repository.

Syntax

Attributes in brackets are optional.

```
<emc.install
    dar="path/to/dar/dar_name.dar"
    docbase="repository"
    username="user"
    password="password"
    [domain="domain"]/>
    [inputfile="path/to/dar/filename.installparam"]
    [localesFolder="/path/to/locales/folder"] />
```

Note: If you are installing a DAR file that depends on other reference DAR files, install the reference DAR files first and in the same Ant script as the DAR file that you want to install. You must do this even if you previously installed the reference DAR files.

The following table describes the emc.install parameters:

Parameter	Description
dar	The absolute file path to the .dar file being installed. The file path must contain only Unicode (UTF-8) characters or the installation fails.
docbase	The name of the repository into which the .dar file is installed.
username	The login name for the repository.
password	The password for logging in to the repository.

Parameter	Description
localesFolder (optional)	The absolute file path to localized .properties files. If you want to make your application available in other languages, localize the project data, for example labels, tabs, and descriptions. "Localizing a Documentum Composer project" on page 30 provides information about localizing a project.
inputfile (optional)	The absolute file path to the install-based parameter file.
domain (optional)	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" />
```

21.10 emc.setUpgradeOption

The emc.setUpgradeOption task allows you to set upgrade options while installing the project file.

Syntax

```
<emc.setUpgradeOption dmProject="project_name" artifactPath="path/to/artifact"
option="OVERWRITE/VERSION/IGNORE">
```

The following table describes the emc.setUpgradeOption parameters:

Parameter	Description
dmproject	Specify the name of the project for which you want to set the upgrade options.
artifactpath (optional)	Specify the path name of the artifact relative to the project. It can be a path of an artifact file or a folder containing artifacts. For example, Artifacts/SysObjects/mysysobject.sysobject and Artifacts/SysObjects. If you specify the artifactpath, the upgrade option is applied to a specific artifact. Otherwise, the upgrade option is applied to a project.
option	Specify the upgrade options. The available options are: OVERWRITE, IGNORE, and VERSION.

This task allows you to set multiple upgrade options to VERSION an artifact and OVERWRITE another artifact. You can set these options for the entire project, an artifact set, or a specific artifact.

21.11 Installing a DAR file with Headless Composer on Linux systems

The Headless Composer distribution that is bundled with Documentum CM Server for Linux can be used to install a DAR file to 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 Linux systems:

1. Log in to the Documentum CM Server system as the owner of the repository where you want to install the DAR file.
2. Ensure that your environment variables are set according to *OpenText Documentum Composer Release Notes*. 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:

