Distributed Configuration Guide

Version 5.3 SP1
September 2005

Table of Contents

List of Figures

List of Tables

Preface

This manual describes the Content Server features that support the Documentum distributed configurations. The manual contains information to help administrators determine which features and configurations best meet the needs of the business and procedures to implement and manage those features and configurations.

Intended audience

This manual is written for system or superusers who are responsible for the implementation and maintenance of a Documentum distributed environment.

Conventions

This manual uses the following conventions in the syntax descriptions and examples.

Syntax conventions

Convention	Identifies
italics	A variable for which you must provide a value.
[ ] square brackets	An optional argument that may be included only once
{ } curly braces	An optional argument that may be included multiple times

Terminology changes

Two common terms are changed in 5.3 and later documentation:

- Docbases are now called repositories, except where the term “docbase” is used in the name of an object or attribute (for example, docbase config object).
- DocBrokers are now called connection brokers.

Revision history

The following changes have been made to this document.

Revision history

Revision Date	Description
September 2005	Initial publication

Chapter 1

Distributed Models

This chapter describes the most common distributed configurations. Use the information in this chapter, in conjunction with the information in Chapter 2, Building Blocks, to help you decide which distributed model or models best suits your business needs. The chapter includes the following topics:

- Overview of Models, page 13, which lists the common distributed models for a single repository and multiple repositories.
- Single-repository distributed models, page 14, which describes, in text and illustrations, the common models for a single repository.
- Multi-repository distributed models, page 19, which describes, in text and illustrations, the common models for multiple repositories.

Overview of Models

A distributed configuration can be set up for either a single repository or multiple repositories. In a single-repository distributed configuration, content is distributed across all sites configured to access the repository. In a multi-repository distributed configuration, objects (content plus metadata) are distributed across all participating repositories.

There are two common models for single-repository distributed environments:

- A single repository, with content stored at the primary site and accessed from remote sites using an Accelerated Content Services (ACS) server and, optionally, Branch Office Caching Services (BOCS) servers
- A single repository, with content stored in a distributed storage area and accessed from remote sites using content-file servers, ACS servers, and optionally, BOCS servers

There are also two common models for multi-repository distributed configurations:

- Multiple repositories that replicate objects among themselves
- Multiple repositories organized as a federation

An enterprise can use one model or a combination of the models. For example, a business might have one repository that contains primarily material, such as SOPS, that users only need to read. That repository may be configured to use BOCS servers for its remote users. Another repository may store business and sales proposals that are written and updated frequently. That repository uses distributed storage areas with ACS servers or Desktop clients for remote users. The two repositories may also be tied together in a federation, to ensure that the users and groups in both repositories are the same.

Single-repository distributed models

In a distributed single-repository, the distributed data is content. Users in many locations want fast access to the documents, that is, content, in the repository. EMC Documentum supports distributed content models for both web-based access and Desktop access.

Note: The figures in this section depict geographic locations, not individual machines.

Single Model 1: Single repository with content persistently stored at primary site and accessed using ACS or BOCS servers

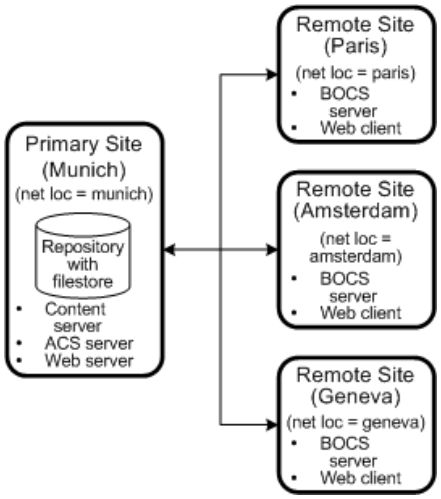
In this model, remote users connect through web browser, using a WDK-based client application. They access content stored at the primary site, through either an ACS or BOCS server. The ACS server is a content server dedicated to serving content. It does not process metadata nor write content to storage. It only processes content requests. The BOCS server is a separate product. It is a caching server that communicates only with ACS servers. Like the ACS server, it does not handle metadata requests or write content to storage. Both the ACS and BOCS servers use HTTP or HTTPS protocol to serve content to clients. Single model 1 is the preferred model when remote users are accessing repository content through a web-based application, such as Webtop.

There are two alternative configurations for this model. The configurations differ in how users at remote sites access the content at the primary site. In the first alternative, remote sites have a BOCS server installed and clients at each remote site use that BOCS server to access content. In the second basic configuration, users at remote sites have only web clients, and they access content using the ACS server at the primary site.

Figure 1–1, page 15, illustrates the first alternative.

Note: The figure depicts geographic locations, not individual machines.

Figure 1-1. Alternative 1: BOCS servers at remote sites communicating with primary site

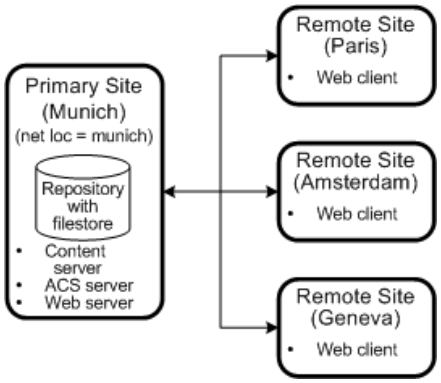


In this example of alternative 1, users at each remote site use a BOCS server to request content. When users in the Paris branch office request a document, their requests are handled by the BOCS server installed at the Paris branch office. Similarly, when users in the Amsterdam branch office request a document, their requests are handled by the BOCS server in the Amsterdam branch office, and requests from Geneva users are handled by the BOCS in the Geneva branch office.

The BOCS server is a caching server. When the BOCS server receives a request for content, it first checks the cache that it maintains. If the content is in the cache, the BOCS server provides that content to the user. If the content is not in the cache, the BOCS server communicates with the ACS server at the primary site to locate the content. The ACS server, in turn, communicates with the web server and Content Server at the primary site.

The second basic configuration is illustrated in Figure 1–2, page 15. In this configuration, the remote users have no installed software other than the web browser.

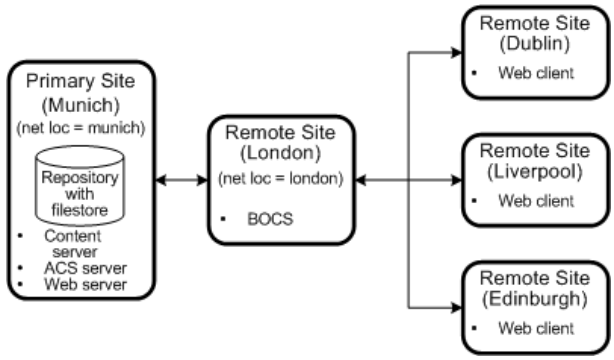
Figure 1-2. Alternative 2: Remote sites, without BOCS servers, using primary site's ACS server



In the second alternative, when users at a remote site request a content file, the request is handled by the ACS server at the primary site.

Figure 1–3, page 16, shows a variation of the configurations that combines the two. In this example, the remote clients are telecommuters, working from their web browsers on home machines. When they request documents, the request is handled by a BOCS server installed at the branch office closest to their location. (This figure depicts geographic locations, not individual machines.) All metadata requests and write requests are handled by the Content Server at the primary site.

Figure 1-3. The two alternative for single model 1 combined



In each of the configuration alternatives, each ACS or BOCS server is defined as a network location. When users begin a web-based session, the client application determines which network locations are available to the user. Then, the users are either automatically assigned to a network location or they may choose from those available, depending on how the client applications are configured. The network location with which the session is associated determines which ACS or BOCS server handles the users' content requests.

Benefits and best use

This model requires the least amount of administrative overhead. It is easy to set up and configure, and ongoing administration needs are minimal. There are no content replication jobs to define, administer, and run.

This model is not available for users on Desktop clients.

Single Model 2: Single repository with content in a distributed storage area

In this model, content is stored in a distributed storage area. A distributed storage area is a storage area with multiple component storage areas. One component is located at the repository's primary site. Each remote site has one of the remaining components. Each site has a full Content Server installation (a content-file server, also called a content-file server or RCS) and an ACS server installation for the repository. Content is replicated from its source component to the remaining components by user-defined content replication jobs.

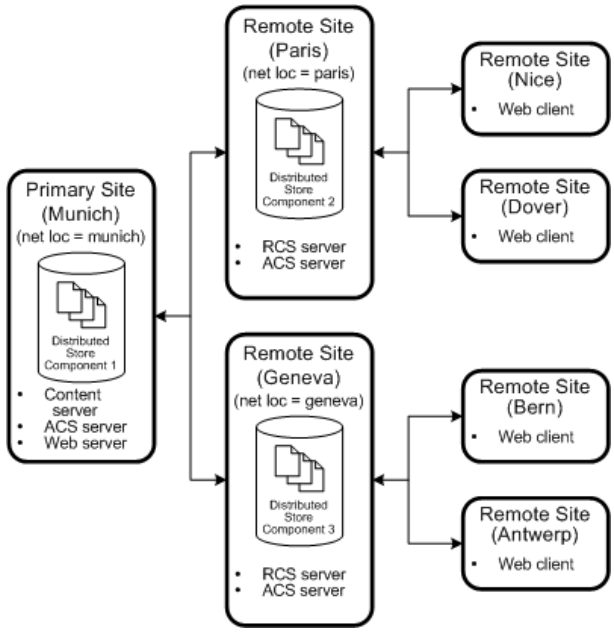
This model can be used for either web-based clients or Desktop clients.

If a remote site is using web-based clients, the site should configure the use of the ACS server or a BOCS server to access content, to provide optimal performance for the web-based users. Desktop clients use the Content Server at the remote site to access content. Figure 1–4, page 17, illustrates this model using web-based clients.

In this configuration, metadata requests are handled by the Content Server at the primary site and requests to write content to storage are handled by the Content Servers (RCS) at the remote sites.

Note: This figure depicts geographic locations, not individual machines.

Figure 1-4. Single Model 2: Single repository with a distributed storage area



In this model, users in the small branch offices in Nice and Dover access the content stored in the distributed storage component at the larger Paris branch office. Similarly, users in the small branch offices at Bern or Antwerp access content stored in the larger Geneva branch office. If the users are logging in using a web-based client, content requests are handled through the ACS server at the appropriate branch office in Paris or Geneva. If the users are logging in using a Desktop-based client, content requests are handled by the Content Server in Paris or Geneva. (The graphic shows only users logging in using web browsers.)

In this model, the remote sites using web-based clients could have BOCS servers, rather than just a web browser, to contact the branch offices for content requests.

Benefits and best use

For sites using web-based clients, this model is best if model 1 is not acceptable for any particular reason. For sites using Desktop clients, this model is the only model available for a single-repository distributed configuration.

UCF mode and HTTP mode

Configurations that use an ACS or BOCS server at a remote site require the WDK Webtop application on the web application server to be running in UCF mode. HTTP mode is supported only for local area network configurations, those having no ACS or BOCS server at remote sites.

For more information

Distributed storage areas, page 26, describes distributed storage areas and their configuration requirements.

Distributed storage areas, page 39, describes the internal implementation of a distributed storage area.

ACS servers, page 25, describes the ACS server and its configuration requirements

BOCS servers, page 25, describes the BOCS server and its configuration requirements

Network locations, page 24, describes network locations and their configuration requirements

Communication flow for web-based models, page 38, describes how the web-based models are implemented internally

Multi-repository distributed models

In multiple repository distributed models, entire objects, both content and metadata, are distributed between repositories. The distribution can occur through user-defined object replication jobs, or internally, when a user manipulates objects from multiple repositories in one repository session. For example, an internal replication occurs if a user starts a repository session and opens his or her Inbox and, from there, opens a document sent from another repository.

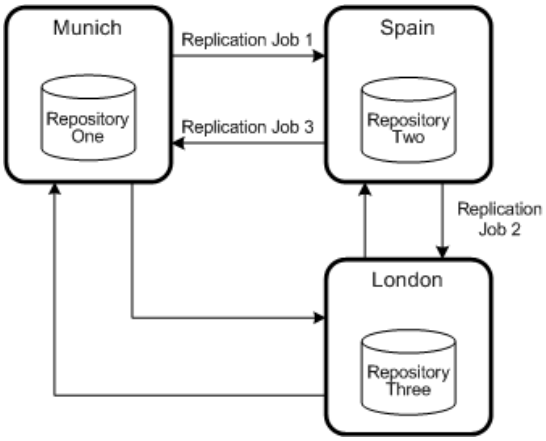
This section discusses the basic multi-repository replication model and the federation model. Both models are based on object replication. The federation model provides system-defined jobs that automate much of the administration work required to ensure that object replication works correctly.

Multiple repositories using object replication

Object replication replicates objects, both content and metadata, between repositories. Object replication jobs are user-defined. In object replication, there is a source and target repository. A replication job replicates objects from the source repository to the target repository. Which objects are replicated and how often the job runs is part of the job's definition. In the target repository, the replicated objects are marked as replica objects.

Figure 1–5, page 19, illustrates object replication.

Figure 1-5. Object replication model architecture



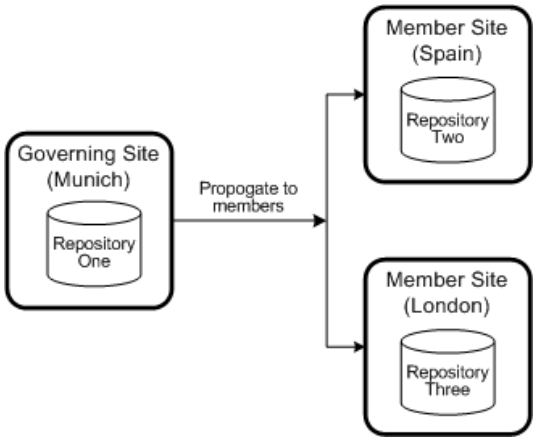
Multiple repositories working as a federation

When sites have multiple repositories and users are accessing objects from those repositories in one session, maintaining consistent security is imperative. The definitions of users, groups, and ACLs should be the same across the repositories. Keeping users, groups, and ACLs synchronized in multiple repositories can be time consuming and error prone if the work is done manually in each repository. Placing the repositories in a federation automates much of the process.

In a federation, one repository is the governing repository. The remaining repositories are member repositories. Changes to global users and groups and external ACLS in the governing repository are propagated from the governing repository to all the member repositories automatically.

Figure 1–6, page 20, illustrates a federation.

Figure 1-6. Federation Model



For more information

Object replication, page 32, has a more complete overview of object replication and its configuration requirements.

Object replication, page 44, has complete information about object replication and how it works

Federations, page 34, has more information about federations and information about their configuration requirements.

Federations, page 54, describes how federation are implemented internally.

Chapter 2

Building Blocks

This chapter describes the Content Server features that are the building blocks for the common distributed configurations, and the architecture, or implementation, of each building block. Use the information in this chapter, in conjunction with the model descriptions in Chapter 1, Distributed Models, to help you decide which distributed model best suits your business needs. The chapter includes the following topics:

- Building blocks, page 23, which introduces the building block features that are used to implement the distributed models and describes benefits, best use, and configuration requirements for each
- Building block architectures for single-repository models, page 36, which describes the implementation of the features that support single-repository distributed models
- Building block architectures for multi-repository models, page 43, which describes the implementation of the features that support multi-repository distributed models
- Distributed environments and the secure connection defaults , page 56, which describes how the secure connection options must be configured in either distributed environment.

Building blocks

This section describes the features that are the building blocks for implementing distributed configuration models. The following building blocks are described:

- Network locations, page 24
- ACS servers, page 25
- BOCS servers, page 25
- Distributed storage areas, page 26
- Content-file servers, page 28
- Shared content, page 29
- Content replication, page 30
- Reference links, page 31
- Object replication, page 32

- Federations, page 34

Network locations

Network locations are a basic building block of a single-repository distributed environment for web-based clients. Network locations represent a place or area on network's topography. A network location can represent a branch office, a set of telecommuters in the same geographical area, or any set of users you choose to aggregate as a network location.

Network locations typically identify one or more IP addresses or address ranges. The addresses generally represent machines that are close to each other. For example, all machines in one branch office may be defined as one network location. Another network location may represent all users in the Western United States or in Eastern Europe. The geographic size of a network location is completely up to the administrator configuring the system.

Benefits and best use

Network locations are useful only when configuring a single-repository distributed environment for web-based clients. Network locations are not used by Desktop clients.

By defining network locations, users connecting to a repository through a web-based Documentum client are automatically connected to the closest server for content requests. This provides enhanced performance when the users access documents and any other object with content.

Configuration Requirements

To use network locations, the following configuration requirements must be met:

- You must designate one repository as the global registry and the network location definitions must be stored in that repository.
- Proximity values must be defined for each network location.

  The proximity value defines the network location's proximity to a Content Server or ACS server. This information is used to ensure that content requests are handled by the server closest to the user.

ACS servers

The ACS, or Accelerated Content Services, server is a light-weight server that serves content to web-based client applications. It uses HTTP or HTTPS protocol to provide content to client applications quickly. Each Content Server host installation has one ACS server. That ACS server communicates with one Content Server per repository installed on the host installation.

The ACS server is installed automatically when the first repository on the installation is configured. If additional repositories are added to the installation, the ACS server's configuration information is updated so that the server can communicate with the Content Server for the new repository.

If you install a remote site with a content-file server, the installation at that site has its own ACS server.

Benefits and best use

The ACS server serves users who are accessing the content through web-based client applications. You cannot use an ACS server to serve content to users on Desktop client applications.

Configuration requirements

To use an ACS server:

- The ACS server must be configured to accept client requests from at least one network location.
- There must be at least one valid acs config object for the ACS server in each repository served by that ACS server.
- The acs.properties file for the ACS server must be correctly configured. (The installation process configures an acs.properties file automatically.)
- ACS server ports must be open if the network locations the server is servicing are outside a firewall.

BOCS servers

A Branch Office Caching Services (BOCS) server is a caching server. It is a separate, optional product with its own installer. It is not installed with Content Server.

BOCS servers cache content locally. When a BOCS server handles content requests from users, it caches the requested content locally. This allows users to obtain frequently accessed content very quickly. The amount of content that can be cached and the length of time which the content is held is configurable.

BOCS servers communicate only with ACS servers. They do not communicate directly with Content Servers. A BOCS server can communicate with any ACS server.

For more information about BOCS servers, refer to the *Branch Office Caching Services Installation and Release Notes*.

Benefits and best use

Use BOCS servers to give users on web-based clients the fastest possible access to frequently used content without the requirement to install and maintain a distributed storage area remote site.

Configuration requirements

To use BOCS servers:

- An acs config object representing the BOCS server must reside in every repository served by the BOCS server.
- An acs.properties file must be correctly configured for each BOCS server. (The installation process configures an acs.properties file automatically.)
- The ports on which BOCS server serves content to users must be open.

Distributed storage areas

A distributed storage area is a single storage area made up of multiple component storage areas. They are the foundation of the single-repository distributed model for desktop clients. A distributed storage area may also be used in web-based models. All sites in a model using a distributed storage area share the same repository, but each site has a distributed storage area component as its own local storage area to provide fast, local access to content.

The component storage areas may be file store or linked store storage areas. A file store component may be encrypted, compressed, using content duplication checking and prevention, or any combination of these.

Content files can be shared or replicated between component storage areas. (Refer to Shared content, page 29, and Content replication, page 30, for a description of those options.)

Benefits and best use

Using a distributed storage area solves the performance problems experienced by remote desktop users when all sites share one repository with centralized content storage. For example, if the repository and its content files are located in Nice, France, users in Toronto, Canada, will experience some delays in accessing files due to the distance between them and the server. If you set up a distributed storage area, each site has fast access to content files in a local storage area. Users in Toronto will no longer experience poor performance when opening documents.

For web-based users, this provides an alternate configuration model if the preferred model (described in Single Model 1: Single repository with content persistently stored at primary site and accessed using ACS or BOCS servers, page 14) is not acceptable.

Use constraint

After a repository begins using a distributed storage area, it is not possible to remove that storage area and return to a standalone, non-distributed configuration.

Configuration requirements

To use a distributed storage area:

- The host machines for the participating servers must be running the same operating system. They can all be running Windows or all UNIX, but you cannot mix the two.
- Your network must have high bandwidth and good reliability to support repository access.
- All the components of the distributed storage area and the containing distributed store object must have the same value in the media_type attribute. This attribute indicates whether the files stored in the storage area are thumbnail renditions, streaming content, or other.
- The index agent and index server must be installed only at the primary site.

There are other configuration requirements that depend on how you choose to handle content files. Refer to the discussion of the shared content and content replication options for these requirements.

Content-file servers

A content-file server (also called an RCS) resides at each remote site that has a component of a distributed storage area. Content-file servers are automatically configured to provide maximum performance for desktop users for content-related queries. Content-file servers do not handle metadata requests.

When a repository is configured with a distributed storage area, there is a primary site, and one or more remote sites. The RDBMS, which holds the repository's metadata, resides at the primary site. Each site, primary and remote, has a local content storage area and a local Content Server. The Content Server at the primary site is configured to service all metadata requests, as well as content requests from clients local to that server. The content-file servers at the remote sites are configured to handle only content requests. These servers provide content to users from their local storage area and write content to that storage area. An RCS does not handle metadata requests.

This arrangement provides reasonably good performance for content requests because content is stored locally and accessed by a local server. It also provides reasonably good performance for data requests because they are handled by the server closest to the RDBMS.

