Rational. Synergy

Synergy Distributed Guide

IBM Rational Synergy

Distributed

Release 7.1

Before using this information, be sure to read the general information under "Appendix: Notices" on page 173.

Contents

1 Introduction

This chapter explains what you should know before you read this manual. This manual describes the features of IBM® Rational ®Synergy Distributed. The Rational Synergy Distributed CM solution is referred to as Distributed Configuration Management (DCM).

Intended audience

This book is intended for the DCM manager and Configuration Management (CM) administrator.

The DCM manager coordinates the data that is shared among databases in a DCM cluster. This person will be most interested in the methodology discussions, descriptions of operations, and "Sample DCM Scenario: Master and Satellite" on page 79.

The CM administrator oversees the daily operation of DCM tools. This person will be most interested in: "Planning a DCM Cluster" on page 19, "Administering DCM Databases" on page 59, and Advanced DCM Topics.

Sources of DCM information

The published DCM information is located in this book and in the Synergy Classic *Help*. Generally speaking, this document gives an overview of DCM and its operations. The specific steps needed to perform these actions are shown in the *Help*.

This document contains the latest information available at the time of publication. For late additions, check the Rational Synergy Readme located on the IBM Rational Synergy Information Center Web site (`http://publib.boulder.ibm.com/infocenter/rsdp/v1r0m0/index.jsp`).

Contacting IBM Rational Software Support

If the self-help resources have not provided a resolution to your problem, you can contact IBM® Rational® Software Support for assistance in resolving product issues.

Note If you are a heritage Telelogic customer, a single reference site for all support resources is located at http://www.ibm.com/software/rational/support/telelogic/

Prerequisites

To submit your problem to IBM Rational Software Support, you must have an active Passport Advantage® software maintenance agreement. Passport Advantage is the IBM comprehensive software licensing and software maintenance (product upgrades and technical support) offering. You can enroll online in Passport Advantage from http://www.ibm.com/software/lotus/passportadvantage/howtoenroll.html

- To learn more about Passport Advantage, visit the Passport Advantage FAQs at http://www.ibm.com/software/lotus/passportadvantage/brochures_faqs_quickguides.html.
- For further assistance, contact your IBM representative.

To submit your problem online (from the IBM Web site) to IBM Rational Software Support, you must additionally:

- Be a registered user on the IBM Rational Software Support Web site. For details about registering, go to http://www.ibm.com/software/support/.
- Be listed as an authorized caller in the service request tool.

Submitting problems

To submit your problem to IBM Rational Software Support:

1. Determine the business impact of your problem. When you report a problem to IBM, you are asked to supply a severity level. Therefore, you need to understand and assess the business impact of the problem that you are reporting.

   Use the following table to determine the severity level.

Severity	Description
1	The problem has a *critical* business impact: You are unable to use the program, resulting in a critical impact on operations. This condition requires an immediate solution.
2	This problem has a *significant* business impact: The program is usable, but it is severely limited.

Severity	Description
3	The problem has *some* business impact: The program is usable, but less significant features (not critical to operations) are unavailable.
4	The problem has *minimal* business impact: The problem causes little impact on operations or a reasonable circumvention to the problem was implemented.

2. Describe your problem and gather background information, When describing a problem to IBM, be as specific as possible. Include all relevant background information so that IBM Rational Software Support specialists can help you solve the problem efficiently. To save time, know the answers to these questions:

- What software versions were you running when the problem occurred?

  To determine the exact product name and version, use the option applicable to you:

  - Start the IBM Installation Manager and select **File** > **View Installed Packages**. Expand a package group and select a package to see the package name and version number.

  - Start your product, and click **Help** > **About** to see the offering name and version number.
- What is your operating system and version number (including any service packs or patches)?
- Do you have logs, traces, and messages that are related to the problem symptoms?
- Can you recreate the problem? If so, what steps do you perform to recreate the problem?
- Did you make any changes to the system? For example, did you make changes to the hardware, operating system, networking software, or other system components?
- Are you currently using a workaround for the problem? If so, be prepared to describe the workaround when you report the problem.

3. Submit your problem to IBM Rational Software Support. You can submit your problem to IBM Rational Software Support in the following ways:

- **Online**: Go to the IBM Rational Software Support Web site at https://www.ibm.com/software/rational/support/ and in the Rational support task navigator, click **Open Service Request.** Select the electronic problem reporting tool, and open a Problem Management Record (PMR), describing the problem accurately in your own words.

For more information about opening a service request, go to http://www.ibm.com/software/support/help.html

You can also open an online service request using the IBM Support Assistant. For more information, go to http://www.ibm.com/software/support/isa/faq.html.

- **By phone**: For the phone number to call in your country or region, go to the IBM directory of worldwide contacts at http://www.ibm.com/planetwide/ and click the name of your country or geographic region.
- **Through your IBM Representative**: If you cannot access IBM Rational Software Support online or by phone, contact your IBM Representative. If necessary, your IBM Representative can open a service request for you. You can find complete contact information for each country at http://www.ibm.com/planetwide/.