```
$JAVA_HOME/bin/java --add-modules java.se --add-exports java.base/
jdk.internal.ref=ALL-UNNAMED --add-opens java.base/java.lang=ALL-UNNAMED --add-
opens java.base/java.nio=ALL-UNNAMED --add-opens java.base/sun.nio.ch=ALL-UNNAMED --
add-opens java.management/sun.management=ALL-UNNAMED --add-opens jdk.management/
com.sun.management.internal=ALL-UNNAMED --add-opens=java.base/java.util=ALL-UNNAMED
--add-exports java.naming/com.sun.jndi.ldap=ALL-UNNAMED --add-opens java.base/
java.text=ALL-UNNAMED --add-opens java.desktop/java.awt.font=ALL-UNNAMED --add-
opens java.naming/com.sun.jndi.toolkit.url=ALL-UNNAMED --add-exports=java.base/
sun.security.tools.keytool=ALL-UNNAMED -Ddar=$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.equinox.launcher.Main -data
$DM_HOME/install/composer/workspace -application  org.eclipse.ant.core.antRunner -
buildfile $DM_HOME/install/composer/deploy.xml
```

Note: If you are installing a DAR file that depends on other reference DAR files, you cannot use the supplied `deploy.xml` script. Create your own Ant script that contains targets to install the reference DAR files first and then install the DAR file. These targets must be in the same Ant script and must be run in the same call to Ant.

Chapter 22

Working with Source Control Systems

22.1 Using a source control system

If a team of developers is working on the same Documentum Composer project and other referenced projects, they often use a source control system to enable collaboration. Most source control systems offer Eclipse-based plugins that support an integrated development environment. These plugins can be used with Documentum Composer, as long as the plugins are compatible with Eclipse.

Note: You cannot open a Documentum Composer project in a source control system that is read-only.

22.1.1 Checking in projects

The following guidelines are recommended when using a source control system:

- Check in the project, not the Documentum 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.

22.1.2 Checking out and importing projects

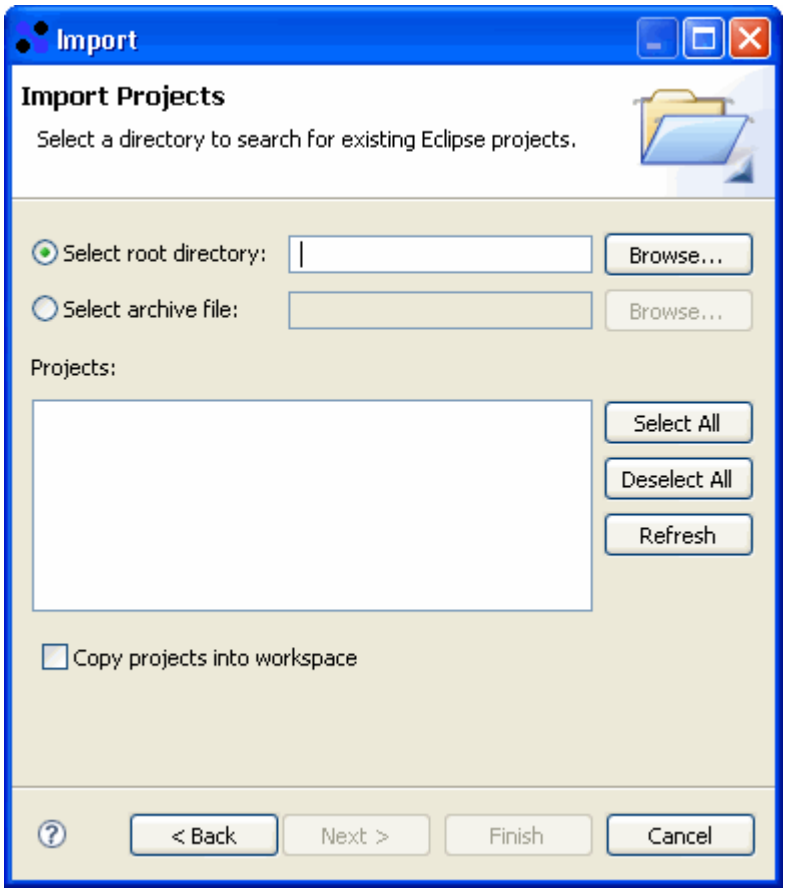
When working with a source control system, import the projects into a Documentum Composer workspace after retrieving them from the source control system.

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 Documentum Composer, navigate to **File > Import**.

   The **Import - Select** dialog appears.
3. Double-click **Documentum**, select **Existing Projects into Workspace**, and then click **Next**.

   The **Import Projects** dialog appears.



4. 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.

Notes

- If you do not see the projects that you want to import in the **Projects** list-box, verify that you do not have a previous version of the project in your Documentum Composer workspace. You can only import projects that do not exist in your local Documentum Composer workspace.
- The **Select archive file** option is not supported in Documentum Composer 7.2 and later.

5. Select the projects to import then click **Finish**.

   Do not select the **Copy projects into workspace** option.

22.2 Building the project

Building a project is analogous to compiling code. During the build process Documentum Composer validates the artifacts and reports any validation errors. The build process can be initiated in one of two ways:

- From the Documentum Composer UI
- Using Headless Composer and Ant tasks

  Using ANT tasks to build the DAR file is the more efficient option if the Documentum 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 "emc.build" on page 233.
- When you want to install DAR files on S3, you must set the dmc_dar type to S3 on the Documentum CM Server.

  If filestore_01 is mentioned in the dmc_dar.type.artifact, then it takes precedence over S3store and is installed in the filestore.

Chapter 23

Frequently asked Documentum Composer questions

23.1 General questions

- Can I use Documentum Composer with a Documentum 6.0 or 5.3 repository?

  Yes, you can use Documentum Composer with a Documentum 6.0 or 5.3 repository as long as you are not trying to install BPM artifacts or artifacts that leverage Documentum functionality, such as Smart Container (introduced after Documentum 6.0) or Aspects (introduced after Documentum 5.3). The latest version of Documentum Composer supports Documentum versions 5.3 SP6 and later.

- Can I use Documentum Composer to migrate cabinets, folders, or content from one repository to another?

  We do not recommend using Documentum Composer in this fashion because Documentum Composer was designed for application development and not data migration. Therefore, if you have a lot of cabinets, folders, and content, we cannot guarantee good migration performance. If the folders and content are related to application development and setup, such as an XML Application Folder with supporting documents, it is acceptable to import those into a Documentum Composer project.