Benefits and best use

Content-file servers provide increased performance for repository queries when a repository has a distributed storage area distributed across different geographical sites.

Constraint

An RCS cannot be used as a fail-over server for the primary, or data, server. If the Content Server at the primary site fails, the system cannot failover to an RCS.

Configuration requirements

A configuration using a distributed storage area and its attendant content-file servers has the following configuration requirements:

- Proximity values for the Content Server at each site must identify one server as the data server and all others as content-file servers.

  The basic proximity values are configured automatically when the sites are installed. However, you may wish to modify the values.

- The index agent and index server must be installed at the primary site.
- The data server's session time-out value should be set to a minimum of 30 minutes.
- Servers at all sites must be able to use the same authentication mechanism to authenticate all remote users and groups accessing distributed documents.
- The content-file servers and the data server must have compatible secure connection mode configurations, so that clients connecting from remote sites can access both the content-file server and the data server, or the secure connection default for the client must allow the client to request a connection on a native or secure port

  Configuring the servers to accept either secure or native connections is the easiest solution. For more information, refer to Distributed environments and the secure connection defaults , page 56 . For instructions about setting the server's connection mode, refer to Setting the secure connection mode, page 109 of the *Content Server Administrator's Guide*.

Shared content

Shared content files are files that are stored in one component only of a distributed storage area but are accessible to users at all sites. Shared content files are fetched directly by the remote servers when needed.

Servers can fetch the content files directly when the components of the distributed storage area are configured as shared drives.

Benefits and best use

Using shared content eliminates the need to enable surrogate get functionality or run ContentReplication jobs at each component site.

Within a distributed storage area, the documents that are the best candidates for sharing are those that are local to one site and are accessed infrequently by users at other sites.

Configuration requirements

To share content files across sites:

- The distributed storage area component at each site must be a shared directory.
- The installation owner (dmadmin) at each site must be the same account at all sites.
- On Windows platforms, there must be a global domain for all sites and a global dmadmin account (or equivalent) in that domain.

Refer to Configuring shared content files, page 73, for details about the server configuration settings required to implement shared drives or surrogate get.

Content replication

Content replication supports configurations with distributed storage areas. When you set up a distributed storage area within a repository, some portion of the content files stored in each site are generally replicated to the other sites. Because each site has a copy of the content files, servers accessing these files don't need to fetch them from a remote storage area.

Content Server provides automatic replication, through the ContentReplication tool, on-demand replication, using the surrogate get feature, or manual replication, using the REPLICATE or IMPORT_REPLICA administration methods.

Note: You cannot replicate just content between different repositories. (If you want to replicate between repositories, refer to Object replication, page 32.)

Benefits and best use

Replicated content optimizes content transfer performance when users open a document because access is local. Consequently, the best candidates for replication are documents that are large or accessed frequently by users at all locations.

Configuration requirements

The ContentReplication tool and the surrogate get feature both require a homogenous server environment. The host machines for the participating servers must be all Windows or all UNIX machines.

If you use the REPLICATE administration method to replicate content or the IMPORT_REPLICA method to copy content, the servers must be able to connect with each other. On UNIX platforms, the servers must be able to connect using NFS.

The secure connection mode setting of the target server must be compatible with the connection mode requested by the client performing the content replication. For information about ensuring compatibility, refer to Distributed environments and the secure connection defaults , page 56 .

Reference links

Reference links are a feature of a multi-repository configuration. If an enterprise has multiple repositories, users are not limited to working only with objects in the repository to which they logged in. Users can also work with objects in other repositories. For example, a user might create a workflow in repository A and attach a document that resides in repository B to the workflow.

A reference link in a repository points to an object in another repository. The server creates reference links as needed when users work with objects in multiple repositories in the course of one repository session. The following operations create reference links:

- Linking a remote object to a local folder or cabinet
- Checking out a remote object
- Adding a remote object to a local virtual document

Benefits and best use

Reference links simplify work sessions for users. Users can start one repository session and manipulate objects in any repository without starting a new session each time they want to access an object in another repository. Reference links can also provide users with local access to the attributes of a remote object.

Configuration requirements

To use reference links:

- Users must have accounts in each repository they access.
- Each participating repository must project to the connection brokers of the other participating repositories.
- If you installed any servers in the participating repositories with trusted server licenses, you must configure all the servers to listen on both a secure and a native (unsecure) port. For instructions, refer to Setting the secure connection mode, page 109 of the *Content Server Administrator's Guide*.

To manage the user requirement easily, you can:

- Put all the repositories users will access in a federation
- Use an LDAP directory server to manage users and groups for the repositories
- Use both a federation and an LDAP directory server

A federation is a Documentum feature that facilitates management of global users and groups and external ACLs across repositories. Any addition or deletion of a global user

or group or modification of a global user or group attribute is made in the governing repository. The governing repository then propagates the changes to the member repositories. External ACLs are automatically replicated to member repositories as needed so that security for the global users and groups is uniform within the federation.

An LDAP directory server is a third-party product that provides a single place for maintenance of some or all users and groups in your enterprise. User and group entries are created in the directory server and those entries are propagated to all repositories set up to use the directory server. The attribute information that is propagated is defined when you set up the repository to use the LDAP directory server. The information is not limited to the global attributes of the users and groups.

Unlike a federation, the LDAP directory server does not replicate external ACLs to participating repositories. If you use a directory server without a federation, you must manage ACL replication manually.

If you use both a federation and an LDAP directory server, the directory server communicates with the governing repository in the federation (and any other unfederated repositories with which you want to use the directory server). The governing repository propagates the user and group changes it receives from the LDAP directory server to the member repositories and also manages the external ACLs within the federation.

Users who are managed by an LDAP directory server do not require an operating system account. However, Content Server requires an operating system login name to create the user. This login must be unique within the system. For LDAP-managed users, you can define operating system names for them without actually creating the operating system accounts.

For more information about federations, refer to Federations, page 34 . You can find information about configuring a repository to use an LDAP directory server in Using an LDAP directory server, page 354 of the *Content Server Administrator's Guide*.

Object replication

Object replication supports multi-repository distributed configurations. These configurations have multiple sites with a separate repository and relational database (RDBMS) at each site. Object replication replicates entire objects (attribute data and content) between repositories.

In the receiving repository, replicated objects are stored as replica objects. Users can manipulate replica objects much as they can the source objects with only a few constraints. Users can review, annotate, or index replicas. They can query against them or add them to a virtual document. They can also manipulate the source object through the replica. (For details of what operations are allowed, refer toTable 2–1, page 52 .)

Replication is automated using jobs, which are defined based on the business requirements of the enterprise.

Benefits and best use

Replication can reduce network traffic because users access local replicas of the objects for most operations. This is of most benefit during peak usage times.

Use object replication if you want local autonomy—if your wide area network (WAN) is not always reliable. If objects are replicated into a repository, users can continue to work even if the replica object's source repository is not available.

Configuration requirements

To use object replication:

- All participating sites must project to the connection brokers at all other participating sites.

  Note: Cross-projection between all sites is only required if you want:

  — To allow users to manipulate source objects through the replicas

  — To allow the server at the target repositories to perform automatic replica refreshes

  If your replicas are read-only, then cross-projection is not necessary. For example, if you are replicating from a repository inside a fire wall to a target repository outside a fire wall, the target server does not need to project to the source repository's connection broker nor the source to the target.
- Because object replication uses the dump and load operations, the databases of the source and target repositories must either use the same code page or the target database must be using Unicode. For example, if the source database is using Japanese and the target is using Unicode, the replication operation succeeds. However, if the source is using Unicode and the target is Japanese, replication fails.
- The secure_connect_mode defined for the server in the target repository must be compatible with the secure connection mode requested by the content replication job that performs the object replication.

  The mode requested by the job is defined in the dmcl.ini file used by the job. That file is found on the host machine where the job resides. The mode defaults to native. For instructions on changing the mode, refer to Requesting a native or secure connection, page 34 in the *Content Server Administrator's Guide*. For instructions on setting

secure_connect_mode for a server, refer to Setting the secure connection mode, page 109 of the *Content Server Administrator's Guide*.

- If you will be replicating documents created on Macintosh client machines, all participating sites must use the same Macintosh access protocol.
- Both the source and the target sites for each replication job must have enough disk space to accommodate the temporary dump files created by the job.
- Each target site must have enough disk space to accommodate the replicated content files.

Additionally, replication performance is affected by network latency (combination of network traffic and network speed). For adequate performance, a ping between any two participating sites should take 200 milliseconds or less.

Refer to Implementing object replication, page 89 for a full discussion of planning and setup considerations.

Federations

A federation is a set of two or more repositories that are bound together to facilitate the management of a multi-repository distributed configuration. One repository in the set is the governing repository. The remaining repositories are member repositories.

One enterprise can have multiple federations, but each repository can belong to only one federation.

A federation can include repositories with trusted servers and repositories with non-trusted servers.

Benefits and best use

In a multi-repository distributed configuration, users are typically working with objects from more than one repository in the same session. The objects may be the original objects, reference links, or replica objects. In all cases, to maintain data consistency, object type and format definitions should be the same across the repositories. Similarly, to maintain consistent security, the definitions of users, groups, and ACLs should be the same across the repositories.

Keeping objects synchronized in multiple repositories can be time consuming and error prone if the work is done manually in each repository. A repository federation automates much of the process. The governing repository automatically propagates changes you make to users, groups, and external ACLs to each member of the federation.

Federations are best used in multi-repository production environments where users share objects among the repositories. We do not recommend creating a repository federation that includes production, development, and test repositories. Typically, the object type and format definitions in development and test repositories change frequently, and their users are a small subset of the users in an enterprise. Development and test repositories may also be on different versions of the server than production repositories.

Configuration requirements

For the most consistent behavior in a repository federation (or any multi-repository distributed configuration):

- Object type and format definitions should be the same across all participating repositories.

  Documentum Administrator does not propagate type or format changes in a governing repository to member repositories. You must do this manually or using your own applications.
- Users and groups should be the same across all participating repositories.

  Documentum Administrator is used to manage all global users and groups in a repository federation. At set up, you must define whether users and groups are global or local. Making them all global is recommended.

  The federation update jobs automatically propagate all changes to global users defined by dm_user objects. If users are defined by subtypes of dm_user, they are not automatically propagated unless you identify those subtypes in the federation's attributes. You can do that when you create the federation or after, as a federation modification.

  Changes to global users and groups are only allowed using the Documentum Administrator and are propagated automatically to all member repositories.
- The server at the governing site must project to the connection brokers at the member sites.
- The servers at the member sites must project to the connection broker at the governing site.
- If you installed any of the participating Content Servers with trusted server licenses, you must ensure that either:
  - The servers are configured to listen on both a secure and a native port
  - The secure connection default for clients allows the clients to request a connection on a native or secure port

    Configuring the servers to accept either secure or native connections is the easiest solution. For more information, refer to Distributed environments and

the secure connection defaults , page 56 . For instructions about setting the server's connection mode, refer to Setting the secure connection mode, page 109 of the *Content Server Administrator's Guide*.

Building block architectures for single-repository models

The distributed models for web-based clients use either an ACS server or a BOCS server (or both) to distribute content to users at remote sites. These models use the network location, ACS server, BOCS server, and proximity values building blocks.

The distributed model for desktop-based clients uses the distributed storage area, shared or replicated content, and proximity values building blocks.

This section describes how the various building blocks work to support the distributed model for a single repository. It also includes a description of the internal communications in the web-based model.

Network locations

Network locations are recorded in the repository designated as the global registry. They are stored as dm_network_location_map objects. The attributes in the object record the name of the network location, which IP addresses or address ranges are assigned to that location, and which storage areas are considered near that location.

Network locations are used determine an end user's proximity to a Content Server or ACS server, and thereby, the user's proximity to a requested document. You must define network locations for remote sites with BOCS servers or ACS servers. The definitions are not required if remote sites are using the ACS at the primary site and there are no servers installed at the remote site.

ACS servers

ACS servers handle content requests from web-based clients. There is one ACS server for each Documentum installation on a server host machine. The ACS server runs in the Java method server.

When you configure the first repository in an installation, the procedure installs a Content Server for the repository and an ACS server for the installation. The procedure

also configures the Java method server Tomcat service if it does not exist and deploys the acs.war file.

If you later configure additional repositories in that installation, the procedure automatically updates the ACS configuration information for the existing ACS server. Adding additional repositories does not add additional ACS servers.

There are two internal components used to configure an ACS server. These are the acs.properties file and acs config objects. Each ACS server has one acs.properties file and at least one acs config object. The file is created when the ACS server is installed. The acs config object is created when a repository is configured.

The acs.properties file identifies the repositories with which the ACS server communicates. It is stored in %DM_HOME%\tomcat\webapps\ACS\WEB-INF\classes\config ($DM_HOME/tomcat/webapps/ACS/WEB-INF/classes/config).

The acs config object resides in a repository. The object contains attributes that identify which network locations can be served by the ACS server. The acs config object has an attribute that identifies the Content Server with which it communicates when handling requests for content from the repository. It may also contain the proximity values that define the ACS server's distance from each of the locations. The object also has an attribute, srv_config_id, that identifies the Content Server with which the ACS server communicates.

The acs config object may identify which storage areas the server can access. These must be either standalone file store storage areas or file stores that are components of a distributed storage area. An ACS server cannot access content in other types of storage areas. Nor can it directly access encrypted or compressed content or virtual documents or XML documents. If an ACS server receives a request for a document that it cannot access, the request is forwarded to its associated Content Server, which will service the request and send the result back to the ACS.

Note: If there are multiple servers installed on a single host machine for a single repository, the ACS server communicates only with one of the servers. You cannot manually configure acs config objects for additional servers on a host machine.

Acs config objects can be modified only through Documentum Administrator. The acs.prpoperties file is edited using a text editor.

BOCS server

The BOCS server configuration is recorded in an acs config object and an acs.properties file also. For BOCS servers, the config object identifies the server as a BOCS server and also identifies the network locations that it serves.

The acs.properties file for a BOCS server configures the location of the server's content cache and defines how long the server holds the content and some other behavioral

characteristics of the server. (For complete details, refer to the *Branch Office Caching Services Installation and Release Notes*.) In a BOCS installation on a client host, the file is placed in a location dependent on the host operating system:

- On a Windows platform, the location is files\Documentum\bocs\bocsTomcatInstance\webapps\bocs\WEB-INFO\classes\config
- On a UNIX platform, the location is $DOCUMENTUM_SHARED/bocs/bocsTomcatInstance/webapps/bocs/WEB-INFO/classes/config

Communication flow for web-based models

The following algorithm describes the basic communications that occur when an end user using a web browser requests a document. This description assumes that the clients are using UCF clients.

1. The request is received by a WDK-based application hosted on a web application server.
2. The application sends a Getfile request to the Content Server through the UCF facilities of the DFC on the web application server host.
3. Content Server sends back a list of the candidate content files.

   For each candidate file, the list describes where the file is, how far the file is from the user requesting the file, and instructions on how to build the URL to access the file. These instructions are digitally signed by Content Server using the private key stored in the dm_cryptographic_key object in the repository. The signature is used to ensure integrity and to enable authentication of the source of the instructions.
4. When the DFC receives the list of candidate content files from Content Server, it contacts a connection broker to determine which of the ACS servers referenced in the list are up and running. To obtain this information, it requests a repository map from the connection broker.
5. Using the information returned by the connection broker, the DFC constructs a URL to the content file closest to the user and accessible by a running ACS. The DFC sends the URL to the UCF client on the user's host machine.
6. When the UCF client receives the URL, it uses the URL to request the content from the chosen ACS or BOCS server.

   Depending on the configuration, the ACS server might be on a remote host machine or on the primary site's host machine.

   If an error occurs while uploading content to the UCF client and the UCF client returns an error to the DFC, the DFC will choose another candidate and construct

another URL. If all candidates fail to return content, the DFC obtains the content from Content Server.

7. If the server is an ACS server, it validates the signature on the request, using the public key provided to the server. Then, it returns the content file to the UCF client. (There is one public key per repository. It is stored in the dm_public_key object.)

   If the ACS is on a remote site and it determines that the storage areas it can access do not have the file, it sends a request to its associated Content Server, which then uses surrogate get to obtain the file.

8. If the server is an BOCS server, it searches its cache for the content. If the content is found, the BOCS server returns the content to the UCF client. If the content is not found, the BOCS server sends the request to the closest ACS server that has the content. The ACS server then returns the content to the BOCS server, which in turn, sends the content to the UCF client.

   Note: If there are no remote ACS servers close to the BOCS server, the BOCS server will request the content from the ACS server at the primary site. To ensure faster performance, frequent replication jobs to replicate content to the remote ACS server sites should be scheduled.

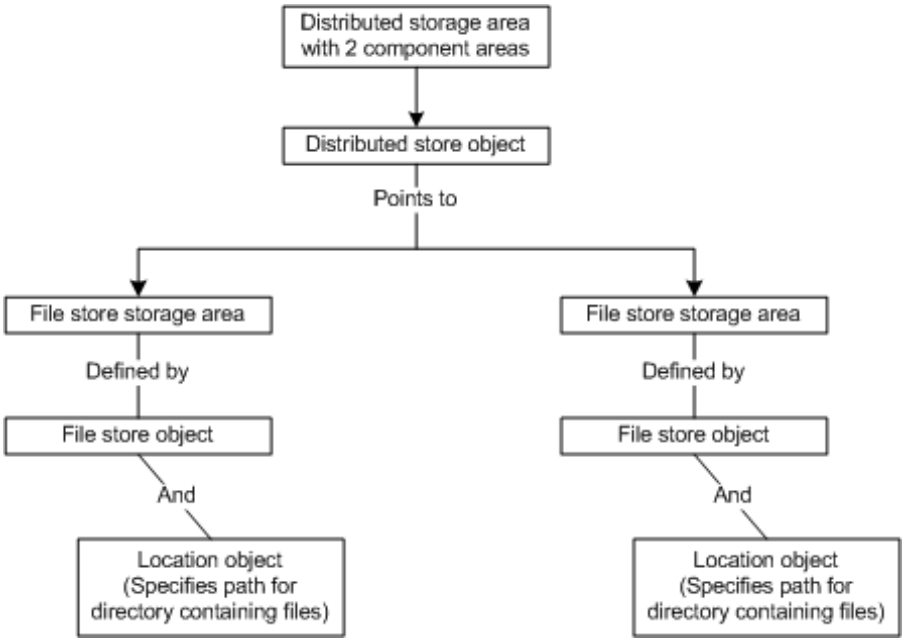
9. The UCF client writes the content to the user's local disk.

Distributed storage areas

A distributed storage area is defined by a distributed store object, which points to the component storage areas. Each component storage area is defined by a location and storage object.

For example, if you define a distributed storage area with two file store components, it is represented in the repository by one distributed store object, two file store objects, and the two location objects associated with the file store objects. Figure 2–1, page 40 illustrates this example.

Figure 2-1. Simple example of distributed architecture



This is a simple example. In reality, the component storage areas of a distributed storage area can be file stores or linked stores, and there can be as many component areas as you need. In one distributed storage area, the components can be all file stores or all linked stores, or they can be a mixture of the two types. However, all the components must have the same value in media_type attribute of their defining storage object. (The media_type attribute indicates the formats of the content files in the storage area.)

A file store component may be encrypted, compressed, using content duplication checking and prevention, or any combination of these.

Proximity values

Proximity values are used to define:

- A Content Server's proximity to a connection broker
- An ACS server's proximity to a connection broker
- A network location's proximity to an ACS server

Use by Content Servers

For Content Servers, proximity values represent the server's distance from the connection brokers. The initial values are set when a Content Server is installed. You can modify the values.

The values are set in the server config object or in the server.ini file. If a server is projecting to multiple connection brokers, the proximity values sent to each connection broker should reflect the server's relative distance from each connection broker. For example, assume a server is projecting to connection brokers, A and B, and the server is farther from connection broker B than connection broker A. In this case, the proximity value projected to connection broker B should be higher than that projected to connection broker A. The proximity values should reflect the topology of the network.

When a client requests connection information, the connection broker returns the proximity value of each known server in addition to other connection information to the client. The client uses these proximity values to distinguish between the data server and content-file servers.

To choose a server for data requests, the client looks for servers with proximity values of 0 to 999 and from that group chooses the server with the lowest proximity value. If there are no servers with a proximity value between 0 and 999, the client will look at the servers with proximity values of 9000 to 9999 and choose the server from that group with the lowest proximity value as the data server.

To choose a server for content requests, the client looks at all servers, compares the first three digits of each proximity value, and selects the lowest value. The digit in the fourth place (0 or 9) is disregarded when choosing a content-file server. Only the first three digits, counting left from the ones position, are considered. (Refer to Guidelines, page 79, for an illustration of this algorithm.) These first three digits are called a server's content proximity.

For example, assume that a server is projecting a value of 9032 to a connection broker. In this case, the server's proximity value is 9032 but its content proximity value is 032.

(For a discussion of where proximity values are stored and how the server projects proximity information to connection brokers, refer to Defining server proximity, page 117, in the *Content Server Administrator's Guide*.)

Use by ACS servers

When an ACS projects to a connection broker, the information tells the connection broker that the ACS server is running. This information is used to determine which ACS server to use when handling content request for a particular user. When a list of candidate content files is presented to DFC by Content Server, the information includes how close

each content is to the user (based on the user's network location). The DFC also asks the connection broker for a list of the ACS servers that are running. Using both sets of information, the DFC will choose the file to return to the user.

For instructions on configuring ACS projection to connection brokers, refer to Projecting an ACS server to connection brokers, page 68.