Other information

For Rational software product news, events, and other information, visit the IBM Rational Software Web site.

Rational Synergy documentation

Rational Synergy includes a complete set of product documentation. Visit the Rational Synergy Information Center site at `http://publib.boulder.ibm.com/infocenter/rsdp/v1r0m0/index.jsp`, where you can access the electronic versions of documents or print the documents you need.

How this manual is organized

This book is arranged as follows:

- "Overview of DCM" on page 7

  This section gives an overview of DCM and briefly describes some of its features.

- "Planning a DCM Cluster" on page 19

  This section explains how DCM works and how to develop an implementation plan.

- "Setting Up a DCM Database" on page 37

  This section shows how to set up a DCM database and perform transfers to other databases.

- "Using DCM" on page 47

  This section explains how to perform the generate and receive operations.

- "Administering DCM Databases" on page 59

  This section describes how to administer DCM databases.

- "Sample DCM Scenario: Master and Satellite" on page 79

  This section provides an example of how to set up and use a DCM cluster.

- "Tracking Distributed Change Requests" on page 97

  This section explains how to manage distributed change requests.

- "Advanced DCM Topics" on page 115

  This section describes advanced DCM features typically used by administrators.

- "Glossary" on page 163

  The *Glossary* defines terms that are used throughout this book.

References used in this manual

Most Rational Synergy Distributed operations must be performed from the Synergy Classic interface. This document contains references to various Rational Synergy dialogs. Except where noted, all references to specific dialogs, dialog fields and options, and information contained in *Help* pertain to the Synergy Classic interface. For information about command line operations, see the *Classic CLI Help*. This help can be used while running the Classic CLI and is also available on the Rational Synergy Information Center Web site (`http://publib.boulder.ibm.com/infocenter/rsdp/v1r0m0/index.jsp`).

Conventions used in this guide

This section describes the conventions used in this document.

Command and path syntax

This document shows the DCM command options with the UNIX® dash (-) delimiter. If a UNIX client is being used, substitute the UNIX dash (-) for the forward slash delimiter in command options, and a forward slash for the backslash in path names. If the UNIX path differs from the Windows® path, both paths are given.

Fonts and symbols

The table below describes the typeface and symbol conventions used in this guide..

Typeface	Description
Italic	Used for book titles and terminology. Also designates names of roles (*developer*), states (*working*), groups (*ccm_root*), and users (*john*).
Bold	Used for dialog box names and options, items that you can select and menu paths, also used for emphasis.
`Courier`	Used for commands, filenames, and directory paths. Represents command syntax to be entered verbatim. Signifies computer output that displays on-screen. Also used for the names of attributes (`modify_time`), functions (`remote_type`), and types (`csrc`).
`Courier Italic`	Represents values in a command string that you supply. For example, (*`drive`*`:\`*`username`*`\`*`commands`*).

This document uses the following additional conventions:

Note Contains information that should not be overlooked.

Caution Contains critical information that must be observed or damage to the database or system could result.

2 Overview of DCM

Introduction

This section gives an overview of DCM and briefly describes some of its features. The following topics are included:

About DCM

DCM is a *distributed configuration management* tool. DCM lets multiple Rational Synergy databases share objects that originate, and are being changed in parallel, in any Rational Synergy database in the world that has been initialized to use DCM.

DCM uses standard Rational Synergy databases to share software development objects. These objects include: the files used to build products, the products themselves, projects, folders, change requests, and tasks. DCM uses task-based CM features that let you group and send logical sets of changes to other databases. However, your software development does not need to be task-based in order to use this tool.

The following are reasons for using DCM:

- To support multiple sites
- For localization or porting
- To split up databases
- For different levels of testing
- For security purposes

Whether you are sharing changes made locally in different databases, or changes made by development teams that are located on different continents, DCM lets you control the process.

Note This document assumes you are familiar with Rational Synergy. At a minimum, you should understand CM methodology and know how to perform task-based CM build management activities.

Features

DCM is designed for distributed and parallel development. Its main features are described in this section.

Transparency

DCM has very little impact on developers' daily operations. Its databases can be set up so that minimal management is required.

Integrating with the existing configuration management suite

DCM has the same user interface as Rational Synergy and supports command line operations. Once a Rational Synergy database is initialized to use DCM, its data is automatically displayed in the appropriate Rational Synergy dialogs. Developers see this data in the dialogs, but they do not need to change their work habits.

Flexibility

DCM lets you send source files, projects, folders, and tasks, in any combination, to any Rational Synergy database that is initialized to use DCM. There are no restrictions on how objects are grouped. DCM works with task-based methodologies. It also supports many distributed methodologies.

Support for parallel development

DCM supports parallel development using the Parallel Notification and Merge and Compare features of Rational Synergy. If a DCM database receives a new object version that has a parallel version, the developer who owns or created the parallel version is notified by email (see Parallel checking). Then, use the Merge and Compare feature in Rational Synergy to resolve the parallels.