- Do you offer Documentum Composer as a set of plugins so that I can install Documentum Composer plug-ins on top of my version of Eclipse?

  No, for supportability reasons and ease of installation, Documentum Composer is available only as an entire package with Eclipse embedded in it.

- Do you support Linux?

  We support using Headless Composer on Linux with a caveat. We do not ship a separate version of Documentum Composer for Linux. The Documentum CM Server Installer includes a version of Headless Composer that has been configured with the correct environment variables for this operating system.

- What is a Documentum supplied reference project? Why do I need it?

  Documentum supplied reference projects are non-buildable projects that allow you to use or extend artifacts whose names begin with 'dm' (Documentum artifacts). Documentum Composer does not allow you to have 'dm' artifacts in your projects to prevent unintentional changes to Documentum artifacts in the repository. You must designate the appropriate reference project to use or extend a 'dm' artifact.

- Can I install individual artifacts?

Currently, artifacts must be installed as part of a Documentum Composer project.

- Do you support uninstallation of Documentum Composer Projects or DARs?

  No, because Documentum CM Server does not support the uninstallation of certain artifacts such as Types. If you are in a development environment where you can experiment, we recommend you to use VMWare so you can roll back any changes, if required.

- Can I still use Docbasic pre- and post-install scripts?

  Yes, pre- and post-install scripts are specified in the Documentum Project Properties. To specify the scripts, right-click the Documentum Composer project in the **Documentum Project > Install Procedures** section.

- Can I use Documentum Composer to develop applications for the High-Volume Server?

  Yes, a Lightweight SysObject Object Plug-in is provided as a separate download that you can install on top of Documentum Composer.

- How do I find out the version of Documentum Composer I am running?

  From the **Help** menu, select **About Documentum Composer**. The **About** dialog box appears. You can find the version of Documentum Composer here.

23.2 DAR files

- What is a DAR file?

  A DAR file is a packaged output file of a Documentum Composer project. It contains artifacts that you install to a repository with the DAR Installer plug-in or Headless Composer, but does not contain the source code for artifacts. An analogy would be a JAR file compared to a Java source code.

- How do I install DAR files?

  Use the DAR Installer or Headless Composer with Ant tasks. The Documentum Composer UI cannot install DAR files.

- My DAR installation fails with the error “Unzipped resource must exist (<artifact path\artifact name>)”. What is wrong?

  The unzipped artifact file path length has exceeded the allowed length in Windows. You can reduce the file path length by installing Documentum Composer to the system root folder or to a folder that doesn't have a long path.

- I try to launch the DAR Installer UI by running dardeployer.exe and I get errors. What is wrong?

  – You do not have a compatible Java environment installed or JAVA_HOME is not set. You can check this by running the 'java-version' command on the command line. If the system does not recognize the Java command or the version is not compatible, install the supported version of Java and set the JAVA_HOME environment variable to point to the Java installation.

– The startup.jar file is missing in the Documentum Composer directory or dardeployer.exe is running from a wrong directory. You must run dardeployer.exe from the Documentum Composer root folder. If startup.jar is missing, reinstall Documentum Composer. The following command either starts the DAR Installer or produces an error: java\-cp startup.jar com.emc.ide.installer.darinstaller.starter.DarInstallerStarterMain.

23.3 Lifecycles

- How do I validate lifecycles?

  In Documentum Composer, lifecycles are validated automatically at install time. There is no need to manually validate a lifecycle in Documentum Composer. If the lifecycle fails validation, an error is displayed and the installation of the Documentum Composer project fails.

- How do I uninstall lifecycles?

  We do not support uninstallation of lifecycles, but we do support deactivating a lifecycle. To deactivate a lifecycle, check the **Inactivate lifecycle** check box that is in the **Overview** tab in the **Properties** window and re-install the Documentum Composer project.