Use by network locations

For network locations, the proximity value identifies the location's distance from an ACS server. The proximity values may be recorded in the acs config objects representing the ACS servers that may service the location. Or, they may be recorded in attributes of the server config object of the ACS server's associated Content Server.

This information is used by DFC in conjunction with the information about an ACS server's proximity to a connection broker, to determine which ACS server handles a user's content request.

For information about setting the values, refer to Setting proximity values for network locations, page 69.

Shared content

Shared content is architected through shared directories. The distributed storage area component's directory at each site must be shared by all sites.

The feature also uses the settings of the far_stores attribute and the only_fetch_close attribute to ensure that content file reads and writes are handled correctly.

The far_stores attribute is a repeating attribute, defined for the server config object type, that contains the names of all storage areas that are considered far for the server. A server cannot save files into a far storage area.

For example, assume that a distributed storage area has component storage areas in Nice, Toronto, and Los Angeles. If the component storage areas in Toronto and Los Angeles are defined as far for the server in Nice, that server can only save files into the component storage area at its own site—Nice. Similarly, for the server in Toronto, if the component storage areas in Nice and Los Angeles are defined as far, the server can only save files into the component area at its site—Toronto.

The only_fetch_close attribute is defined for the distributed store object type. This attribute is FALSE by default, which allows servers to read from far storage areas. (Setting the attribute to TRUE means that servers can only read from storage areas not named in the far_stores attribute of their server config objects.)

Content replication

Content replication is used when a repository has a distributed storage area. There are several ways to replicate content to components of the distributed storage area:

- ContentReplication tool

  The ContentReplication tool provides automatic replication on a regular schedule. This tool is implemented as a job. After you define the parameters of the job, the agent exec process executes it automatically on the schedule you defined.

- Surrogate get

  The surrogate get feature provides replication on demand. If you use surrogate get, when users request a content file that is not present in their local storage area, the server automatically searches for the file in the component storage areas and replicates it into the user's local storage area.

  For surrogate get, you can use the system-defined SurrogateGet method or you can write your own program.

- REPLICATE or IMPORT_REPLICA

  The two administration methods, REPLICATE and IMPORT_REPLICA, let you replicate content manually. On UNIX, these two methods, these two methods require that the servers be able to connect using NFS.

Building block architectures for multi-repository models

This section describes the features that implement a multi-repository distributed object environment.

Reference links

A reference link is implemented as a pair of objects: a *mirror object* and a *dm_reference* object.

Mirror objects

A mirror object is an object in one repository that mirrors an object in another repository. The term *mirror object* describes the object's function. It is not a type name. For example,

if a user logs into repository A and checks out a document in repository B, the server creates a document in repository A that mirrors the document in repository B. The mirror object in repository A is an object of type dm_document.

Mirror objects only include the original object's attribute data. Any content associated with a remote object is not copied to the local repository when a mirror object is created.

Note: If the repository in which the mirror object is created is running on Sybase, values in some string attributes may be truncated in the mirror object. The length definition of some string attributes is shortened when a repository is implemented on Sybase.

Only a limited number of operations can affect mirror objects. For example, users can link mirror objects to local folders or cabinet or retrieve their attribute values. Most operations affect the remote objects represented by the mirror object.

For more information about directing operations to remote objects, refer to Appendix B, Writing Distributed Applications in *Content Server Fundamentals*.

Reference objects

Every mirror object has an associated dm_reference object. A dm_reference object is the internal link between the mirror object and the source object in the remote repository. Reference objects are persistent. They are stored in the same repository as the mirror object and are managed by Content Server. Users never see reference objects.

Object replication

Content Server's replication services allow you to replicate objects (attribute data and content) between repositories. Replication is automated, using jobs. Using parameters you define, the jobs dump a set of objects from one repository, called the source repository, and load them into another repository, called the target repository. Jobs can be initiated by:

- The source repository
- The target repository
- A third repository that starts the job but is neither the source nor the target repository.

  A repository that performs this function is called a *mediator* repository.

A replicated object in the target repository is called a *replica* and is designated by a replica icon.

Replication jobs

A replication job is defined as a job object in the repository that initiates the job. The attributes of the job object define the parameters of the replication job. They include how often the job is executed, the source and target repositories, what is replicated, and the security assigned to the replicas. Additional attributes provide status information about the job. They can tell you the time at which it last executed, the time of the next expected execution, and whether the job is active or suspended.

Jobs are created using Documentum Administrator by users with Superuser privileges.

Multi-dump file replication jobs

A replication job puts the objects to be replicated in a dump file. That file is then used to load the target repository. By default, all objects replicated by a single job are placed in a single dump file. If the job replicates a large number of objects, the size of the generated dump file can be very large because replication jobs replicate the specified objects plus some associated objects as needed. (Refer to What is replicated, page 49, for a description of what is actually replicated.)

A very large dump file can be problematic. For example, the size might be a problem if the job must transfer the file to a target repository over an unreliable network. Or the size might be a problem if either the source or target repository has limited disk space for the file.

To avoid problems arising from a large dump file, you can use a method argument, -objects_per_transfer, to direct the replication method to break down the job into a series of smaller dump and load operations. Each of the smaller operations dumps and loads a specified portion of the objects to be replicated. For example, suppose you want to replicate 100,000 objects in a single job. Rather than defining the job to use a single dump file, containing all 100,000 objects and associated objects, you could set the argument to 10,000, which causes the job to use a series of smaller dump and load operations, each replicating 10,000 objects (plus associated objects).

Replication jobs defined as a series of smaller dump and load operations continue until all objects to be replicated are replicated. The job dumps a specified number of objects (plus any associated objects), loads those into the target, then dumps the next set of objects and loads those, and so forth. There is no manual intervention necessary.

Best use of multiple-dump file replication

Defining a replication job to use multiple dump and load operations is beneficial if

- The disk space on either the source or target sites cannot accommodate a dump file of the size that would be generated if all objects to be replicated are put into a single file.

  Disk space requirements, page 97, describes how to estimate the needed disk space for a replication job.
- You want to avoid long, continuous remote dump or load operations

  Such operations require an operational network connection for the duration of the operation, which may require special configuration of firewalls, and so forth.
- An unreliable network makes it difficult to support an automatic transfer of large files.
- You want to limit the amount of lost work if the replication job encounters an error.

Conditions of use

Defining a replication job to use multiple dump and load operations is subject to the following conditions:

- The job must be defined as a fast replication job.
- The job cannot be defined as a manual transfer job.
- The initiating repository (source or mediator) may not be a version level prior to 5.3.

  Note: The target repository can be at any version level.

Note: If any of the above conditions are true (if the job is a fast replication job or defined as a manual transfer job or if the initiating repository is a version prior to 5.3), the -objects_per_transfer argument is automatically set to unlimited, so that all objects are replicated in one operation.

Replication modes

There are two replication modes: non-federated and federated. The modes determine how the owner_name, acl_name, acl_domain, and a_storage_type attributes of each replicated object are handled.

Non-federated replication mode

In the non-federated replication mode, the owner_name, acl_name, acl_domain, and a_storage_type attributes are considered local attributes. This mode is typically used when the source and target repositories are not members of the same federation.

Non-federated replication mode provides two choices for handling the owner_name, acl_name, and acl_domain attributes in the target repository. You can:

- Remap the attributes to default values provided in the replication job's definition

  If you choose this option, all replicas created by the job have the same ACL and are stored in the same storage area in the target repository.

  The ACL entries can be as open or restrictive as needed. This means that a user who has access to a document in one repository may not necessarily have access to the document's replicas in other repositories. Access in each repository is controlled by the ACL assigned to the replica in that repository.

  If you choose this option and fail to provide a default ACL, the server creates a default ACL that gives Relate permission to the world and group levels and Delete permission to the owner.
- Preserve current security and storage settings when possible

  If you choose this option, the owner_name attribute is reset to the default value, but the server does not automatically assign default values to the security and storage attributes. Instead:

  — The server looks for a matching ACL in the target repository if the source object has a public ACL.

  An ACL is a match if the acl_name and acl_domain values match those of the source object's ACL. If the server finds a matching ACL, that ACL is assigned to the replica. Otherwise, the replica's acl_name and acl_domain attributes are assigned the default values specified in the job definition.

  — The server looks for a matching storage area in the target repository.

  A storage area is a match if it has the same name as the storage area defined in the source object's a_storage_type attribute. If the server does not find a matching storage area in the target repository, the a_storage_type attribute in the replica is set to the storage area specified in the job definition.

  Note: When a repository is configured, a storage area named replica_filestore_01 is created. This area can be specified in the job definition as the storage area.

  If there is no storage area whose names matches the source storage area and a storage area is not defined in the job, the content is placed in the default storage area defined for the dm_sysobject type.

  — The group_name attribute of the replica object is always reset to the default group of the user specified in the replica's owner_name attribute.

Federated replication mode

In federated mode, the replica objects are controlled by global security, and the source repository retains control of the ownership and security of the replicas. The owner_name, group_name, acl_name, and acl_domain attributes are not mapped to local values in the target repository. Their values can only be changed from the source repository.

This means that the owner_name and group_name attributes in the replicas may contain values representing users or groups that do not exist in the target repository.

Federated replication handles ACLs in the following way:

- If the target repository contains an ACL that matches the ACL of a source object, the object's replica is assigned the local matching ACL.

  An ACL is a match if its acl_name and acl_domain attribute values match those of the source object's ACL. The entries in the ACL are not required to match those in the source object ACL.

  If the source ACL is a public ACL, the acl_domain value is reset to dbo when it is written into the replication dump file. When it is loaded into the target repository, dbo is considered a match for the owner of the target repository.
- If there is no matching ACL in the target repository, you have two options. You can choose to:

  — Preserve security

  The source ACLs are replicated into the target repository. The acl_name and acl_domain values are retained.

  — Remap security

  If the source ACL is an external ACL, the replica's acl_domain is reset to dbo and the acl_name is reset to the default value specified in the replication job definition.

In federated mode, the a_storage_type attribute is a local attribute. If the target repository contains a storage area of the same name, the replica content is placed in that storage area. If no storage area with the same name exists, the replica's a_storage_type attribute is reset to the storage area specified in the replication job definition and the replica's content is placed in that storage area.

Note: When a repository is configured, a storage area named replica_filestore_01 is created. This area can be specified in the job definition as the storage area.

If there is no storage area whose names matches the source storage area and a storage area is not defined in the job, the content is placed in the default storage area defined for the dm_sysobject type.

What is replicated

The first time an object replication job runs, the job replicates all objects of type dm_sysobject and its subtypes that are linked to the source folder or cabinet (and any subfolders) specified in the job definition. The entire version tree for each object is replicated.

Note: It is up to the system administrator or other person responsible for replication operations to determine which documents and folders should be linked to the source folder or cabinet and to ensure that the linking is done. The server does not select objects for replication.

The folder hierarchy within the source cabinet or folder is recreated in the target cabinet or folder. (The target cabinet or folder is defined in the job object.) The source cabinet or folder itself is not recreated in the target repository.

If a remote folder is linked to the source folder, the replication job replicates the reference link representing the remote folder. The job does not replicate the remote folder or any objects in the remote folder.

Additionally, the first run of a replication job also replicates certain objects that are related to the objects in the source folder or cabinet:

- All annotations attached to a replicated object
- All renditions of the replicated object
- If the replicated object is a parent in any user-defined relationships in the source repository, the child object and the associated relation and relation type object are replicated.
- If the replicated object is a virtual document, all of its components are replicated.

  Note: If a component is a remote object and is itself a virtual document or a folder, the replication does not traverse the remote object to replicate its children (in the case of a virtual document) or any objects it contains (in the case of a folder).
- If the replicated object has an associated assembly, all of the objects in the assembly are replicated.

  Note: It is possible to create an assembly for a virtual document and attach the assembly to a simple document. If such a simple document is replicated, the components of the assembly attached to it are also replicated.
- If the object is attached to a lifecycle, the policy object representing the lifecycle is replicated.

  Note: The replica lifecycle does not function as a lifecycle in the target repository. Documents cannot be attached to replica lifecycles.
- If a replicated object has attributes that reference other SysObjects by their object IDs, those referenced SysObjects are also replicated with one exception.

The exception is retainer objects. If a replicated object is assigned to a retainer object (a retention policy), the retainer object is not replicated to the target repository. The i_retainer_id attribute in the replica is not set.

- If the replication mode is federated, any events associated with the replicated objects are also replicated.

What is replicated on subsequent runs of the job depends on whether the job is defined as fast replication or not.

If the job isn't defined as fast replication, each time it runs, the job replicates all changed objects in the source folder. The job also examines all related objects and replicates any that have changed, even if their parent object—the SysObject to which they are related—is unchanged. The job also replicates any new objects in the source folder and their related objects.

In fast replication, with the exception of annotations, the job doesn't consider related objects for replication unless their parent object is changed or has not been previously replicated. If an object in the source folder hasn't changed since the last replication job, it isn't replicated and only its annotations are considered for inclusion. New or changed annotations are replicated but the object's other related objects are not included in the job.

If an object in the source folder has changed, it is replicated and changed related objects are not replicated unless they are also stored in or linked to the source folder.

A fast replication job also replicates any new objects in the source folder and their related objects.

For example, suppose you have a source folder named Source_TargetB with 10 documents in it, including a virtual document called MyVirtualDoc. MyVirtualDoc has two direct components, X and Y, and X has one child itself, X_1. The document X_1 is stored outside the source folder.

Now, suppose you run fast replication jobs with Source_TargetB as the source folder to repository B. The first time the job runs, the job replicates all the documents in Source_TargetB and their related objects. After the first run, one document, Mydoc, is added to the source folder and the document X_1 is versioned. On the second job run, the job replicates Mydoc and all its related objects. However, MyVirtualDoc hasn't changed. Consequently, it isn't replicated. Its component, X_1 isn't replicated either because it isn't stored in or linked to the source folder.

In fast replication, if you want to replicate related objects regardless of whether the parent objects are changed, you must link the related objects to the source folder.

Format objects and replication

It is recommended that you manually ensure that all format objects for a particular format across participating repositories are the same.

If the format of a replicated SysObject is not present in the target repository, Content Server will create the format object in the target repository. However, if you later change the format object in the source, the changes are not reflected in the target repository, nor will the changed format object be recreated in the target the next time the job runs. To ensure that format definitions are the same across participating repositories, create and maintain them manually.

Data dictionary information and replication

With one exception, data dictionary information is not replicated for object types or attributes. The exception is the dm_policy object type. If the replication job replicates a dm_policy object, the data dictionary information for the policy object type and its attributes is replicated also.

In the target repository, data dictionary information for any object types or attributes created by the replication process is available after the next execution of the data dictionary publishing job.

Display configuration information and replication

Display configuration information is information used by client applications to determine how to display one or more attributes when they appear in a dialog box. Display configuration information is stored in the repository in scope config and display config objects. When you install Content Server, a script sets up default display configurations for attributes in dm_sysobject, dm_document, and dm_folder.

Object replication does not replicate scope config and display objects associated with any object type. If you have defined display configuration information for types other than dm_sysobject, dm_document, or dm_folder, or have modified the information for these types, you must explicitly set it in the target repository.

If you have defined such information in the source repository and you want the same display configuration for the attributes in the target repository, you must set it up explicitly in the target repository.

Display configuration information is used by client applications, not by Content Server. For instructions on defining such information, refer to the *Documentum Application Builder User Guide*.

Full refreshes

It is possible to run a replication job as a full refresh job. Such a run will behave as a first-time execution of the job and update all replicas, regardless of whether the source has changed. However, full refreshes do not change the r_object_id attribute of the replicas in the target repository.

Related objects storage

Those objects that are related to objects in the source folder and are replicated because of their association with the source objects are stored in folders called Related Objects in the target repository. There is one Related Objects folder for each replication job that targets the repository. The Related Objects folder is located under a job's target folder in the target repository.

Replica objects

Replica objects are much like mirror objects. (Mirror objects are created when users share objects between repositories.) When an object is replicated into a repository, the replica is given a local object ID. The operation also creates a reference object for the replica that points to the replica's source object. Users perform operations on the source object or on the replica object. Table 2–1, page 52 lists the valid operations and which object they affect.

Table 2-1. Object affected by operations on replicas

Operation	Object affected
Update attribute values	Source object
Check out and check in	Source object
Execute an Addesignature or Verifyesignature	Source object
Link the object to a folder or cabinet	Replica object

Operation	Object affected
Replace the replica's ACL.	Controlled by the replication mode that created the replica: • If external mode was used, users can change the replica. • If federated mode was used, security changes must be made to the source object.
Move the object's storage location	Replica object
Change the object's indexing status or conduct full-text searches on the object	Replica object
View the object	Replica object
Perform lifecycle operations (Attach, Promote, Demote, Suspend, or Resume) on the object	Source object
Create renditions of the object	Replica object
Annotate the object	Replica object **Note:** If you want to annotate the source object, create the annotation in the source repository and issue the Addnote method with the source object as the target of the annotation. It is best practice to keep note objects in the same repository as the objects which they annotate.
Include the object in a virtual document.	Replica object
Copy the object	Replica object
Delete the object	Replica object

To perform an operation on a replica that affects the source object, the repository in which the source object resides must be active. That is, the source repository must have a server that is running.

The replication job definition determines whether the replicas are marked for indexing in the target repository.

Replica objects cannot be assigned to a retention policy.

To determine the source of a replica, examine its reference object or query the computed attribute _masterdocbase for the replica. This attribute returns an object's source repository.

Replica objects are identified in the user interface by a replica icon. Programmatically, you can identify a replica using the computed attribute _isreplica. This attribute is set to T for replicas. You can query this attribute using the Get method or the DQL SELECT statement. For DQL, specify the attribute name without the underscore (isreplica).

How the replication process works

Documentum's replication services use the agent exec process that is part of Content Server. This process runs continuously and, at intervals, scans the repository for jobs that are in need of execution. The process uses attributes in the job definition to determine whether to launch the job. These attributes define how often the job is run and when.

When the agent exec program finds a job that must be run, it executes the method that runs the program associated with the job. This method is also specified in the job definition. For replication, the method is the replicate_folder method, which is created as part of the replication installation and configuration process. The replicate_folder method runs the replicate_folder program, which is provided as part of Content Server.

The replicate_folder program performs the actual work of replication. In the source repository, it dumps the source cabinet or folder to a file, performs delete synchronization, transfers the dump file to the target repository, filters the file, and loads the filtered file into the target repository. (Delete synchronization makes sure that any objects that have been deleted from the source repository are also deleted from the target repository.)

Federations

Federations are created using Documentum Administrator. You define a repository as the governing repository in a federation, add member repositories, and activate the jobs that keep global users, groups, and external ACLs synchronized.

Supporting architecture

Internally, federations are supported by:

- A dm_federation object in each participating repository
- An attribute in the associated docbase config objects
- The globally_managed attribute in the user, group, and acl object types
- The replicate_temp_store storage area

The federation object in each repository is named for the federation in which the repository is participating. The attributes in a federation object list the federation's

members and the governing repository and provide some information about the federation's management, such as when each federation member was last updated. (Refer to Table 2–78, page 242 of the *EMC Documentum Object Reference Manual* for a list of the attributes defined for the federation type.)

In addition, each docbase config object has an attribute called r_federation_name that contains the name of the federation to which the repository belongs, if any. This attribute is provided as insurance because it is possible for a repository to have multiple federation objects (dump and load operations could create such a situation). In such instances, the server uses the federation object whose name matches that in the r_federation_name attribute in the docbase config object.

When a repository becomes a member of a federation, the governing repository's name and r_federation_name value are projected to all its connection broker targets immediately, to reflect the repository's new membership.

If the globally_managed attribute is set for a user, group, or ACL, that object is managed through the governing repository. Changes to globally managed objects in the governing repository are propagated to all member repositories. Documentum Administrator provides an easy tool for changing globally managed objects.

The replicate_temp_store storage area is used as a temporary storage place for the content of replication jobs. Object replication jobs create dump files that must be transferred from the source repository to the target repository. These files are held in the replicate_temp_store storage area until the replication job completes. Then the files are removed by Content Server.

The directory location of the replicate_temp_store storage area for a particular repository is %DOCUMENTUM%\data\replicate_temp_store*docbase_id* ($DOCUMENTUM/data/replicate_temp_store/*docbase_id*), where *docbase _id* is the hex value.

Jobs and methods

After a federation is created, global users, groups, and external ACLs are synchronized automatically using jobs and methods. Documentum Administrator manages any changes, additions, and deletions to users and groups using a change record file. The changes recorded in the file are propagated to all members by jobs. The Documentum Administrator manages external ACLs using replication jobs.

The jobs that perform these operations are activated by Documentum Administrator.

Distributed environments and the secure connection defaults

When users or applications connect to a server, they request either a secure or native (unsecure) connection. By default, all connection requests default to a request for a native connection and all servers listen on a native port. However, a server can be configured to listen on a secure port or on both a secure and a native port. If a server is listening on a secure port, then a client can request a secure connection with that server.

The building blocks that support single-repository distributed environments rely on connections between servers within a repository. The building blocks that support multi-repository distributed environments rely on connections between repositories. You must configure compatible secure connection defaults for the servers and clients to ensure successful connection requests.

There are two ways to configure the secure connection default settings to avoid connection failures:

- Set the secure_connect_mode attribute for the servers to dual.

  secure_connect_mode is a server config attribute. If you set the attribute to dual for a server, the server listens on both a native and a secure port. Consequently, a client requesting a connection through that server can request either a native or a secure port with success. Setting the secure connection mode, page 109 of the *Content Server Administrator's Guide* contains complete information about the secure_connect_default attribute and its settings.
- Set the secure_connect_default key in the client's dmcl.ini file to either try_secure_first or try_native_first.

  try_secure_first directs the DMCL to try to obtain a secure connection first and if that fails, to request a native connection. try_native_first directs the DMCL to try to obtain a native connection first and if that fails, to request a secure connection. In either case, the target server's default setting does not affect the success of the connection request.

  The jobs and features provided with Content Server that support and implement single- or multi-repository environments use a dmcl.ini file found on the host machine on which the job resides. The dmcl.ini file used by Documentum client products is found on the either client's host machine or in a network location determined by your system administrator. Requesting a native or secure connection,

page 168 of the *Content Server Administrator's Guide* provides complete information about setting the default for the client.

Chapter 3

Implementing Single-Repository Models

This chapter describes how to implement the building blocks most commonly used to create single-repository distributed configurations. The topics in this chapter are:

Implementing a distributed repository without a distributed storage area

Use the instructions in this section to implement the single-repository distributed model discussed in Single Model 1: Single repository with content persistently stored at primary site and accessed using ACS or BOCS servers, page 14. This is the preferred model for single-repository distributed configurations if users are accessing the repository using web-based clients.

To set up a distributed repository without a distributed storage area:

1. If not already installed, install the primary site by following the instructions in the *Content Server Installation Guide*.

   Note: One repository in the installation must be designated as the global registry if you are also installing BOCS servers at the remote sites.

2. To use a BOCS server at each remote site, install Branch Office Caching Services.

   Use the instructions in the *Branch Office Caching Services Installation and Release Notes*.

3. Define the network locations for the remote sites.

   If the remote sites are accessing content through the ACS server at the primary site and you are not installing BOCS servers at the remote sites, it is not necessary to define network locations.

   Use Documentum Administrator to define network locations. You must be connected to the repository designated as the global registry to define network locations.

Installing with Distributed Storage Areas

Use the information in this section if you are setting up a distributed configuration that uses a distributed storage area and you do not already have a Documentum installation. This section does not provide instructions for converting an existing installation. Following this procedure gives you one repository that uses a distributed storage area distributed across multiple geographical locations.

Caution: After a repository is set up with and using a distributed storage area, it is not possible to remove the distributed storage area and return to a standalone, non-distributed configuration.

There are three stages in the process of setting up a distributed storage area:

- Planning
- Setup at the primary site (the site where the RDBMS resides)
- Setup at the remote sites

Planning

Before you begin installing the Content Servers at any site:

- Decide whether to share distributed content or replicate distributed content or use a combination of both.

- Read and follow the guidelines in the next section to ensure that your environment is set up correctly.
- Use our guidelines for estimating the required disk space take sure you have enough disk space at the site.

Guidelines

To ensure that your distributed architecture works properly, follow these guidelines:

- The Content Server at each site (primary and remote) must be able to authenticate the user, using the same mechanism.

  When a remote user logs into a repository, the client sends two connection requests, one to the content-file server and one to the data server. Each server must be able to authenticate the user using the same authentication mechanism.
- If you intend to share content files among the component storage areas, the installation owner for all servers accessing the repository must be the same account at all sites.

  Having the same installation owner at each site allows the Content Server at each site to access content files at the other sites. On Windows platforms, to meet this requirement, you must have a global domain for all sites and you must establish a global dmadmin account (or equivalent) in that domain. At each site, you must log into the global dmadmin account when you install and configure the Content Server.
- If you intend to replicate content files among the component storage areas on Windows platforms, using a global domain dmadmin account for all sites is optional. You can install each site in a separate local domain. However, if you want to share content with other sites in the future or want to use enterprise-wide email notification, it is recommended that you install using a global dmadmin account.
- Method objects must be resolvable at all server sites if the sites are not connected using NFS.

  In a distributed installation, the method commands defined by the method_verb attribute of the method object must exist at each server site.

  Note: Some method objects may not have a full file system path defined (in the method_verb attribute) for the program they represent. For such programs to work correctly, the command executable must be found in the PATH definition for the user who is executing the command. If the run_as_server attribute for the method object is set to TRUE, the user executing the command is the installation owner. If run_as_server set to FALSE, the user executing the command is the user who has issued the EXECUTE statement or the DO_METHOD administration method. (Note that run_as_server is set to FALSE by default in the methods defined in the headstart.ebs file.)

Estimating Disk Space

Before you begin installing distributed sites, estimate how much disk space is required for content storage at each site.

The amount of space required at each distributed site depends on the following factors:

- Total number of bytes per document
- Total number of documents in the distributed repository
- Number of distributed sites
- Number of versions of each document that you intend to keep online
- Whether you intend to keep renditions of the documents
- Whether you intend to replicate each document to all sites

The formula for estimating disk space is:

```
(total # of documents)x(# of sites)x(bytes/document)
 x(# of versions) = total amount of disk space
```

Estimating Document Size

To estimate the total number of bytes per document, sum the following figures:

- Number of bytes for an average document
- Number of bytes for any renditions

Disk Space Calculations—An Example

To illustrate estimating disk space, assume that:

- Your enterprise has three distributed sites
- Site 1 will have 10,000 documents
- Sites 2 and 3 will have 5,000 each
- You intend to index PDFText renditions of the documents
- Each document is an average of 10K bytes.

First, estimate the total number of bytes per document. Include in the total the estimated size of the document's content and renditions. For example, assume that each document has 10K of content and PDF and PDFText renditions. For a 10K document, the PDF rendition is approximately 8K and the PDFText rendition is approximately 6K. Sum these estimates to arrive at the estimated total number of bytes per document. In our example, the sum is 24K. Use this total in the disk space formula to determine the disk space needed at each site.

In this example, the three sites have a total of 20,000 documents at 24K per document. Three versions of each will be kept online, with each replicated at each site:

20,000 docs x 3 sites x 24K/doc x 3 versions is approximately 4.68 gigabytes

This calculation indicates that you need a total of 4.68 gigabytes of disk space at each distributed site.

Setting up the Sites

The procedure in this section describes how to install a distributed storage area as part of a new Documentum installation. It does not provide instructions for adding a distributed storage area to an existing installation.

Installing a remote site installs a content-file server and an ACS server and creates a local file store storage area for the site. The name of the local file store has the following format:

`fs_rcs_server_config_name`

where *server_config_name* is the name of the server config object for the site's content-file server. If the full name, including the fs_rcs_ prefix is longer than 32 characters, the prefix is truncated so that the full name of the storage area is no longer than 32 characters.

The installation procedure also runs a script to install all the administration methods needed for that storage area. For information about the script, refer to The dm_rcs_setup.ebs script, page 66.

To implement distributed storage:

1. Make sure that you have read and are complying with the guidelines in the previous section.
2. Decide how much disk space you need at each distributed site to store content files.

   Refer to Estimating Disk Space, page 62 for instructions on estimating disk space requirements.
3. At the primary site (where the RDBMS is located), install Content Server and configure the repository using the instructions in *Installing Content Server*.
4. Install the index agent and index server at the primary site also.
5. Start Documentum Administrator and connect to the repository as a user with Superuser privileges.
6. Create the component storage area for the primary site.

   Use Documentum Administrator to create the storage area.

To ensure backwards and future compatibility, do not use the default file store, filestore_01, as the distributed component at the primary site. Create a new storage area to be the distributed component.

It is generally recommended that you place the content store on a different drive from the Documentum installation. This allows you to separate Documentum programs from Documentum data.

7. Create the distributed storage area.

   Use Documentum Administrator to create the distributed storage area.

   Be sure to add the storage are you created in the previous step as a component of the distributed storage area. (Do not use filestore_01 as the distributed component.)

   If you intend to implement content sharing for any or all of the content files in the distributed storage area, do not check Fetch content locally.

   If all the files will be replicated among the distributed storage area components (no sharing at all), if the components are encrypted filestores, or if you intend to use Surrogate Get, check Fetch content locally. Checking Fetch content locally sets only_fetch_close to TRUE, which restricts the server's read access to its local component storage area. It will not be able to fetch from the other component areas.

8. Set the default storage area for SysObjects to the distributed storage area.

   Use type management facilities in Documentum Administrator to set the default storage area for SysObjects.

9. Move the objects currently in filestore_01 to the distributed store.

   Note: There are objects in filestore_01 created by default during Content Server installation that are required to be in the distributed store's component storage areas in order for EMC Documentum Desktop to function properly. However, you cannot simply make filestore_01 a component of the distributed store to resolve this because doing so does not reset the attributes in the objects correctly and, additionally, it is strongly recommended that filestore_01 not be a component

   Use a DQL UPDATE...OBJECTS statement to move the objects:

   ```
   UPDATE dm_sysobject (ALL) OBJECTS
   SET a_storage_type = 'name_of_distributed_store'
   WHERE a_storage_type = 'filestore_01'
   ```

   name_of_distributed_store is the name of the distributed storage area you created in Step 7 .

   Note: If by some chance the filestore storage area has been in use already, it may have objects in it, such as installed lifecycles, that cannot be moved. The query above fails in such circumstances. This situation may occur if you are adding a distributed storage area to an existing repository that has been in use for any time. To resolve this, either modify the query to exclude such objects or, in the case of lifecycles (dm_policy objects), make sure that they are uninstalled first.

10. Set the time-out value for the server at the primary site to a minimum of 30 minutes.

The primary server's connection with a remote desktop client may time-out while content is being transferred between the client and the content-file server. If this occurs during a save operation on a new document, the primary server does not save the new object to the repository. To prevent this occurrence, set the client_session_timeout key in the [SERVER_STARTUP] section of the server.ini file to a minimum of 30 minutes. (Refer to Modifying the server.ini file, page 107 of the *Content Server Administrator's Guide* for information about modifying the server.ini file.)

11. At each remote site, use the instructions in *Content Server Installation Guide* to install a content-file server.

    Note: All servers must be trusted servers or all servers must be non-trusted servers. You cannot mix trusted servers and non-trusted servers against one repository.

12. Log into Documentum Administrator to perform the following manual steps:

    a. Update the distributed storage area to include the file store storage areas created at the remote sites.

    b. Update the Far stores list in each site's server config object to include the component storage areas for the other sites.

       For example, suppose there are three component sites, SiteX, SiteY, and SiteZ. If you are editing the server config object for siteX, its Far Stores list must include the component storage areas for SiteY and SiteZ.

    c. Optionally, reset the proximity values for the content-file servers.

       The installation process automatically sets a remote server's proximity to its local connection broker as 9001 and its proximity to the primary Content Server as 9010. You may want to reset the values to reflect your network's topography. Setting proximity values for Content Server projection to a connection broker, page 79, contains instructions and guidelines for choosing and setting appropriate proximity values.

13. Ensure that users and groups accessing distributed documents can be authenticated by servers at all sites, using the same mechanism.

    In a configuration using a distributed storage area, when a user connects to a repository, the client sends two connection requests: one to the data server and one to the content-file server. Each server must be able to authenticate the user's name and password using the same authentication mechanism.

14. If users at remote sites will be accessing the repository using web-based client applications (such as Webtop), perform the following additional steps for each remote site:

    a. Define at least one network location for the ACS server at the site.

       Creating Network Locations, page 66, contains instructions.

    b. Add the network location to the acs config object.

Add the location to the acs config object for each ACS server that can service requests from that network location. For instructions, refer to Adding network locations to an acs config object, page 68.

c. Define the network location's proximity to the ACS server or servers.

   Refer to Setting proximity values for network locations, page 69, for guidelines and instructions.

d. Define the ACS server's projection to a connection broker.

   Projecting an ACS server to connection brokers, page 68, provides instructions and guidelines for defining projection targets for ACS servers.

15. Ensure that all machines hosting a Content Server, content-file server, or ACS server are using UTC time and are synchronized.

    When a server at a remote site (content-file server or ACS) connects to the primary site to fetch content using surrogate get, it uses a global login ticket configured to be valid for 5 minutes to connect. If the clocks in the host machines are not synchronized, the connection may fail if the host machine finds the login ticket invalid if the time difference is greater than 5 minutes.

The dm_rcs_setup.ebs script

The script m_rcs_setup.ebs is run automatically when you install a remote site. It creates the administration jobs necessary to manage and administer the remote site's file store storage area, such as Content Replication, dmclean, and dmfilescan. It also enables Surrogate get for the storage area at the remote site and creates a sysadmin log directory on the remote host.

This script can be run manually if desired. Re-running the script in an installation in which it has already run is harmless. The command line is:

```
dmbasic -f %DM_HOME%\install\admin\dm_rcs_setup.ebs -eRCSSetup
- repository.server_config_name
```

where *repository* is the name of the repository and *server_config_name* is the name of the server config object for the Content Server at the remote site.

Creating Network Locations

Use the instructions and information in this section to create network locations after you install the servers supporting a web-based single-repository distributed configuration.

Use Documentum Administrator to create network locations. For instructions, refer to Documentum Administrator Online Help. Do not create or modify a network location

using DQL or the API. Changes made through DQL or the API are not recognized by the ACS server.

Network locations represent places or portions of a network's topography. Typically, each network location is defined as one or more IP addresses or range of addresses that are "in" that location. The addresses identified in network locations can overlap. That is, one or more network locations can include the same IP address or address range. This provides an opportunity for users to choose their network location when they start sessions with web-based clients. The clients can be configured to present the users with a list of all network locations configured for their IP address.

However, defining an IP address or address range in a network location is optional. If a client application fails to find a network location that includes a user's IP address, the application assigns the user to a default location or presents the users with a list of default locations and allows the user to choose. If there are no defaults, content requests are serviced by the ACS at the primary site. Default network locations are identified by a Boolean attribute in the objects that record network location definitions.

Each ACS and BOCS server typically serves one network location. For example, suppose a remote office in Florida has a Content Server and an ACS server. All the IP addresses for machines in that office and those of the Florida-based telecommuters would be specified in one network location map object, and that object is then identified in the acs config object associated with the ACS server in the Florida office.

However, an ACS or BOCS server can serve multiple network locations. For example, perhaps a separate network location object is defined for each building in the Florida office and one for each telecommuter. In this case, each network location must be included in the acs config object.

Network locations are recorded in the installation as dm_network_location_map objects that reside in the global registry. They must be stored in the global registry. The name of each location, stored in the dm_network_location_map.netloc_ident attribute, must be unique among the set of network locations in the global registry. Within the acs config object, they are identified in the acs_network_locations repeating attribute.

After you create the locations, update the acs config objects to include those locations that the associated ACS or BOCS server will service. In addition to specifying the locations in the acs config object, you must define the proximity of each location to the associated ACS or BOCS server. For instructions on updating the acs config objects, refer to Adding network locations to an acs config object, page 68. For instructions on setting proximity values, refer to Setting proximity values for network locations, page 69.

Adding network locations to an acs config object

After you create a network location, it cannot be used by the system until you add it to the appropriate acs config object or objects. To add a network location to an acs config object, you:

- Identify the network location as a location serviced by the ACS or BOCS server represented by the acs config object
- Specify the network location's proximity to the server

  A server may service multiple network locations, but it may be closer to some of those locations than it is to others. Set the network location's proximity value for each server appropriately. For guidelines and instructions, refer to Setting proximity values for network locations, page 69.

Use Documentum Administrator to update an acs config object to add a network location and its proximity values. For guidelines on setting the proximity, refer to Setting proximity values for network locations, page 69.

Projecting an ACS server to connection brokers

ACS servers must project their presence to at least one connection broker. Content Server uses that information to determine which ACS servers are running. It is not necessary to set these attributes if the acs config object represents a BOCS server.

Table 3–1, page 68, lists the attributes used to project presence to a connection broker. These attributes are repeating attributes and the values in one index position across the attributes represent the settings for the projection to one connection broker.

Table 3-1. Attributes in acs config objects related to connection broker projection

Attribute	Use
projection_enable	Determines whether the ACS server is projecting to the connection broker identified in the corresponding index position in projection_target. Set this to T (TRUE) to project to the connection broker. **Note:** If this is true for a particular index position, then projection_netloc_enable at the same index position must be false.

Attribute	Use
projection_targets	List of the connection brokers to which the ACS server projects its presence and proximity. Set this to the name of the connection broker.
projection_ports	List of the ports on which connection brokers are listening. The port specified at a particular index position corresponds to the connection broker identified at the corresponding index position in projection_targets.

Setting proximity values for network locations

Proximity values for network locations represent the location's proximity to an ACS server. The values are not set in the network location object, but instead are set in the acs config object for the ACS server. Alternatively, the proximity values defined in the server config object for the Content Server associated with the ACS server can be used. Those values are set to modifiable defaults when the Content Server is installed.

If you want to use the settings in the server config object, you must set the config_type attribute in the acs config object to 1. That setting directs the system to use the values in the server config object to obtain network location names and proximity values for the ACS server. If you want to edit those values, you must do so by editing the server config object.

Note: Setting that attribute to 1 also directs the system to consider the storage areas accessible to the Content Server as the storage areas accessible to the ACS server. For more information, refer to Defininging accessible storage areas for an ACS server, page 70.

Table 3–2, page 69, lists the attributes in the acs config or server config object that are used to record and handle proximity values for network locations. All the attributes in the table are repeating attributes. The values across one index position represent the proximity definition for one network location.

Table 3-2. Attributes in acs config and server config objects related to network proximity values

Attribute	Use
projection_netloc_enable	Determines whether the system regards the ACS server as available for the network location identified in the corresponding index position in projection_netloc_ident. Set this to T (TRUE) to make this ACS or BOCS server accessible to users in the network location identified in the corresponding index position in projection_netloc_ident.

Attribute	Use
	If this is F (FALSE), the system assumes that the ACS or BOCS server is currently unable to handle requests originating in the corresponding network location. **Note:** If this is true for a particular index position, then projection_enable at the same index position must be false.
projection_netloc_ident	List of the network locations that the ACS or BOCS server can service. Set this the name of the network location. This is the name set in the netloc_ident attribute of the dm_network_location_map object. **Note:** Network locations specified in this attribute must also be listed in the dm_acs_config.acs_network_locations attribute.
projection_proxval	Proximity of the network location at the corresponding index position in the projection_netloc_ident attribute to the ACS server. In an acs config object, set this to a number between 1 and 8999. The lower the number, the closer to the ACS or BOCS server the network location is presumed to be. In a server config object, the number may be over 9000. The system will adjust the value to use only the actual proximity, the value minus the 9000. (The number may also be being used to define a remote server's proximity.)

Use Documentum Administrator to edit the acs config or server config object and set the attributes listed in Table 3–2, page 69, to define network location proximity values.

The ACS server installed at the primary site has a built-in proximity of 9001. This means that, by default, all network locations have a proximity of 9001 to the ACS server at the primary site. If a user at a network location cannot access other, closer ACS servers, the ACS server at the primary site is used.

Definining accessible storage areas for an ACS server

Which storage areas an ACS server can access is dependent on the value in its config_type attribute:

- If config_type is set to 1, the ACS server uses the storage areas listed in the server config object's far_store attribute as a list of *unaccessible* storage areas. That is, if a storage area is identified in the far_stores attribute, the ACS server cannot fetch content from that storage area. Instead, the ACS server can fetch content from any storage area visible to the server and not listed in that attribute.

  Note: When the config_type is 1, the ACS Properties page in Documentum Administrator displays those storage areas listed in the Content Server's far_stores attribute. You cannot edit that field from the ACS Properties page. You must edit it from the server config's Properties page. (If the config_type is 1, the radio button labeled "Use the projection targets, network locations, and near stores already configured for the associated server config object" is selected on the ACS Properties page.)
- If the config_type attribute is set to 2, the ACS server uses the storage areas listed in the near_stores attribute of its acs config object as its *accessible* storage areas. (If the config_type is 2, the radio button labeled "Manually enter projection targets, netowrk locations, and near stores" is selected ACS Properties page.)

  The near_stores attribute is not set during the installation process.

By default, config_type is set to 1 when an ACS server is installed. To change the setting, and to set the near_stores attribute, use Documentum Administrator to edit these from the ACS server Properties page.

Modifying an acs.properties file

The acs.properties file contains configuration information for the ACS or BOCS server with which it is associated. The repository.acsconfig entry in the file identifies the server associated with the file. Use a text editor to modify the acs.properties file.

Note: For information and instructions about modifying the file to configure a BOCS server, refer the *BOCS Installation and Release Notes*.

File location

When the associated server is an ACS server installed on a content server installation, the file is found in %DM_HOME%\tomcat\webapps\ACS\WEB-INF\classes\config ($DM_HOME/tomcat/webapps/ACS/WEB-INF/classes/config). When the associated server is a BOCS server, the location differs by operating system platform:

- On a Windows platform, the location is files\Documentum\bocs\bocsTomcatInstance\webapps\bocs\WEB-INFO\classes\config

- On a UNIX platform, the location is $DOCUMENTUM_SHARED/bocs/bocsTomcatInstance/webapps/bocs/WEB-INFO/classes/config

The mode.cachestoreonly entry

This entry indicates whether the acs.properties file is associated with an ACS or BOCS server. The value in this entry must match the value specified in the is_cache_acs attribute of the acs config object for the server. The values of both these items are set when the ACS or BOCS server is installed.

Do not change these values. If the values of the entry and the acs config attribute do not match, the server does not work. Similarly, if the values are incorrect for the server, the server will not behave appropriately.

Adding entries for additional servers

If you are updating the file to allow an ACS server to communicate with an additional server for a repository in an installation, set the following configuration parameters for the server:

```
repository.name          # name of the repository
repository.login         # login name to be used to connect
                           the server
repository.acsconfig     # name of the acs config object
                           created for the server
```

Each parameter entry of a particular type is numbered after the first. For example, suppose the file currently has entries for only one server, the server for the Engineering repository. You want to add entries for an additional server for that repository. The additional entries would be:

```
repository.name.1=engr_1
repository.login.1=engr_1_login_user
repository.acsconfig.1=<acs config object name>
```

Note that each additional entry for each parameter is designated with a ".1". If you added a third server, the number is incremented by one. The entries for the third server would end in ".2" (The first entry for each parameter has no number.) You can add up to 99 entries for these parameters.

Setting repository.password is not needed or recommended. The ACS server uses trusted login to connect to the Content Server by default.

Disabling access to a server

You can disable access to an ACS server by particular network locations or by all network locations using that server.

To disable a particular network location's access to a particular ACS server:

- If the config_type attribute in the acs config object is set to 2, set the projection_netloc_enable attribute in the acs config object to F at the index position associated with that network location. For example, if you set projection_netloc_enable[2]=F, then access from network location identified in projection_netloc_ident[2] is disabled.
- If the config_type attribute in the acs config object is set to 1, set the projection_netloc_enable attribute in the associated server config object to F at the index position associated with that network location. For example, if you set projection_netloc_enable[2]=F, then access from network location identified in projection_netloc_id[2] is disabled.

To disable all access to the server, from any network location, set the acs config object's config_type attribute to 0.

Configuring shared content files

To enable content file sharing among a distributed storage area's components, you must:

- Configure each component storage area directory as a shared directory before you add it to the distributed storage area.
- Set the far_stores attribute for the server config object at each site.
- Set the only_fetch_close attribute of the distributed storage area to FALSE.

The far_stores attribute in a server config object defines the distributed storage area components that are not local for that server. For example, suppose a distributed site has three locations: London, New York, and Paris. At the New York site, the far_stores attribute for the server is set to London and Paris. In London, the attribute is set to New York and Paris. And finally, in Paris it is set to New York and London.

Note: If you followed the installation procedure for distributed storage areas, the far_stores attribute for the server at each of your distributed sites contains the names of all the other sites.

A server cannot save to a storage area defined as far. Whether the server can fetch from a far storage area is controlled by the only_fetch_close attribute.

When the attribute is FALSE (the default), the server can fetch files from far storage areas directly if the storage areas are shared directories.

When only_fetch_close is TRUE, servers cannot fetch content files from component storage areas named in the far_stores attribute in their server config objects. If a user opens a document whose content file is found only in a far storage area, the server will attempt to satisfy the request using surrogate get functionality. (Refer to Using the Surrogate Get Feature, page 77, for information about setting up surrogate get.)

Distributed storage areas and Documentum Transformation Services

Documentum Transformation Services is an optional product that provides extended processing and handling capabilities for rich media documents (audio, video and streaming formats). Installing Transformation Services installs a Media Server and a Thumbnail Server.

If Transformation Services is installed, when users save a document with a rich media format, Content Server queues a request to the Media Server to process the document. The Media Server saves metadata specific to the format, creates a thumbnail rendition of the document, and saves the thumbnail to a storage area specifically for thumbnails. To retrieve the thumbnails, applications typically use URLs. The URLs are sent to the Thumbnail Server, which returns the thumbnail rendition.

The following sections contain guidelines for setting up a distributed storage area for rich media documents.

Format objects

Set the default_storage attribute of the format objects representing the rich media primary content stored in the distributed store to the object ID of the dm_distributedstore object.

If you are putting the thumbnail renditions in a distributed store, set the default_storage attribute of the format objects representing the rendition formats to the object ID of the rendition's distributed storage area.

Using content-file servers

Install Documentum Transformation Services on the data server host only.

The thumbnail storage area must be a single (not distributed) file store storage area and must be located on the data server host (or a host that is visible to the data server).

The content-file servers at the component sites increase performance of queries and content retrieval because the requests go the server closest to the requested information. However, retrieving thumbnails is less efficient because they are not stored locally. If you want to store them locally and retrieve them with a content-file server, you must use a Getfile, instead of a URL, to retrieve the thumbnails. Typically, though, using Getfile is slower than using a URL.

Surrogate get

Surrogate get is a Content Server feature that is triggered when a user uses Getfile to request a file stored in an inaccessible (typically remote) component of a distributed storage area. In such cases, the server automatically fetches the file and replicates it to the local area. Surrogate get only works if the file is requested using a Getfile method. (Using the Surrogate Get Feature, page 77 describes the feature in detail.)

Thumbnail renditions and streaming content are generally retrieved using URLs. To obtain a URL, an application specifies the THUMBNAIL_URL keyword in a SELECT statement or issues a GET_FILE_URL administration method. When Content Server receives the request for the URL, it attempts to locate the requested file in the local component of the distributed storage area. If the server doesn't find the file in the local component, the server returns default parameters, from which the application can construct a URL that returns a default thumbnail. The server will not attempt to find the file in another storage component and replicate it in the local component.

Setting up content replication

Replicating content files among distributed storage area components ensures that users at each site have local copies of the files to access. Content Server includes several tools for content replication. You can set up replication to run automatically or you can perform the operation manually.

Deciding Which Tool to Use

Decide which tool to use based on how often replicated content changes and how important it is for users at any site to have access to current versions of the files.

Automatic Replication

There are two ways to replicate content automatically. You can use either or both in a single-repository distributed configuration. The tools are:

- ContentReplication tool
- Surrogate get feature

The ContentReplication tool greatly simplifies administration of content replication. The tool requires only enough temporary disk space to transfer the largest individual content file to be replicated. (In contrast, using dump and load requires enough temporary space to hold a dump file containing all content files to be replicated.) The ContentReplication tool is recommended for most situations. Refer to Content Replication, page 477 of the *Content Server Administrator's Guide* for instructions on using this tool.

The surrogate get feature replicates content on demand to the user's local storage area. When a user attempts to access the content, the server recognizes that the content is not available locally and calls the dm_SurrogateGet method. The method obtains the content from another server and executes an IMPORT_REPLICA call (an administration method) on the local server to make the content available to the user. The dm_SurrogateGet method is a system-defined method installed with Content Server. (For more information about the method, refer to The dm_SurrogateGet Method, page 78.) Using the Surrogate Get Feature, page 77 describes how to set up the surrogate get feature.

Manual Replication

To manually replicate content files, you can use:

- The REPLICATE administration method
- The IMPORT_REPLICA administration method

The REPLICATE administration method copies a file from one storage area to another. The disks on which both component storage areas reside must be accessible to the server.

The IMPORT_REPLICA administration method imports a file from an external file system into a storage area. You can use IMPORT_REPLICA with or without NFS access.

You can execute either REPLICATE or IMPORT_REPLICA from Documentum Administrator or using the EXECUTE statement or the Apply method. On UNIX platforms, both source and target storage areas must be available to the server through NFS.

Using the Surrogate Get Feature

When a user tries to access a content file that is stored in a remote component of a distributed storage area, the server must either fetch the file from the remote area directly or invoke the dm_SurrogateGet method to fetch the file and replicate it to the local storage area.

Use surrogate get if your site is not using shared directories. (If you are using shared directories, refer to Configuring shared content files, page 73 for instructions.)

To use surrogate get, you must:

- Set the far_stores attribute for the server config object at each site.
- Set the only_fetch_close attribute of the distributed storage area to TRUE.
- Make sure the get_method attributes in the storage areas are set to the name of the surrogate get method.
- Set the clocks on participating host machines to UTC time and synchronize the clocks.

  Surrogate get uses a login ticket to connect to servers to obtain content files. The login tickets are valid for five minutes. If the machine on which the dm_SurrogateGet method is running and the host machine of the server to which it connecting are not synchronized in time, the ticket may be invalid (timed out) on the target due to perceived differences in time. In such cases, the job will fail.

The far_stores attribute in a server config object defines the distributed storage area components that are not local for that server. For example, suppose a distributed site has three locations: London, New York, and Paris. At the New York site, the far_stores attribute for the server is set to London and Paris. In London, the attribute is set to New York and Paris. And finally, in Paris it is set to New York and London.

Note: If you followed the installation procedure for distributed storage areas, the far_stores attribute for the server at each of your distributed sites contains the names of all the other sites.

The only_fetch_close attribute controls whether a server can fetch content from a component storage area named in the far_stores attribute. When only_fetch_close is TRUE (the default), servers cannot fetch content files from component storage areas named in the far_stores attribute in their server config objects. If a user opens a document whose content file is found only in a far storage area, the server invokes the dm_SurrogateGet method, identified in the storage area's get_method attribute, to retrieve the file.

The dm_SurrogateGet Method

The dm_SurrogateGet method implements the surrogate get feature. The method is installed automatically along with the system administration tools. When you create a storage area, the get_method attribute of the storage area object is set to dm_SurrogateGet.

The method uses a global login ticket to connect to remote servers. The ticket has a validity period of five minutes. Host machines must be set to UTC time and synchronized. If that is not done, the host machine of the process receiving the login ticket may consider the ticket invalid if the time difference is greater than five minutes.

All surrogate get operations can be traced. When tracing is turned on, the trace file is called sugg.trc and is found in the %DOCUMENTUM%\bin ($DOCUMENTUM/bin) directory. Tracing surrogate get, page 110, contains instructions for turning on tracing of surrogate get operations.

You can also turn on tracing for the method invocation. Tracing the method invocation logs the method command line in the server log whenever the method is invoked. For instructions on turning on tracing for method invocation, refer to Tracing surrogate get, page 110.

Using REPLICATE

The REPLICATE administration method copies a file from one component storage area to another. The arguments determine which file or files are copied and where the copies are put. (Refer to REPLICATE, page 295 in the *Content Server DQL Reference Manual* for the syntax.)

You cannot select a component storage area from which to copy the file or files. You select only the target area; the server searches the remaining component areas in the distributed storage area and replicates the first copy of the file it finds.

If you are using EXECUTE to execute this administration method, the basic syntax is:

```
EXECUTE replicate
WITH query = 'dql_predicate',
store = 'target_storage_area_name'
```

If you are using the Apply method, the syntax is:

```
dmAPIGet("apply,session,NULL,REPLICATE,QUERY,S,dql_predicate,
STORE,S,target_storage_area_name")
```

When you use the Apply method, you must set the object_id argument to NULL. REPLICATE and the argument names and datatypes must be entered in uppercase.

Using IMPORT_REPLICA

The IMPORT_REPLICA administration method copies a file into a distributed storage component storage area. The file can be located in another component of the distributed storage area or in an external file system. If the source file (the file you are copying) is in another component of the distributed storage area, the server must have access to the disk on which the area resides. On UNIX platforms, if the disk is on another machine, the server must have access to NFS. If the file is in a directory on the same machine as the server or on a tape, diskette, or floppy, NFS is not required.

If you are using EXECUTE to execute IMPORT_REPLICA, the basic syntax is:

```
EXECUTE import_replica FOR content_object_id
WITH store = 'name_of_target_storage_area',
file = 'file_path_of_desired_file'
```

If you are using an Apply method, the syntax is:

```
dmAPIGet("apply,session,content_object_id,IMPORT_REPLICA,
STORE,S,name_of_target_storage_area,
FILE,S,file_path_of_file")
```

Setting proximity values for Content Server projection to a connection broker

Use the information in this section to help you set appropriate proximity values for Content Servers.

Guidelines

Use the following guidelines to define proximity values:

- Define proximity values for data servers in the range of 0 to 999. It is not necessary to include leading zeros.
- Define proximity values for content-file servers in the range of 9000 to 9999.
- Define proximity values for the primary server (the Content Server at the primary site) so that:
  - The server always projects the lowest proximity value among all servers projecting to any connection broker.
  - The server projects a content proximity value to the primary site connection broker that is lower than those projected to the site by the remote servers.

  - The server projects content proximity values to remote connection brokers that are higher than those projected from the remote sites.
- Define proximity values for a content-file server so that:
  - The proximity value it projects to a connection broker is higher than the proximity valued projected by the primary server.
  - The proximity value it projects to its local connection broker is lower than any projected to that connection broker from other sites.
- If two Content Servers have the same content proximity value and a client has already established a data connection to one of them, the client uses the established connection for content requests also, instead of opening another connection.
- A proximity value of -1 represents a server that is not available for any type of connection.

Example of selecting proximity values

Table 3–3, page 80, illustrates the proximity value guidelines. It shows projected proximity values for the servers in a repository called Demo. The repository has three sites: A, B, and C, with corresponding servers named DemoA, DemoB, and DemoC, and connection brokers named Connection BrokerA, Connection BrokerB, and Connection BrokerC.

Table 3-3. Example proximity values for a three-site configuration

	Projected proximity values		
Server	To connection brokerA	To connection brokerB	To connection brokerC
DemoA	0001	0002	0003
DemoB	9002	9001	9003
DemoC	9003	9002	9001

At site A

When a user requests a connection, Connection BrokerA returns the following server and proximity value pairs:

DemoA/0001
DemoB/9002
DemoC/9003

For data requests, the client uses DemoA because it is the only server with a proximity value in the 0 to 999 range. (If there were more than one server with a proximity value in that range, the client would choose the server with the lowest value.)

For content requests, the client also selects DemoA also because it has the lowest projected content proximity (001). Using the same server for data and content at the primary site is fine. DemoA is the closest server for users at the primary site.

Note: You can set up both a data server and a content-file server at the primary site. The content-file server at the primary site must project a proximity value to the local connection broker that is higher than the data server's proximity value and a content proximity value that is lower than any value projected by the servers at the remote sites.

At site B

When a user requests a connection, Connection BrokerB returns the following server and proximity value pairs:

DemoA/0002
DemoB/9001
DemoC/9002

For data requests, the client uses DemoA, as it is the only server with a proximity value in the 0 to 999 range. For connection requests, the client selects DemoB because its content proximity value (001) is lower than the content proximity value (002) projected by DemoA or DemoC.

At site C

When a user requests a connection, Connection BrokerC returns the following server and proximity value pairs:

DemoA/0003
DemoB/9003
DemoC/9001

For data requests, the client uses DemoA, the only server with a proximity value between 0 and 999. For connection requests, the client selects DemoC because its content proximity value (001) is lower than the content proximity value projected by either DemoA or DemoB (003).

First-Time Use

The first time a content-file server issues a Getfile request to retrieve a document, the response time may be slow. The server constructs some internal structures to handle Getfile requests. Subsequent calls use the structures built with the first call and are faster.

Chapter 4

Implementing Multi-Repository Models

This chapter describes how to set up the most commonly used building blocks for multi-repository distributed environments. Topics in this chapter include:

- Repository configuration for distributed environment, page 83 , below, which contains the configuration instructions for a distributed environment.
- Setting up a federation, page 87 , which describes how to create a repository federation.
- Implementing object replication, page 89 , which describes how to implement object replication among repositories.
- Best practices for object replication, page 105 , which contains rules for maintaining consistency between repositories.

For descriptions of federations and object replication, refer to Federations, page 34 and Object replication, page 32 .

Repository configuration for distributed environment

The information in this section is a supplement to the installation and configuration information in *Installing Content Server*. Use the information in this section to complete the configuration of your repositories if your repositories will be participating in:

- A repository federation
- Object replication
- Distributed workflow

The configuration procedures described in the following sections, when completed, allow the participating repositories to communicate correctly with each other:

- Connection broker setup, page 84 : Each server in the distributed configuration must be able to find the other servers through its connection broker.
- User setup, page 85 : A Documentum user must be established for each repository that can be used to access the other servers. This user must have superuser privilege.

- Password setup, page 85 : Each server must store the passwords needed for its distributed user to connect to the other repositories in the configuration.
- Distributed operations job activation, page 87 : The distributed operations job must be enabled at each repository.

Connection broker setup

Every server in the distributed environment must be able to find the other servers through its connection broker.

Like an end user, a Content Server also uses a connection broker to find outside repositories. (The connection brokers that a server uses are identified in the dmcl.ini file on the server host machine) If the participating servers all project to the same connection broker and send to that connection broker for connection information, then no special setup is required. However, if the participating servers use different connection brokers for connection information, then additional configuration is required. In such cases, you must ensure that each participating server projects its connection information to the connection brokers used by all other participating servers.

Target connection brokers for server projection are defined in the server config object. The target information is stored in five repeating attributes:

- projection_targets

  The projection_targets attribute contains the name of the host machine on which the connection broker resides.
- projection_port

  The projection_ports attribute contains the port number on which the connection broker is listening.
- projection_proxval

  The projection_proxval attribute contains the proximity value that the server projects to the connection broker.
- projection_enable

  The projection_enable attribute determines whether the server projects to the connection broker. If it is set to TRUE, the server projects to the connection broker. If it is set to FALSE, the server does not project the connection broker.
- projection_notes

  The projection_notes attribute is a place for you to record short notes about the target. The attribute is 80 characters long.

Use the repository management facilities in Documentum Administrator to modify the server config object to set the projection targets for a server. For instructions, refer to the Documentum Administrator online help.

More information about connection broker projection targets can be found in Communicating with connection brokers, page 112 in the *Content Server Administrator's Guide*.

User setup

In the current implementation, the participating repositories must all use the same user name when establishing a connection for the internal jobs that manage distributed operations.

To configure the repository user account to be used for distributed operations:

1. Create a user account that has Superuser privileges in each participating repository.

   The simplest arrangement is to use the same user for all repositories, as this requires defining only a single global superuser account.

   Note: If you are creating a global user through a federation's governing repository, the governing repository propagates the user to the other repositories. However, the user is created with no special privileges in the member repositories. You must then connect to each member repository and set the user's privileges to Superuser.
2. In each participating repository, set the operator_name attribute of the server's server config object to the user's login name.
3. Set up the password for the user in each repository.

   Refer to Password setup, page 85, for instructions.

Password setup

The server for each repository must have a password for the user you defined in the previous section (User setup, page 85). To define the password, set up a dm_operator.cnt for each repository and the appropriate number of *repository*.cnt files for each repository. The following procedures describe how to set up these files.

To set up the dm_operator.cnt file:

1. Create a file called dm_operator.cnt that contains the password for the user identified in the operator name.

   The password must be the same in all the dm_operator.cnt files.

2. Place a copy of the file in each repository in the following directory:
   - On Windows platforms: %DOCUMENTUM%\dba\config*repository_name*
   - On UNIX platforms: $DOCUMENTUM/dba/config/*repository_name*

To set up the *repository*.cnt file:

1. For each repository:
   a. Create a file named *repository_name*.cnt, where *repository_name* is the name of the repository.
   b. Put the user's password in that file.
      The password may be the same or different for each repository.
2. In each repository, place a copy of the *respository_name*.cnt file for each of the other participating repositories in the following location:
   - On Windows platforms: %DOCUMENTUM%\dba\config*repository_name*
   - On UNIX platforms: $DOCUMENTUM/dba/config/*repository_name*

   For example, suppose there are three participating repositories: RepositoryA, RepositoryB, and RepositoryC. You create three files:
   - repositoryA.cnt, which will contain the user's password for Repository A
   - repositoryB.cnt, which will contain the user's password for Repository B
   - repositoryC.cnt, which will contain the user's password for Repository C

   Then, you place:
   - Copies of repositoryB.cnt and repositoryC.cnt in Repository A
   - Copies of repositoryA.cnt and repositoryC.cnt in Repository B
   - Copies of repositoryA.cnt and repositoryB.cnt in Repository C

Object replication jobs

If you have object replication jobs that have defined a user other than the user identified in User setup, page 85 as the remote user, you may experience errors when you execute the jobs. The errors may occur because object replication jobs now use the dm_operator.cnt or *repository_name*.cnt password files to retrieve the password for the remote user.

For example, if you are running a replication job from target repositoryB that connects to repositoryA as a pull replication, the repositoryB server reads the local dm_operator.cnt or repositoryA.cnt file (depending on which is present) to obtain a password to use to connect to repositoryA.

To avoid errors, either give the remote superuser defined in your jobs the same password as the user identified in User setup, page 85 or change the remote user in the job to the user identified in User setup, page 85 .

Distributed operations job activation

To complete the configuration for distributed operations, the dm_DistOperations must be activated in each participating repository. This job is installed in the inactive state.

Use the Jobs management facilities in Documentum Administrator to activate the job in the participating repositories. The dm_DistOperations job is categorized as a replication job.

Setting up a federation

A federation is two or more repositories that are bound together to facilitate management of global users, groups, and ACLs in a multi-repository distributed configuration. One repository in the federation is defined as the governing repository. All changes to global users, groups, and external ACLs must be made through the governing repository.

Typically, an enterprise will have one federation. If an enterprise includes multiple, mutually exclusive groups that don't need to share documents, you can set up multiple federations. However, a repository can belong to only one federation.

A federation can include repositories with trusted servers and repositories with non-trusted servers.

We do not recommend mixing production, test, and development repositories in one federation.

Choosing the governing repository

Consider creating a new, empty repository to be the governing repository. Such a repository has the advantage of small size, making backups easier, and supporting jobs are run in a quiet repository rather than a big, busy repository.

If you are creating a federation from a group of existing repositories, choose the dominant repository as the governing repository. The dominant repository is the repository in which the majority of users are already defined as repository users.

Identifying user subtypes for propagation

By default, the federation jobs propagate all global users defined as dm_user objects in the governing repository. If you have created subtypes of the dm_user object type, global users defined with those subtypes are not propagated automatically.

To propagate users defined by dm_user subtypes, the subtypes must be present in all member repositories and you must identify the user subtypes to propagate when you create the federation. (You can add or delete from the subtypes after the federation is created also.)

Creating a federation

Federations are created and managed through Documentum Administrator. The following procedure summarizes the necessary steps. Documentum Administrator includes detailed online help for each step.

To create a federation:

1. Decide which repositories will participate in the federation and which repository will be the governing repository.
2. Make sure the governing repository is projecting to the connection brokers for all member repositories.
3. Make sure the member repositories are projecting to the governing repository's connection broker.

   Only repositories that project to the governing repository's connection broker appear in the list of possible member repositories.

   Projection targets are defined in the server config object, which can be modified using Documentum Administrator. For instructions about adding connection broker projection targets to servers, refer to Chapter 3, Servers , in the *Content Server Administrator's Guide*.
4. Create the federation using Documentum Administrator.

   For instructions, refer to the Documentum Administrator online help.

   Guidelines:

   - Checking the checkbox to make all current users and groups in the repository global activates the dm_FederationUpdate job. This job is responsible for starting the methods that keep the global objects synchronized across the federation. The job is inactive when installed.

If you do not check the checkbox to set all users and groups to global, you must manually set the dm_FederationUpdate job to active. (Use Documentum Administrator to do that.)

- The federation name must consist of ASCII characters and cannot contain any single quotes (’). You cannot change this name after the federation is created.
- The server uses the superuser name and password you provide for each member to connect to the member repository when running federation jobs and methods that update the global objects.
- If the member repository is running on a domain-required Windows server or if the superuser is in a domain that is different from the default, enter a domain for the superuser.

Implementing object replication

The following procedure lists the basic steps for planning and implementing object replication.

To implement object replication:

1. Define the business requirements for replication.

   The next section, Defining business requirements, page 90 , describes what you must consider when analyzing your business requirements in the context of object replication.

2. Determine whether your current infrastructure meets the needs of the business requirements.

   Infrastructure, page 93 discusses how to determine whether your infrastructure meets these.

3. Determine the needed computing resources.

   Disk space requirements, page 97 uses an example to illustrate how to determine the computing resources needed for replication.

4. Set up each site.

   Site set-up, page 100 describes the configuration changes you must make at each site to set up object replication.

5. Define the replication jobs.

   Defining jobs, page 103 contains step-by-step instructions for defining replication jobs.

Defining business requirements

Begin planning object replication implementation by determining your business requirements for replication. Two considerations can affect your business requirements:

- A replication job can only replicate documents to one target repository from one source repository.
- A replication job does not replicate users, groups, or object types.

The first consideration means that more than one replication job may be needed to satisfy a business requirement. For instance, to distribute documents to three geographically dispersed locations, you need three replication jobs.

The second consideration means that you must coordinate multi-site, multi-repository users, groups, and access permissions as a business function between repositories. Object types, users, and groups, are not replicated as part of the replication job. ACLs (dm_acl objects) may or may not be replicated, depending on how you configure the job.

Setting up a repository federation that includes all the repositories participating in object replication is the easiest way to coordinate users, groups, and security access across repositories. In a federation, users, groups, and external ACLs are global objects. All changes to them are made at the governing site and propagated automatically to other members of the federation. (Object types and formats must be manually managed regardless of whether the participating repositories are in the same federation.)

If you choose not to use a federation, you must manage users, groups, and security manually.

For example, suppose a planning group has users in two different repositories and the repositories are not members of the same federation. Tom, Dick, and Harry are users in repository 1, while Jane and Mary are users in repository 2. To allow these users to share documents of the object type planning_doc, the administrators for the two repositories must, at a minimum:

- Ensure that the planning_doc object type is defined identically in both repositories

  Note: If you replicate an object into a repository in which the object's type does not exist, the operation creates the type in the target repository. However, if the type already exists in both source and target repositories, it should be defined identically in each. Refer to Document types, page 91 for details.
- Create a planning group within each repository
- Add the users to both planning groups
- Create an ACL with an entry for the planning group that gives the group sufficient access to view replicated documents and their annotations
- Create a folder appropriate for the planning group's document sharing

The following sections describe the typical business requirements for users, groups, object types, security, workflow, and administration needs and discuss how these affect your replication implementation.

Functional divisions and groups

If your company has clearly defined functional divisions, translate these into appropriate Documentum group objects, to be defined in all repositories involved in replication. If you are developing an enterprise-wide replication plan prior to actually creating any repositories, create a standard script that defines these groups.

If you are not placing the repositories in a federation, you can run the standard script in each repository and be assured that the group definitions meet the business requirement and are standard across all repositories. (Edit the script for each repository so that the group creation statements are specific to the users in that repository.)

If the participating repositories will belong to a federation, it is only necessary to run the script at the governing repository site. The groups are automatically propagated to the other sites when the federation is created and the management jobs are active.

If you are converting existing repositories to accommodate a new or modified business requirement for replication, you may find that you must create new groups or modify existing groups. It still may be possible to utilize a standard script for this purpose, but some repository-by-repository modifications may also be necessary.

Document types

Document types usually evolve out of a combination of enterprise-wide and functional business requirements. The document types must have attributes that capture all of the information necessary for users to access and utilize the document in all business contexts. To preserve this information in replicated documents, it is important to define and maintain enterprise-wide document type definitions.

Document types can be defined using a standard script. This is particularly easy (and advised) if your repositories are not yet created. If you are converting existing repositories to meet replication business requirements, you may have to create new definitions or modify existing definitions.

Maintaining enterprise-wide document type definitions is desirable. It preserves all attribute and content information when a document is replicated. However, it is not mandatory for Documentum replication. If the definitions are not identical, the system will copy all information possible and ignore any information that cannot be replicated.

For example, suppose the user-defined document type planning_doc has 15 user-defined attributes including the attribute project_leader in repository 1, but it lacks that attribute in repository 2. Replication from repository 1 to repository 2 would result in a replica with 14 of the user-defined attributes, but not project_leader. A replication from repository 2 to repository 1 would result in a replica with all 15 attributes present, but with no information in the project_leader attribute.

If you replicate an object whose type is not defined in the target repository, the operation will create the type in the target repository as part of the replication process.

User distribution and geography

Documentum defines users at the repository level. If a user is defined in more than one repository, that user has a unique Documentum user ID in each repository. In a default installation, the server considers each a different user, even if the user_os_name and user_name attributes are identical in both repositories. In a federation, users are global objects managed by the governing repository, and the server considers users having the same user_os_name and user_name attributes in different repositories to be identical.

There are no user management concerns if you are replicating between repositories in the same federation. The federation's management jobs ensure that the user definitions are the same in each member repository.

Replication between repositories that are not in a federation or not in the same federation does not require user definitions to be the same across the repositories. In such cases, the replication job maps the ownership of the replicated objects to users in the target repository.

If your company has no policy for uniquely identifying users across sites, you can define users repository by repository. If there is a policy for uniquely identifying users across all sites, you can use this identification scheme in Documentum without modification. The product works in either situation.

When a new user joins a project or the company, it is up to the administration personnel to add the user to the appropriate groups, including any groups participating in replication processes if necessary. Generally, cross-repository coordination is not required. However, you may have set up some customizations that require coordination. For example, if you create registered tables of remote repository users and user_names in each repository (to support cross-repository event notification, for example), adding a new user to a repository requires coordination. In such cases, the new user must be added to the registered tables in the remote repositories also.

Security

The choices you make for security depend on the replication mode you are using. There are two replication modes: non-federated and federated. Each provides different security options. Replication modes, page 46 describes the security options available for each mode.

If you are using non-federated mode and choose to assign the same ACL to all replicas, security requirements may dictate what documents are included in a replication job. A replication business requirement with very complex security could be implemented by creating an ACL containing grants to many groups within the target repository, each with different access rights. Alternatively, the replication job could be divided into a group of replication jobs, each with its own simple but unique ACL.

Infrastructure

After you define your business requirements for replication, determine whether your infrastructure meets the needs of the requirements. Infrastructure is defined as the hardware, networking software, and people that support Documentum replication. You must determine whether you have adequate hardware resources, whether you want to perform replication online or offline, and how you want to assign the duties associated with managing a replication site. The following sections address these issues.

Reference metrics

Reference metrics provide a baseline for capacity planning, identifying potential infrastructure weaknesses, and determining what performance can be expected. To help determine whether the hardware and software infrastructure at your site is adequate to support your replication needs, compute reference metrics on each server that will participate in your enterprise-wide replication configuration. Compute the metrics during both off-peak and peak times, as replication jobs may need to run at both times.

Obtain the following baseline metrics:

- Server CPU capability

  Record the elapsed time for a local Documentum client to create and save 1,000 objects that have no content. This is an approximating metric for gauging CPU capability. Comparing CPU capability across environments is generally difficult.
- Server disk capacity

  Report df command information for the server.
- Server disk speed

Perform a local Documentum client Setfile/Getfile and record the elapsed time for each.

- Network speed

  Perform a Setfile/Getfile between each server pair participating in replication and record the elapsed time for each.

Note: Documentum provides a Docbasic script that you can use as a base for writing a script to obtain all of the reference metrics except disk capacity. You can find the script in %DM_HOME%\unsupported\replicate\metric.ebs ($DM_HOME/unsupported/replicate/metric.ebs).

For example, Table 4–1, page 94 shows the baseline metrics obtained for the two servers participating in replication testing. The metrics are expressed as minutes.

Table 4-1. Sample table for reference metrics

Metric	Fox, off-peak	Fox, peak	Bison, off-peak	Bison, peak
Server CPU	3:40	8:25	3:50	7:14
Local Setfile	0:30	1:30	:17	1:25
Local Getfile	0:17	1:57	0:14	1:30
Remote Setfile	31:02	37:29	30:56	36:17
Remote Getfile	29:57	35:56	29:30	36:49

Determining computing resources, page 95 illustrates how these metrics are used.

If you obtain inadequate metrics in any of these areas, factor that into your infrastructure planning:

- Alleviate CPU shortfalls by adding processors to your server or by putting RDBMS processing on a different server than Content Server.
- Alleviate disk shortfalls by procuring additional disk devices and rearranging the map of logical devices to controllers and physical disks.
- Alleviate network shortfalls by examining network router losses, procuring additional network bandwidth, or both.

  If you cannot resolve network bandwidth shortfalls, consider using the offline replication option (described in the following section, Network replication options, page 94).

Network replication options

There are two basic options for object replication: online and offline.

Online replication is the default. In online replication, the replication job originates at the target site, synchronously requests source-site processing, synchronously transfers the resulting dump file, then synchronously performs the target-site processing to complete the replication.

In offline replication, the replication job originates at the target site and requests the source-site processing. Asynchronously, the source site places the dump file in a requested location. A system administrator then uses ftp or tape to move the dump file from its source location to the target location. After the dump file is placed in its target location, the replication job picks up where it left off, performing the remaining target-site processing.

Replication system administration

In addition to the initial planning and coordination between sites, enterprise-wide replication has ongoing administrative tasks. For example, the duties associated with replication may include:

- Adding groups and users to the appropriate repositories
- Resolving problems when networks or repositories are not available
- Monitoring the progress of replication jobs and resolving failures
- Coordinating object type definitions across repositories

An enterprise with a number of large repositories at relatively autonomous sites will probably require one additional full-time individual per site to administer replication. This individual will need conventional Documentum skills (API, DQL, server environment, client environment, and scripting) as well as business knowledge in order to translate business replication requests into appropriate replication jobs.

Determining computing resources

Determining the computing resources required for replication is primarily a matter of listing business requirements, translating those requirements into specific replication jobs, and extrapolating the required machine resources based on the parameters of each job. The two areas that require careful examination are disk space requirements and job scheduling at each site. This section demonstrates this process by example.

Determining needed jobs

XYZ Enterprises has three geographically dispersed repositories: X, Y, and Z. They have two products on the market that were developed at their original site, X, which continues to control everything related to these products. However, Site Y needs rapid access to the documentation for products 1 and 2, and site Z needs rapid access to product 2 documentation.

Additionally, XYZ Enterprises is developing a new product that requires collaboration among all three sites. Every document produced in connection with the new product will be replicated from its originating site to the other two. All three sites are expected to generate large amounts of review annotations, which must also be replicated to all sites.

The company has defined the following six replication jobs to achieve these business objectives:

Product 1 Replication Jobs:
Job 1. From X to Y once a week

Product 2 Replication Jobs:
Job 2. From X to Y once a week
Job 3. From Y to Z (X documents received indirectly from Y) once a week

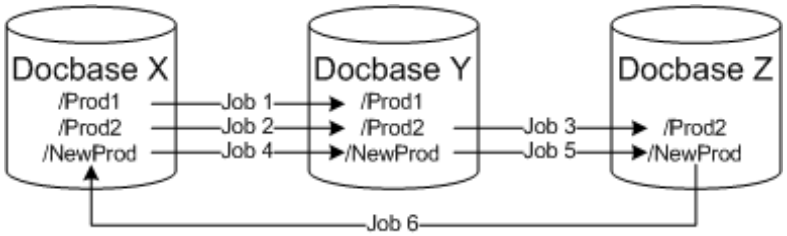
New Product Replication Jobs:
Job 4. From X to Y every 2 hours
Job 5. From Y to Z every 2 hours
Job 6. From Z to X every 2 hours

Jobs 1 and 2 represent the standard distribution of X documents to Site Y. Because it is possible to replicate replica documents, Job 3 completes the distribution by replicating X documents to Z through Y.

For Jobs 4, 5, and 6, each site's target and source folder for the new product documentation is called /NewProd. This, plus the circular nature of Jobs 4-6, means that each site will have its own documents and annotations as well as all documents and annotations from the /NewProd folders of the other sites in its own /NewProd folder.

Figure 4–1, page 96 illustrates these jobs.

Figure 4-1. XYZ jobs



Disk space requirements

This section illustrates how to estimate the disk space required for replication as accurately as possible. The values used in this section for each required estimate are only examples. For your calculations, you must determine the correct figures for the required estimates.

For replicated documents

Use a table to calculate disk space requirements. First, compute the requirements for the source repository documents and metadata. This is a function of the:

- Number of source documents and virtual descendant (related) documents
- Estimated average document size
- Estimated rendition size
- Estimated annotation size

Most documents have multiple versions, so be sure to take versions into account also.

Table 4–2, page 97 shows these figures for the XYZ Enterprises example.

Table 4-2. Disk requirements by source

	Product 1, X	Product 2, X	New Product, X	New Product, Y	New Product, Z
Number of documents	1,000	2,000	100	50	75
Number of versions	6	5	4	3	2
Content (KB)	10	20	5	15	18
Renditions (KB)	20	40	5	15	18
Annotations	0	0	33	5	5
Total content (KB)	30	60	43	35	41
Estimated Total (MB)	200	650	18	4.35	11.41

Total content is the sum of content, renditions, and annotations for each document.

The total represents the total size of the repository document and metadata content. It is the product of the total number of documents (number of documents times the number

of versions per document) and the total number of bytes per document (total content + 2,500 bytes per document for metadata overhead), or

Total = (no. of documents x no. of versions per document) x (total content + 2,500 bytes)

The metadata overhead varies, depending on the complexity of the document types and should be verified for your mix of documents.

After you calculate source disk space, you can project the total requirement to each of the sites. The total requirement is the sum of the source and replicated documents at each site plus required temporary space for dump files.

Temporary space for dump files

Each replication site needs temporary space for dump files. (Disk space for temporary files, page 101 describes the storage locations of the source and target dump files.) Replication processing generates a dump file at the source site, then stores it as the content of a document in the source repository before transferring it to the target repository. After the dump file arrives at the target site, it is filtered into another dump file specific to the target repository (owner and ACL information for each document are modified, if necessary, to conform to replication job requirements, and some objects are removed). This means that the temporary disk space available for dump files on both the source and target sites must be twice the size of the largest dump files.

If disk space is limited on either the source or target site, consider using the option to run the job as a series of smaller dump and load operations. This option is described in Multi-dump file replication jobs, page 45.

If you are not planning to run the job using multiple dump and load operations, calculate temporary space using the assumption that at some point a full-refresh replication will occur. (A full refresh replicates all documents in a single replication request rather than incrementally as they are modified.) Increase the temporary disk space if scheduling requires you to run multiple replication jobs simultaneously. Table 4–3, page 98 shows the calculations for XYZ Enterprises. The Temp column represents twice the disk space requirement of the largest full-refresh replication job at that site. In this example, Product 2 has the largest content size, so that figure is doubled for the Temp calculation.

Table 4-3. Disk requirements for each site, in MB

Site	Product 1	Product 2	New Product	Storage	Temp	Total
X	200	650	22.76	872.76	1300	2,172.76
Y	200	650	22.76	872.76	1300	2,172.76
Z	-	650	22.76	672.76	1300	1,972.76

Job scheduling

A replication job consists of three components:

- Source site processing to generate the dump file
- Dump file transfer
- Target site processing to load the dump file

The processing time required to complete all three steps is the processing window for the job. To verify that your replication schedules are achievable, estimate the processing window for each job.

In the XYZ Enterprises example, Jobs 1, 2, and 3 can be scheduled for off-peak server usage times (such as the weekend) because they are run weekly. It is unlikely that these jobs will conflict with other uses of the servers. On the other hand, Jobs 4, 5, and 6 are scheduled to run every 2 hours. These jobs will be running during peak load times for the source and target servers. If the processing window exceeds 2 hours, the schedule cannot be maintained.

To assess this possibility, develop an estimate of elapsed time for the three steps, based on reference metrics and the size of the replication job. Use the CPU metric and the local Setfile metric to estimate the time required for dump file generation at a source. Use the remote Getfile metric to estimate the time required for dump file transfer. Use the CPU metric to estimate the time required by the filter program and dump file load. The sum of these estimates will provide an estimate for total processing time.

The table below shows the calculations for Job 4, using the peak reference metrics shown in Table 4–1, page 94 and assuming a full refresh of the replicated objects. The metrics measuring documents per second in Table 4–4, page 99 are obtained by dividing the size of the server executable by the operation's elapsed time as shown in Table 4–1, page 94 , because the server executable was the file used to generate the metrics in Table 4–1, page 94 . Peak values are used because Job 4 will execute throughout the day.

Table 4-4. Processing time calculations for job 4

Processing Step	Reference Metric	No. of Objects or File Size	Time
Generate dump file	0.5 docs/second	700	350 seconds
Store dump file	111K/second	22.76 MB	205 seconds
Transfer dump file	4.4K/second	22.76 MB	5172 seconds
Filter dump file	0.43 docs/second	700	301 seconds
Load dump file	0.43 docs/second	550	237 seconds
Total			6265 seconds (1 hr, 44 min, 25 sec)

Handling overlapping jobs

Overlapping jobs are two replication jobs that replicate the same object either into one repository or into and out of the same repository. For example, suppose you have two replication jobs with the same source and target repositories. If the execution times overlap and the jobs happen to replicate some of the same objects, errors may occur. Or, suppose one job A replicates into a target repository YY and another job B that replicates from repository YY. In such cases, if the execution times overlap, it is possible that job B may attempt to replicate objects that job A is loading. This is an error.

The recommended way to avoid such clashes is to group jobs that may overlap into a job sequence. A job sequence is a set of jobs that are executed in a user-defined order. If a job in the sequence has a predecessor in the sequence, that job is not executed until its predecessor completes successfully. For more information about job sequences, refer to Creating jobs and job sequences, page 143.

Site set-up

Defining business requirements, page 90, describes the cross-repository requirements for object definitions and security. Establishing and following enterprise-wide standards for groups, document types, and security is critical to the success of replication.

Some other aspects of an installation must also be reviewed and any required reconfiguration performed before you start replication jobs. The following sections describe these considerations.

Connection broker setup and validation

If you want to use all the features associated with object replication, including automatic refresh of replicas and the ability to manipulate source objects through the replicas, then each participating site must project to the connection broker at each of the other participating sites. However, if you want only to create read-only replicas in the target repository, then cross-projection is not required.

This section describes how to set up cross projection. It may be helpful to have a listing of the repositories already known to your local connection broker before setting up cross projection.

To add a new repository to a cross-projection environment:

1. Modify the server config object for the new repository to add connection broker projection targets for each remote site that is a potential replication target.

After you modify the server config object, you must reinitialize the server for the changes to take effect. Use Documentum Administrator to modify the server config object. Refer to the Documentum Administrator online Help for instructions.

2. Modify the server config objects in existing repositories that are potential replication sources to add a connection broker projection target for the new repository.

   After you modify the server config object, you must reinitialize the server for the changes to take effect. Use Documentum Administrator to modify the server config object. Refer to the Documentum Administrator online Help for instructions.

Macintosh access protocol

If you intend to replicate Macintosh documents, all participating repositories must be using the same Macintosh access protocol. The protocol controls header, trailing and type creator information that is added to and removed from the resource fork when the client DMCL is sending the resource fork to the server or receiving it from the server.

The access protocol is set in the mac_access_protocol attribute of the repository's docbase config object. Although the replication setup scripts check for consistency, confirm that your sites are consistent prior to installation. To determine which protocol a repository is using, execute the following DQL query:

```
SELECT "mac_access_protocol" FROM "dm_docbase_config"
```

Disk space for temporary files

Replication creates temporary files in both the source and target repositories. The largest temporary files are the dump files. The temporary files are stored in the %DOCUMENTUM%\share\temp\replicate ($DOCUMENTUM/share/temp/replicate) directory in each installation. The replication operation creates a directory underneath this directory for each repository participating in replication on that machine. Make sure that this directory is located on a file system with enough space for the temporary files.

On UNIX, if the directory is not located on a file system with enough space for these temporary replication-related files, you can allocate a separate area and create a UNIX link, called replicate, to the share/temp area. For example:

```
% cd /u107/dm/* file system with adequate space */
% mkdir dmtemp
% cd $DOCUMENTUM/share/temp
% ln -s /u107/dm/dmtemp replicate
% cd replicate
<create directories for all repositories on site that will participate
in replication and set all permissions for the installation owner>
```

Content storage

Disk capacity for replication is a critical issue.

On the source side, the documents associated with the replication dump files are stored in a storage area called replicate_temp_store. This area is created by default in the same directory as the default SysObject file store area (filestore_01).

Where content is stored in the target repository is dependent on the replication mode chosen for the job. Replication modes, page 46, describes how each mode determines where to place content files associated with replicas.

Cabinets and folders

Object replication jobs replicate objects linked to a particular folder or cabinet called the source folder. The source folder is identified in the job definition. The replicated objects are stored in the target repository in a particular folder or cabinet designated as the target folder in the job definition. Both the source folder and the target folder must exist before you define the replication job. When you create these folders (or cabinets), keep in mind the following code page requirements:

- If the participating repositories have the same server_os_codepage value, the names of the source and target folders must be compatible with that code page.
- If the participating repositories have different server_os_codepage values, the names of the source and target folders must consist of only ASCII characters.

The replication process does not replicate the source folder or cabinet in the target repository's destination folder or cabinet. However, the process does replicate the folder hierarchy found within the source folder or cabinet.

If the source folder or cabinet contains a reference link, the replication process replicates the reference link but does not replicate any objects contained within the object pointed to by the reference link.

If two repositories are exchanging documents using folders with the same names and subfolder structures, name the subfolders identically in both repositories. For example, suppose that repository1 and repository2 both have /Product folders that are used for replication. In repository1, /Product has a subfolder named market_outlook, while in repository2, /Product has a sub folder named marketoutlook. The names of the subfolders only differ by one character, the underscore. However, because the names do not match exactly, after bidirectional replication both repositories will have the following folder paths:

repository1
/Product/market_outlook, which contains source documents
/Product/marketoutlook, which contains replicated documents

repository2
/Product/market_outlook, which contains replicated documents
/Product/marketoutlook, which contains source documents

This was not the desired result. If subfolders are named consistently, all documents, whether replicas or not, will appear in one folder in both the source and target repositories.

Additionally, make sure that all users participating in the replication have permission to link documents into the source folder.

Defining jobs

New jobs are defined using Documentum Administrator. You must be a superuser to define a replication job.

Before you create a job, the target folder or cabinet for that job must exist. The server only allows you to specify an existing folder or cabinet as the target when you are defining a job. If the folder or cabinet does not exist when you define the job, you must save the unfinished job definition, go back and create the folder or cabinet, and then finish the job definition. Refer to the previous section, Cabinets and folders, page 102, for guidelines for creating the source and target folders.

Guidelines for all jobs

Observe the following guidelines when creating a job:

- To define a push replication job, create the job in the source repository. To define a pull replication job, create the job in the target repository. If you want the job run by a mediator repository (a repository that is neither the source nor the target of the job), create the job in the mediator repository.
- If the participating repositories are 4.2 or higher and the underlying databases of the repositories are using different code pages, set the Codepage field for the replication job to UTF8. If the databases are using the same code page, set the field to that code page.
- Choosing fast replication can make the job run faster. However, it accomplishes the faster speed by limiting which related objects are replicated. For detailed information about what is replicated by default and fast replication, refer to What is replicated, page 49.
- If you define multiple jobs with the same target and source repositories, all jobs on the source repository must run under the same user account.

- You can set a flag in the job definition that directs the server to restart the replication job if the job fails. However, we recommend that you make sure that your job is correctly defined and running smoothly before you set that flag. The restart feature is intended to recover jobs that fail because some factor in their environment, such as the source or target repository, is unavailable.

Guidelines for multi-dump file jobs

Observe the following guidelines when defining a job that uses multiple dump and load operations to perform the replication:

- Set the -objects_per_transfer argument in the method arguments to number of objects you want to transfer in each operation.

  It is recommended that you do not specify fewer than 1000 objects in the argument.
- You must define the job as a fast replication job.
- The repository in which the job is defined must be at least at the 5.3 version level.
- You cannot define the job as a manual transfer job.

Setting up tracing

Object replication jobs have two tracing-related arguments: -trace_level and -trace_detail. The -trace_level argument defines the tracing level for the job. You can set the trace level to any of the tracing levels accepted by the Trace API method. The -trace-detail argument is a Boolean argument. Its setting controls what is traced when the -trace_level is set to 10 or higher. If -trace_detail is set to T and -trace_level is 10 or higher, then all DMCL traces are enabled. If -trace_detail is set to F and -trace_level is 10 or higher, DMCL traces for the API are enabled. Setting -trace_detail has no effect if the trace level is lower than 10.

All tracing information appears in one file whether you are using one dump and load operation or multiple dump and load operations for the job.

Manual dump file transfers

If you manually transfer the replication job dump file to the target repository, you must place the dump file in the following location on the target repository:

On Windows:

`%DOCUMENTUM%\share\temp\replicate\`*`target_db_name`*

On UNIX:

`$DOCUMENTUM/share/temp/replicate/`*`target_db_name`*
When you put the dump file in this location, be sure to give the file the same name as the original dump file. Within 60 minutes, the replication job agent will detect the presence of the file and automatically load the file.

Note: You cannot use a manual file transfer if the replication job is using multiple dump and load operations. (For more information about the use of multiple dump and load operations, refer to Multi-dump file replication jobs, page 45.)

Best practices for object replication

Observe the following rules regarding object replication to preserve structural and data consistency in the participating repositories:

1. If a folder is a target folder in one replication job and a source folder in another replication job, do not schedule the two jobs to run concurrently or overlapping.
2. Do not change job settings for the source and target after a job has been run.
3. If the same documents are replicated to more than one target folder in the same repository, the parameters of each job (security and fulltext settings, federation mode, and so forth) must match and the jobs must not run concurrently or overlapping. (Version mismatch errors may occur if the jobs are run concurrently).
4. To delete a replica from a target repository, first move the source object out of the source folder. If there are multiple jobs replicating the object to that target repository, remove the object from each source folder that replicates to that target repository. Then, use a Prune method and delete the replica's entire version tree in the target repository.
5. Do not create local folders in or link local folders to a target folder.
6. Do not move a replicated folder in a target repository to another location.

   It is acceptable to link a replicated folder to another, local folder or cabinet. However, do not remove the primary link to the target folder.
7. To delete a folder that contains replicas, first remove the replicas from the folder.

Chapter 5

Managing Single-Repository Models

This chapter contains procedures for administering the building blocks of a single-repository distributed configuration. This chapter includes the following topics:

- Adding a distributed component , page 107 , which describes how to add a site to a single-repository distributed configuration.
- Removing a distributed component, page 109 , which describes how to remove a site from a single-repository distributed configuration.
- Removing files from component storage areas, page 109 , which describes how to remove a file from a storage area that is part of distributed storage area.
- Troubleshooting surrogate get, page 110 , which provides information about correcting problems with the SurrogateGet tool.
- Overriding content-file server use, page 113 , which describes how to bypass content-file servers.
- Login failures in content-file server setups, page 117 , which describes the errors returned by login failures when content-file servers are implemented.
- Tuning query performance, page 117 , which discusses using statistics to maintain query performance.

Adding a distributed component

Use this procedure if you are adding a new remote site to an existing single-repository distributed installation. If you are setting up a new single-repository distributed installation, refer to Installing with Distributed Storage Areas, page 60 for instructions.

To add a new site to a distributed storage area installation:

1. Make sure the installation at the new site complies with the guidelines described in Guidelines, page 61 .
2. Make sure there is enough disk space at the new site to support the content storage requirements of a distributed storage area.

Refer to Estimating Disk Space, page 62 for guidance.

3. At the new site:
   a. Follow the installation instructions in *Installing Content Server* to install a content-file server and ACS server.

      Installing Content Server and the ACS server creates the required server config object for the new server, the acs config object in the repository for the ACS server, updates the acs.properties file for the new ACS server, and creates a file store storage area at the remote site.

   b. Start the remote site's server.

   c. Add the file store storage area at the remote site to the distributed store storage area as a component.

   d. Add the new component to the Far Stores list in the server config object for each of the other component sites in the distributed storage area.

   e. Optionally, reset the proximity values for the content-file server.

      The installation process automatically sets a remote server's proximity to its local connection broker as 9001 and the its proximity to the primary Content Server as 9010. You may want to reset the value to reflect your network's topography. Setting proximity values for Content Server projection to a connection broker, page 79, contains instructions and guidelines for choosing and setting appropriate proximity values.

4. Ensure that users and groups accessing distributed documents from the new site can be authenticated by servers at all sites, using the same mechanism.

   In a configuration using a distributed storage area, when a user connects to a repository, the client sends two connection requests: one to the data server and one to the content-file server. Each server must be able to authenticate the user's name and password using the same authentication mechanism.

5. Ensure that the machine hosting the Content Server is using UTC time and is synchronized with the other machines hosting distributed components.

   The servers at the remote sites (content-file server and ACS) use global login tickets configured to be valid for 5 minutes to connect to the primary site to fetch content. If the clocks in the host machines are not synchronized, the connection may fail if the host machine finds the login ticket invalid if the time difference is greater than 5 minutes.

6. To set up repository access for web-based clients at the remote site:
   a. Define at least one network location for the ACS server at the site.

      Creating Network Locations, page 66, contains instructions.

   b. Add the network location to the acs config object.

Add the location to the acs config object for each ACS server that can service requests from that network location. For instructions, refer to Adding network locations to an acs config object, page 68.

c. Define the network location's proximity to the ACS server or servers.

   Refer to Setting proximity values for network locations, page 69, for guidelines and instructions.

d. Define the ACS server's projection to a connection broker.

   Projecting an ACS server to connection brokers, page 68, provides instructions and guidelines for setting proximity values.

Removing a distributed component

It is possible to remove a site from a distributed storage area. Removing a site does not destroy the underlying directory or the files stored at the site. Use the content storage facilities of Documentum Administrator to remove a component from a distributed storage area.

Removing files from component storage areas

When a user deletes a document from the repository, regularly scheduled repository maintenance removes the associated files from the storage areas. However, on some occasions you may want to remove a file from a component storage area without removing the copies in other component areas and without destroying the associated SysObject, content object, or replica record object.

To remove a file without removing the associated objects, use the DELETE_REPLICA administration method. DELETE_REPLICA removes a file from a storage area and modifies the replica record object associated with the file's content object. If you simply remove the file using operating system commands, the replica record stills indicate that the file exists in the storage area. This can cause errors when users try to access the file.

DELETE_REPLICA has one argument, STORE. This string argument identifies the storage area that contains the file you want to remove.

Using Documentum Administrator is the recommended way to execute DELETE_REPLICA. However, you can also use DQL EXECUTE or the Apply method.

Using DQL EXECUTE or the Apply method

If you use DQL EXECUTE, the syntax is:

```
EXECUTE delete_replica FOR content_object_id
WITH store = 'storage_area_name'
```

You must specify the object ID of the file's associated content object. *storage_area_name* is the name of the storage object for the component storage area from which you are removing the file.

If you use Apply, the syntax is:

```
dmAPIGet("apply,session,content_object_id,DELETE_REPLICA,
STORE,S,storage_area_name")
```

When you use the Apply method, the name DELETE_REPLICA, the argument name, and the datatype must be in uppercase. It is not necessary to specify the storage area name in uppercase.

Troubleshooting surrogate get

The surrogate get feature provides automatic on-demand content replication within the components of a distributed storage area. It is intended for configurations that are not using shared directories.

Tracing surrogate get

You can use the trace file to trace dm_surrogateget method operations. You can also turn on Trace Launch to trace the method's invocations.

The trace file

If surrogate tracing is turned on, all dm_surrogateget method operations and timing information are traced in a file called sugg.trc. The file is stored in the %DOCUMENTUM%\bin ($DOCUMENTUM/bin) directory.

Turning on trace file generation

The sugg.trc file is not generated by default. You must manually turn on tracing for surrogate get.

To trace surrogate get operations:

1. Open the mthd6.ebs script in a text editor.

   The script is typically stored in %DM_HOME%\install\admin ($DM_HOME/install/admin).

2. Remove the single quote (') at the beginning of the following line:

   ```
   'ret% = dmAPIExec("trace,s0,10,sugg.trc")
   ```

3. Save your changes and exit the editor.

Tracing method invocations

To trace method invocations for surrogate get, set the Trace Launch property on the property page for the job in Documentum Administrator.

After you turn on Trace Launch for the method, the system logs dm_surrogateget method's command line in the server log file whenever the method is invoked.

Resolving problems

If the on-demand content replication of the SurrogateGet method is not working properly, check the following configuration items:

- The only_fetch_close attribute
- The get_method attributes
- connection broker projection targets
- Time settings on the participating host machines

The only_fetch_close attribute

The only_fetch_close attribute is defined for the distributed store object. It controls whether a server invokes the surrogate get method to fetch content files from component storage areas defined as far for the server.

Make sure that this attribute is set to TRUE. When the attribute is FALSE, the surrogate get method is not invoked.

If Trace Launch is turned on for surrogate get, you can look in the server log file to determine if the surrogate get method is invoked when the server needs to fetch a content file stored in a far storage area. The method's command line is recorded in the server log file when the method is invoked.

The get_method attributes

The server invokes the method defined in the get_method attribute for the distributed store object and the component storage areas. All must contain dm_SurrogateGet if you are using the SurrogateGet tool or the name of your user-defined surrogate get method. Check the attribute's value and modify it if necessary in the distributed store object and in the component storage areas.

You can use Documentum Administrator to check the attribute's value.

Connection broker projection targets

Make sure that the server at the site where the content file is stored is projecting correct connection information to the connection broker used by the server invoking the SurrogateGet method.

The connection broker used by the server invoking SurrogateGet must know about the server at the site where the content file is stored. In practice, this means that the server at each site in a single-repository distributed configuration must project its connection information to the connection broker at each of the other sites in the configuration.

Time settings

If the job is failing, check that the host machine on which the job runs and the host machine of the Content Server to which the job is connecting are both using UTC time and that the clocks are synchronized.

Surrogate get uses a login ticket to connect to servers to obtain content files. The login tickets are valid for five minutes. If the machine on which the dm_SurrogateGet method is running and the host machine of the server to which it connecting are not synchronized in time, the ticket may be invalid (timed out) on the target due to perceived differences in time. In such cases, the job will fail.

Overriding content-file server use

Content-file servers improve repository query performance for desktop users at remote sites. However, you may want to override this functionality on occasion. For example, the majority of the administration methods that manipulate content files and storage areas require the client to use one server for both content and data requests.

There are two ways to override the content-file server functionality:

- Set the use_content_server flag in the dmcl.ini file to FALSE
- Specify a server when you issue the Connect method

Using the use_content_server key

The use_content_server key is an optional key in the dmcl.ini file. If you set use_content_server to FALSE, desktop clients always connect to one server for both data and content requests. The key is TRUE by default.

With one exception, TRUE causes the client to connect to two different servers for data and content requests. The exception occurs when the connection request specifies the local server. Refer to Using both use_content_server key and Connect, page 114 for details.

Using the Connect method

The Connect method allows you to specify a repository using syntax that identifies which server to use for connection. The server is identified by the name of its server config object. For example, assume a repository called Demo has two servers, with server config objects named DemoA and DemoB. Either of the following Connect method calls is acceptable:

```
connect,Demo.DemoA,username,password
```

```
connect,Demo.DemoB,username,password
```

The first method call connects users to the server whose server config object is named DemoA. The second method call connects users to the server whose server config object is named DemoB.

Identifying the server in the Connect method causes the client to use that server for both content and data requests when use_content_server is NULL or FALSE. When use_content_server is TRUE, the server identified in the Connect method is used instead of the content-file server only if it is the user's local server. Refer to Using both use_content_server key and Connect, page 114 , below, for details of the interaction.

Using both use_content_server key and Connect

To illustrate how the use_content_server flag setting interacts with a server specification in the Connect method, assume that an enterprise has two servers, cat and mouse, installed on different machines and that:

- cat's proximity to the connection broker is defined as 2.
- mouse's proximity to the connection broker is defined as 9001

Table 5–1, page 114, describes the interaction between the use_content_server flag setting and a server specification in the Connect method.

Table 5-1. Interaction of use_content_server and Connect in connection requests

Connection request	Data server	Content-file server	Explanation
use_content_server is not set			
connect, *repositoryname*, *username*,*password*	cat	mouse	The connect method doesn't identify a server and use_content_server is not set, so the client connects to cat as the data server and mouse as the content-file server.
connect, *repositoryname*.cat, *username*,*password*	cat	cat	The Connect method identifies a server and use_content_server is not set to TRUE, so the client uses cat for both data and content requests.

Connection request	Data server	Content-file server	Explanation
connect, *repositoryname*.mouse *username*,*password*	mouse	mouse	The Connect method identifies a server and use_content_server is not set to TRUE, so the client uses mouse for both data and content requests.
use_content_server=TRUE			
connect, *repositoryname*, *username*,*password*	cat	mouse	The Connect method doesn't identify a server and use_content_server is set to TRUE, which reinforces the default behavior. The client connects to cat as the data server and mouse as the content-file server.
connect, *repositoryname*.cat, *username*,*password*	cat	mouse	The Connect method specifies a server for the connection. However, the client connects to two different servers because the use_content_server flag is TRUE.

Connection request	Data server	Content-file server	Explanation
connect, *repositoryname*. mouse *username,password*	mouse	mouse	The Connect method identifies the local server, which is also the default content-file server. Consequently, the client uses mouse for both data and content requests.
use_content_server=FALSE			
connect, *repositoryname*, *username,password*	mouse	mouse	The Connect method doesn't identify a server. Because use_content_server is FALSE, the client does not have to connect to different servers for data and content requests. Consequently, the client connects to the local server for both data and content requests.
connect, *repositoryname*.cat, *username,password*	cat	cat	The connect method identifies a server. The client connects to the server for both data and content requests because use_content_server is FALSE, so there is no requirement to use different servers.
connect, *repositoryname*.	mouse	mouse	The Connect method identifies

Connection request	Data server	Content-file server	Explanation
mouse *username*,*password*			a server. The client connects to mouse for both data and content requests because use_content_server is FALSE, so there is no requirement to use different servers.

Login failures in content-file server setups

When content-file servers are functioning in an installation, remote clients connect to both a data server and a content-file server by default when a session starts. Each connection requires a separate user authentication.

If a login failure occurs at the data server site, the failure occurs when the user tries to log in to the repository.

If a login failure occurs at the content-file server site, the user is logged in to the repository, but all content transfer requests fail.

Tuning query performance

As users add documents to a distributed storage area, the dmi_replica_record tables in the repository grow. This affects the performance of queries against the documents in the distributed storage area. To ensure the best performance, make sure that you are updating the statistics for the RDBMS tables underlying the repository. The query optimizer uses statistics to choose the best query plan.

You can generate statistics using either the Documentum Update Statistics system administration tool or the statistics tool provided by the underlying RDBMS. Update Statistics generates statistics for all tables in the repository. The RDBMS tools typically allow you to specify the table or tables for which you want to generate statistics.

For more information about Update Statistics, refer to Update Statistics , page 530 in the *Content Server Administrator's Guide*. Consult the documentation provided by your RDBMS vendor for information about running statistics directly in the RDBMS.

Chapter 6

Managing Multi-Repository Models

This chapter describes how to perform common administrative tasks for multi-repository distributed configurations. It includes the following topics:

- Manipulating a federation, page 120 , which contains procedures for adding and removing members, destroying a federation, or inactivating a governing repository.
- User operations, page 121 , which contains procedures for adding, modifying, or deleting a global user.
- Group operations, page 126 , which contains procedures for adding, modifying, or deleting a global group.
- Modifying object replication jobs, page 128 , which describes how to modify object replication jobs.
- Obtaining a list of jobs , page 128 , which provides instructions for obtaining a list of jobs defined in a repository.
- Scheduling federation jobs, page 129 , which describes how federation jobs are scheduled.
- Identifying the federation jobs operator, page 129 , which describes how the operator of the federation jobs is identified.
- Tracing ACL replication in federation jobs, page 130, which describes how to turn on tracing for ACL replication in federations.
- Job reports and log files, page 130 , which describes the reports and log files generated by the federation and object replication jobs.
- The dm_DistOperations job, page 132 , which describes the Distributed Operations job and its arguments.
- Monitoring and debugging federation jobs, page 133 , which describes how to monitor federation jobs and resolve any problems with them.
- Recovering from replication job failures, page 134 , which describes how to resolve failures in object replication jobs.
- Clearing the replicate_temp_store storage area, page 135, which describes how to handle the replicate_temp_store storage area if a replication job fails or is aborted.

Manipulating a federation

This section contains procedures for managing a federation. The procedures include:

Adding a member

A repository can belong to only one federation. Consequently, the member you are adding cannot currently belong to any federation.

Use the federation management facilities of Documentum Administrator to add a member repository to a federation. For instructions, refer to the Documentum Administrator online help.

Removing a member

You cannot remove a member repository while any federation jobs are running. Removing a member from a federation updates the federation object. When a federation job is running, the federation object is locked and cannot be updated.

Removing a member repository also sets the dm_FederationImport job in that repository to inactive and hidden. (The a_is_hidden attribute is set to TRUE for the job).

It is not necessary to make the global objects in the repository local objects. However, if you intend to add the repository to a different federation, we recommend that you make the global objects in the repository local objects first.

Use the federation management facilities of Documentum Administrator to remove a member repository from a federation. For instructions, refer to the Documentum Administrator online help.

Destroying a federation

You cannot destroy a federation while any federation jobs are running. Destroying a federation updates the federation object. When a federation job is running, the federation object is locked and cannot be updated.

After you destroy a federation, the former member repositories still have global users and groups. This does not affect the functioning of these repositories. However, we recommend that you make the global users and groups in these repositories local users and groups before adding the repositories to a new federation.

Use the federation management facilities of Documentum Administrator to destroy a federation. For instructions, refer to the Documentum Administrator online help.

Inactivating a governing repository

Inactivating a governing repository halts all federation-related jobs. To inactivate the governing repository, use the federation management facilities of Documentum Administrator. For instructions, refer to the Documentum Administrator online help.

User operations

This section contains information for managing the global users within a federation. Use the information in this section only if the global users are common only to the repositories participating in the federation.

A federation's global users are users who are present in all repositories in the federation. Global users are managed through the governing repository. You create a new global user in the governing repository, and the federation jobs automatically propagate the user to all member repositories. Similarly, if you delete a global user from the governing repository, a federation job automatically deletes the user from the member repositories.

Most of a user's attributes are global attributes. Global attribute values are the same in each repository in the federation. Global attributes must be modified through the governing repository, using Documentum Administrator.

A few attributes are local attributes. Typically, local attributes have different values in each member repository. However, there are four local attributes that can behave like global attributes. These four attributes are user_state, client_capability, worklfow_disabled, and user_privileges. When you create a global user, you can choose to propagate the values of these attributes to all member repositories when the attributes are changed in the governing repository.

Note: Sysadmin and Superuser user privileges are never propagated even if you choose to have the user_privileges attributes managed as a global attribute. Extended user privileges are not propagated either.

Information for the following procedures are included in this section:

- Creating a user , page 122

Perform the procedures using Documentum Administrator if your site is not using an LDAP directory server. The procedures are performed on the governing repository, through the federation management facilities of Documentum Administrator.

If your site is using an LDAP directory server to implement global users across all repositories, federated or not, use the procedures in the LDAP directory server documentation to add or change global user entries in the directory server. Do not use Documentum Administrator.

Creating a user

Use the federation management facilities of Documentum Administrator to add a global user to a governing repository. For instructions, refer to the Documentum Administrator online help. Chapter 9, Users and Groups, in the *Content Server Administrator's Guide* has information about user attributes.

Use the following guidelines when creating a user:

- The name of the user must be unique among the user and group names in the governing repository.
- A global user's user_name, user_os_name, user_address, and user_db_name must be compatible with the server operating system code pages (server_os_codepage) of all participating repositories. If the repositories are using different code pages, this means that those attributes must contain only ASCII characters.
- If a member repository has a local user with the same name as the global user, the global user will overwrite the local user.
- Assigning a global external ACL to the user as the user's default ACL is recommended.

  Note: If the ACL assigned to the user doesn't exist on the member repository, the server uses a system ACL named Global User Default ACL to create a custom ACL for the user. The permissions in Global User Default ACL are dm_world=3 (Read), dm_group=5 (Version), and dm_owner=7 (Delete). Content Server determines which group the user belongs to, creates a custom ACL from the Global User Default ACL that references that group, and assigns that ACL to the user.
- If you check a Send to members? checkbox, the value of the associated local attribute is propagated to member repositories, with one exception. The exception occurs for user privileges and extended user privileges. If the assigned user privilege is

Sysadmin or Superuser or any of the extended user privileges, even if the Send to members? checkbox is checked the privilege value is not propagated.

- The client capability setting is used by Documentum client applications such as DesktopClient™ to determine what functionality to make available to the user. Content Server does not use the client capability setting. For more information about the functionality provided with each capability level, refer to the client documentation.

After you create a global user on the governing repository, the federation jobs automatically propagate the new user to the member repositories. The global attribute values assigned in the governing repository are copied to each member repository. The local attributes are assigned default values. Table 6–1, page 123 describes the default values assigned to the local attributes in the member repositories.

Table 6-1. Default local attribute values for new users in federations

Attribute	Default setting
user_db_name	A null string
default_folder	If the member repository is the user's home repository, the default_folder attribute is set to the default_folder value specified when the user was created in the governing repository. If the repository is not the user's home repository, default_folder is set to /Temp.
user_privileges	The privilege level assigned when the user was created in the governing repository, unless the user was given Sysadmin or Superuser privileges. Superuser and Sysadmin privileges are not propagated automatically. This ensures that member repositories don't acquire unexpected users with these privileges. Only users who are installation owners retain Sysadmin or Superuser privileges when they are propagated to member repositories. All other users with Superuser or Sysadmin privileges in the governing repository are given no privileges in the member repositories (user_privileges=0).

Attribute	Default setting
user_state attribute	The user_state attribute is set to the value assigned when the user was created.
alias_set_id	The alias_set_id attribute is not assigned a default value.

Modifying user information

You can perform the following modifications for users:

- Modify global and local attribute values for a user
- Change the user's default ACL
- Change the user's state
- Change the user's home repository

Modify global users in the governing repository. For instructions, refer to the Documentum Administrator online help.

Renaming a user

Renaming a user affects more than just the user's dm_user object. The server stores a user's name in a variety of attributes for different object types. For example, r_creator_name and owner_name values are user names. User names also appear as values for the acl_domain and authors attributes.

When you run the job that renames a user, it attempts to change all references to that user in the repository. If some of the references are in the attributes of locked (checked out) objects, by default the job unlocks the objects (cancels the checkout) to make the change. If you don't want the job to unlock these objects, be sure to set the Checked Out Objects preference to Ignore before running the job.

You can choose whether to run renaming job immediately or the next time jobs are executed. In a large repository, the rename job can take a very long time to complete. It touches many tables and can use a large amount of resources. Running the job during peak usage hours is not recommended.

Rename a user through the governing repository. Use the federation management facility of Documentum Administrator. For instructions, refer to the Documentum Administrator online help.

Making a local user global

To change a local user to a global user you can use Documentum Administrator, DQL, or the API.

Using Documentum Administrator

If the user is local to the governing repository, you can simply change the user to global in the governing repository. The user is then propagated to all members in the next update operation.

If the user is local to a member repository, you must create the user as a new global user in the governing repository for the federation. The new user is automatically propagated to all the member repositories in the next update operation. As part of the process, the user is updated to global in the original member repository also.

Using DQL or the API

To use DQL to change a local user to a global user, use the UPDATE...OBJECT statement to set the value of the globally_managed attribute to TRUE for the user:

```
UPDATE "dm_user" OBJECT
SET "globally_managed"=TRUE
WHERE "user_name"='local_user_name'
```

To use the API to change a local user to a global user, use the Set method:

```
set,session,user_id,globally_managed,T
```

To obtain the user's object ID, you can use the Id method:

```
id,session,dm_user where user_name='local_user_name'
```

Making a global user local

To change a global user to a local user, use Documentum Administrator. Make the change in the federation's governing repository. The next federation update operation updates the user information in the member repositories, making the user local.

Deleting a global user

You must use Documentum Administrator to delete a global user. Use the federation management facilities to connect to the federation and delete a global user. For instructions, refer to the Documentum Administrator online help.

Deleting a user also deletes all registry objects that reference the user as the subject of an audit or event notification request.

Group operations

This section contains the information about procedures for managing global groups within a federation. This section includes information for the following procedures:

- Creating a group, page 126
- Modifying a group, page 127
- Renaming a group, page 127
- Deleting a group, page 128
- Making a global group local , page 128

Perform the procedures using Documentum Administrator if your site is not using an LDAP directory server. The procedures are performed on the governing repository, through the federation management facilities of Documentum Administrator.

If your site is using an LDAP directory server to implement global groups across all repositories, federated or not, use the procedures in the LDAP directory server documentation to add or change global group entries in the directory server.

Creating a group

All the attributes of a group are global. The values you define for a global group when you create it are propagated to all member repositories. The members of a global group must be global users or other global groups. To create a group, you must have Create Group, Sysadmin, or Superuser privileges.

A global group's group_address must be compatible with the server operating system code pages (server_os_codepage) of all participating repositories. If the repositories are using different code pages, this means that group_address must consist of only ASCII characters.

The name of the group must be unique among the user and group names in the governing repository.

Note: Content Server stores all group names in lowercase.

If a member repository has a local group with the same name as the global group, the global group will overwrite the local group.

You can use the federation management facilities in Documentum Administrator to create a global group. (Refer to the Documentum Administrator online help for instructions.)

Alternatively, you can execute DQL CREATE GROUP and UPDATE...OBJECT statements using IDQL or Documentum Administrator. After you create the group, update it to set the globally_managed attribute to TRUE. The syntax is:

```
CREATE GROUP name_of_group
WITH MEMBERS list_of_members

UPDATE "dm_group" OBJECT
SET "globally_managed"=TRUE
WHERE "group_name"='name_of_group'
```

Modifying a group

To modify a group, you must be one of the following:

- The group's owner
- A superuser
- A member of the group that owns the group to be modified (if the group is owned by a group)
- Identified in the group's group_admin attribute, either as an individual or a member of the group specified in the attribute

Only a superuser can change the ownership of an existing group. Only a superuser, the owner of a group, or a member of the group that owns the group can change the group_admin attribute of a group.

Use the federation management facilities in Documentum Administrator to modify a global group.

Renaming a group

To rename a global group, you must use Documentum Administrator. Refer to the Documentum Administrator online help for instructions.

Deleting a group

To delete a global group, you must use Documentum Administrator. You must be a superuser, the group's owner, or a member of the group that owns the group to delete a group. Refer to the Documentum Administrator online help for instructions.

Deleting a group also deletes all registry objects that reference the group as the subject of an audit or event notification request.

Making a global group local

Content Server does not support changing a global group to a local group. If you want to do this, create a matching local group and then delete the global group.

Modifying object replication jobs

You can modify a job after it is created. You can change the frequency with which a job is executed, suspend a job, and resume a job.

You can also change the settings for some attributes that affect the replicas created by the job. Changing these settings only affects replicas created after the change. Replicas created by previous executions of the job are not affected. For example, if you change the ACL assigned to replicas, subsequent executions will assign the new ACL to the replicas created by those executions, but existing replicas created by previous executions retain the original ACL.

If you want to change all replica objects in the target cabinet or folder, delete the replica objects and perform a full-refresh execution of the job.

Warning: Never change the source or destination folder. Running a job stores context information about the source and target cabinets and folders. If you need to change these parameters, define a new job.

Use the job management facilities in Documentum Administrator to change a job. For instructions, refer to the Documentum Administrator online help.

Obtaining a list of jobs

Use the job management facilities in Documentum Administrator to view a list of jobs defined for a particular repository. You can view all jobs in the repository or the jobs

defined by a category, such as those that operate on the repository as a whole or only content management jobs. For instructions on viewing a list of jobs in the repository, refer to the Documentum Administrator online help.

Scheduling federation jobs

Federation jobs manage global users, groups, and external ACLs in the federation. These jobs are executed automatically, in a sequence controlled by the dm_FederationUpdate job. (Refer to Appendix A, Federation Infrastructure , for a table describing the federation jobs.)

Federation jobs and the methods they invoke are installed as part of the Content Server installation process. The jobs are installed in the inactive state with the is_hidden attribute set to TRUE.

When you create a federation, only certain jobs are activated:

- In the governing repository, the dm_FederationUpdate and dm_FederationStatus jobs are activated and made visible. This occurs automatically if you check the Propagate Global Objects to Member repositories checkbox when you create the federation. Otherwise, you must do it manually.
- In the member repositories, only the dm_FederationImport job is activated and made visible.

All other federation jobs in the governing repository and member repositories remain inactive and invisible. Their execution is controlled by the dm_FederationUpdate job running in the governing repository. By default, the dm_FederationUpdate job runs once a day, during the night. When it executes, it runs the methods associated with the jobs in the correct sequence for each member repository.

You can change the scheduling for the dm_FederationUpdate job on the governing repository. However, we strongly recommend that you do not activate any of the other federation jobs. Doing so will cause them to run out of sequence.

Identifying the federation jobs operator

The federation jobs operator is the person who receives email sent by the system reporting the status of the federation jobs. By default, this is the user identified in the operator_name attribute of the server's server config object. (The value in operator_name defaults to the repository owner.)

If the federation administrator is not the user identified in operator_name, reset the queueperson argument for the federation jobs to the federation administrator's user

name. Use Documentum Administrator to display the Properties screen for each job and reset the queueperson argument.

Tracing ACL replication in federation jobs

The dm_FederationUpdate job calls the dm_ACL_Repl*repository_name* job to replicate ACLs to member repositories. There is one dm_ACL_Repl job for each member repository. By default, the tracing level in the dm_ACL_Repl jobs is set to 0.

To turn on tracing for the ACL replication jobs, use IAPI. For each member repository, do the following:

1. Retrieve the job object from the repository:

   ```
   API>retrieve,c,dm_job where object_name='dm_ACL_Repl<repository_name
   ...
   <job_id>'
   ```

   where *repository_name* is the name of the repository. The method returns the job object's object ID.

2. Set the method trace level to 10.

   ```
   API>set,c,job_id,method_trace_level
   SET>10
   ...

   OK
   ```

3. Save the job object:

   ```
   API>save,c,job_id
   ```

The tracing information is recorded in the job report generated by the federation update job.

Job reports and log files

Federation jobs and object replication jobs generate reports and log files.

Job reports

A job report summarizes the results of a job in an easily interpreted format. To view reports for federation jobs, you connect to the federation. To view job reports for

object replication jobs, you must connect to the repository that executed the job. Use Documentum Administrator to view job reports.

In the repository, job reports are stored in /System/Sysadmin/Reports. Each time a job executes, the job report in this location is versioned.

The reports are also stored in the file system in %DOCUMENTUM%\dba\log*repository_id*\sysadmin ($DOCUMENTUM/dba/log/*repository_id*/sysadmin). In this directory, the report name has the format *job_name*Doc.txt. The reports in this directory are overwritten each time the job executes.

Federation job reports are also stored in %DOCUMENTUM%\share\temp\ldif*repository_name* ($DOCUMENTUM/share/temp/ldif/*repository_name*) for access through Documentum Administrator.

Job log files

Each job execution generates a log file in addition to a report. The log file is typically used for debugging. The log file is the first place to look if you want to know how long a job took to complete or why a job did not run to completion.

For object replication jobs, the log file information includes:

- Whether the job is working in Phase 1, 2, or 3
  - Phase 1 is when the dump file is created
  - Phase 2 is when the file is transferred to the target
  - Phase 3 is when the dump file is loaded into the target repository
- The current job status
- The elapsed time of the job
- The incremental time of the job.
- The incremental time is the length of time since the last status message.
- The elapsed time is the length of time since the job began.

To view federation job log files, you connect to the federation. To view the log files for object replication jobs, you must connect to the repository that executed the job. Use Documentum Administrator to view job log files.

In the repository, job log files are stored in /Temp/Jobs/*job_object_name*. The log files in this directory are versioned when the jobs execute.

The log files are also stored in the file system in %DOCUMENTUM%\dba\log*repository_id*\sysadmin ($DOCUMENTUM/dba/log/*repository_id*/sysadmin). The name of the log file has the format *job_name*Trace.txt. The log file in this directory is overwritten each time the job executes.

Federation job log files are also stored in %DOCUMENTUM%\share\temp\ldif*repository_name* ($DOCUMENTUM/share/temp/ldif/*repository_name*) for access through Documentum Administrator.

The dm_DistOperations job

The dm_DistOperations job performs inter-repository distributed operations. These tasks include:

- Propagating distributed events (dmi_queue_items) across repositories
- Creating checkout references for remote checkout operations
- Refreshing reference links

This job is one of the system administration tools installed with the Content Server installation. (The other tools are described in the *Content Server Administrator's Guide*.)

Like replication jobs, the dm_DistOperations jobs generates reports and log files. The report is saved in the repository in /System/Sysadmin/Reports/DistributedOperations. For more information about where to find the reports and log files generated by system-defined jobs, refer to Job reports and log files, page 130 .

The dm_DistOperations job runs every 5 minutes at each repository. Table 6–2, page 132 describes the job's arguments.

Table 6-2. dm_DistOperations job arguments

Argument	Value	Description
-process_queued _operations	Boolean	If TRUE, the job processes all dmi_queue_item objects that have remote_pending set to TRUE. The default value is TRUE.
-process_refreshes	Boolean	If TRUE, the job checks for and performs needed reference links refreshes. The default value is TRUE.

Argument	Value	Description
-refresh_checks_per _cycle	integer	Defines the maximum number of reference links to check for needed refresh in each job execution. The default value is 100.
superuser and passwords for each repository	string	The job must have a valid superuser name and password for each of the participating repositories. These are entered as arguments using the format: *-server_config_name user_name,password*[*,domain*] server_config_name is the object name of the server config object for the repository. The superuser and password must be valid for the repository. They can be the same or different for each repository. domain is required only when unique domain user is enforced.

Monitoring and debugging federation jobs

All federation jobs return one of three values: success, warning, or failure.

A warning indicates that the job encountered a nonfatal error. The job continued to execute but some errors may have occurred during its execution. Warnings typically occur in the following situations:

- The status or copy methods cannot establish a session with a member repository to retrieve the status or copy global objects.

- The status or copy methods cannot validate the dm_federation object in a member repository.
- The import method on a member repository experienced difficulties such as:
  - The default folder could not be created
  - The user's default ACL was not found, necessitating the use of the default system ACL

Failures indicate a fatal error that stopped the job's execution. Fatal errors may leave the dm_federation object locked in the repository. In such cases, be sure to manually check in the federation object.

If a warning or failure occurs:

1. Review the job report.
2. If necessary, review the job log file.

Recovering from replication job failures

Object replication failures generally result from an improperly configured source or target installation or from infrastructure inadequacies, such as insufficient network bandwidth or CPU capacity. If a replication job fails to complete successfully:

- Examine the Properties for the job to determine the last status for the job.
- Review the log file for the job for detailed information on why the job failed.
- If the error message in the trace file suggests that something is configured incorrectly, reconfigure it appropriately and rerun the job.

  For example, if the replication job calls for replicas to be stored in a nonexistent file store, create the appropriate file store and rerun the job. (Refer to Creating a job, page 146 the *Content Server Administrator's Guide* for instructions on running a job outside of its scheduled time.)
- Make certain that adequate machine and network resources were available when the job failed.
- If the message in the log file suggests that there were insufficient resources available, rerun the job to see if the same failure occurs.

In general, replication job failures are not destructive. They do not harm existing replicas in the target repository. The next successful execution of the job will clean up after the failure.

Correcting the cause of the failure should restore successful replication. However, as part of the process of correcting the problem and rerunning the job to check it, you may have to reset the job schedule. Refer to Scheduling jobs, page 148 of the *Content Server Administrator's Guide* for information about scheduling jobs.

Clearing the replicate_temp_store storage area

If an object replication job is aborted or uncompleted, Content Server may leave data in the replicate_temp_store storage area. (This area is used to hold the dump files created by object replication jobs for the duration of the job.) You can remove data left behind under such circumstances.

The directory location of the replicate_temp_store storage area for a particular repository is %DOCUMENTUM%\data\replicate_temp_store*repository_id* ($DOCUMENTUM/data/replicate_temp_store/*repository_id*), where *repository _id* is the hex value.

Appendix A

Federation Infrastructure

This appendix contains a table that describes the jobs, methods, files, and objects that support federations.

Table A-1. Objects that support federations

Object name	Type	Description
dm_FederationUpdate	job/method	Executes on the governing repository to run all the other methods in sequence, to push changes to users, groups, and ACLs to the member repositories. The job and its associated method have the same name.
dm_FederationStatus	job/method	Polls all the member repositories to determine the current status of the dm_FederationImport jobs on the member. The job and its associated method have the same name.
dm_ldif_status	method	Called by dm_FederationStatus to poll the member repositories.

Object name	Type	Description
dm_FederationExport	job/method	Exports the user and group information from the governing repository to an LDIF file. The job and its associated method have the same name.
dm_ldif_export	method	Called by dm_FederationExport to generate an LDIF export file.
dm_FederationCopy	job/method	Transfers LDIF files to member repositories. The job and its associated method have the same name.
dm_ldif_copy	method	Called by dm_FederationCopy to transfer the LDIF files from the governing repository to member repositories.
dm_FederationImport	job/method	Imports an LDIF file into member repositories.
dm_ldif_import	method	Called by dm_FederationImport to import the users and groups from the LDIF file into a member repository.
dm_ACLReplication	job/method	A staging job that sets external ACLs for replication.
dm_ACLRepl_*repository*	job	Replicates external ACLs to member repositories. There is one job for each member repository. *repository* is the first 19 bytes of the repository name.

Object name	Type	Description
ldif	directory	Contains the LDIF files generated by dm_FederationExport. The directory is found in %DOCUMENTUM%\share\temp ($DOCUMENTUM/share/temp).
dm_federation_log	registered table	Tracks certain changes to users and groups. This is the change log.

Index

D

E

F

G

H

I

O

P

Q

R

S

T

U

V

W