Time zone independence

DCM supports multiple databases that are located in different time zones. When data is transferred to a destination database, DCM interprets and records all times in Greenwich Mean Time (GMT), thus ensuring time zone independence.

Terms and concepts

Before using DCM, you should be familiar with the following terms and concepts.

DCM database

A *DCM database* is any Rational Synergy database that has been initialized to use DCM (for more information, see "Initializing DCM" on page 11).

Source and destination databases

A *source database* is any DCM database that sends data to another database. A *destination database* is any DCM database that receives data from another database. A given database can be a source database, a destination database, or both.

DCM cluster and database IDs

A *DCM cluster* is any group of DCM databases that shares data. Each database in a cluster is manually given a unique database ID (`dbid`). database IDs are used to define source and destination databases. They are also used to make each object version in the cluster unique. For example, DCM assigns the Irvine database ID to a `main.c` object created in the `irvine` database. This makes the object different from a `main.c` object created in the `san_jose` database.

Destination database definitions, transfer sets, and transfer packages

DCM uses the *destination database definition* to send data to the destination database. This definition names, describes, and provides transfer information about the destination database.

A *transfer set* is a related group of objects that can be sent to a destination database.

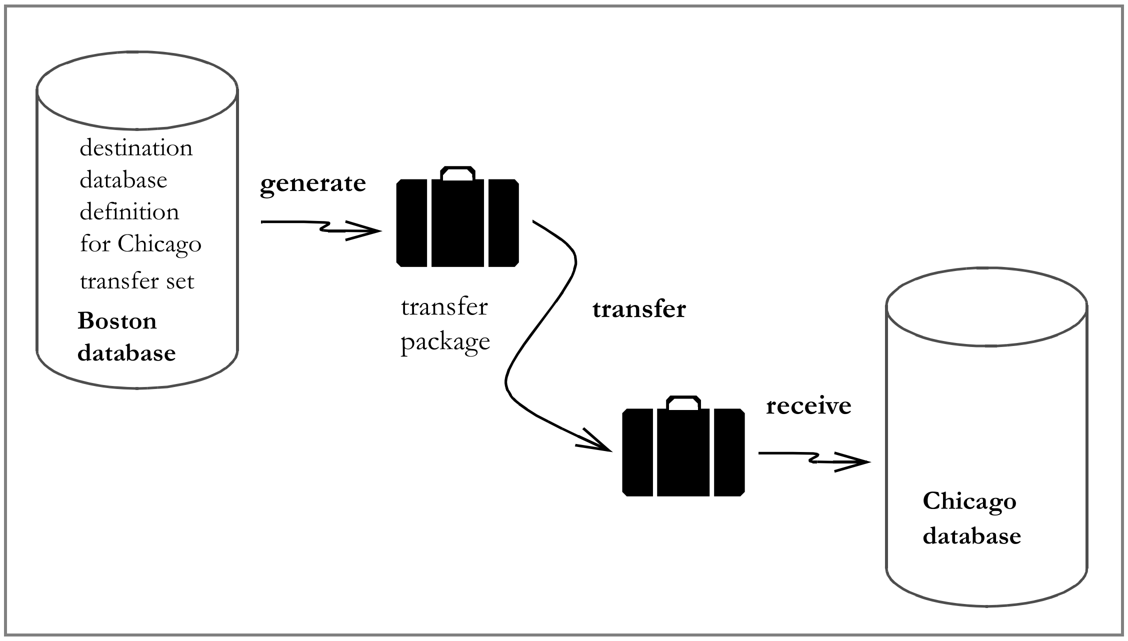
To send data to a destination database, the following occurs at the source database: 1) A destination database definition is created for each database to which data is to be transferred, 2) A transfer set is defined for each group of objects that is to be sent, 3) Each transfer set is paired with a destination database definition (each pairing is called a *transfer package*), and 4) The data is sent.

Generating, transferring, and receiving

Three DCM operations must be performed to send data from a source to a destination database: *generate*, *transfer*, and *receive*. The *generate* operation creates a transfer package. This package is sent from the source to the destination database using the *transfer* operation. The transferred data is then loaded into the destination database using the *receive* operation.

DCM lets you combine the generate, transfer, and receive operations. With one action you can either: 1) Generate and transfer a transfer package, or 2) Generate, transfer, and receive a transfer package (see "How to use automatic receive" on page 142). You also have the option of performing each operation separately.

The figure below shows the generate, transfer, and receive process for a transfer package that was sent from the Boston database to the Chicago database.



Initializing DCM

In order for a Rational Synergy database to be usable by DCM, a *DCM initialization* must be performed on the database. This operation sets DCM properties for the database, objects currently in the database, and any objects that will be added to the database in the future. For more information, see "Initializing a database" on page 39.

Properties assigned to DCM objects

All objects in a DCM database are automatically assigned special properties. Some of these properties are: Local To, From, Modifiable In, and Work in DB. DCM uses these traits to preserve the uniqueness and histories of all objects in a DCM cluster. The properties are described as follows:

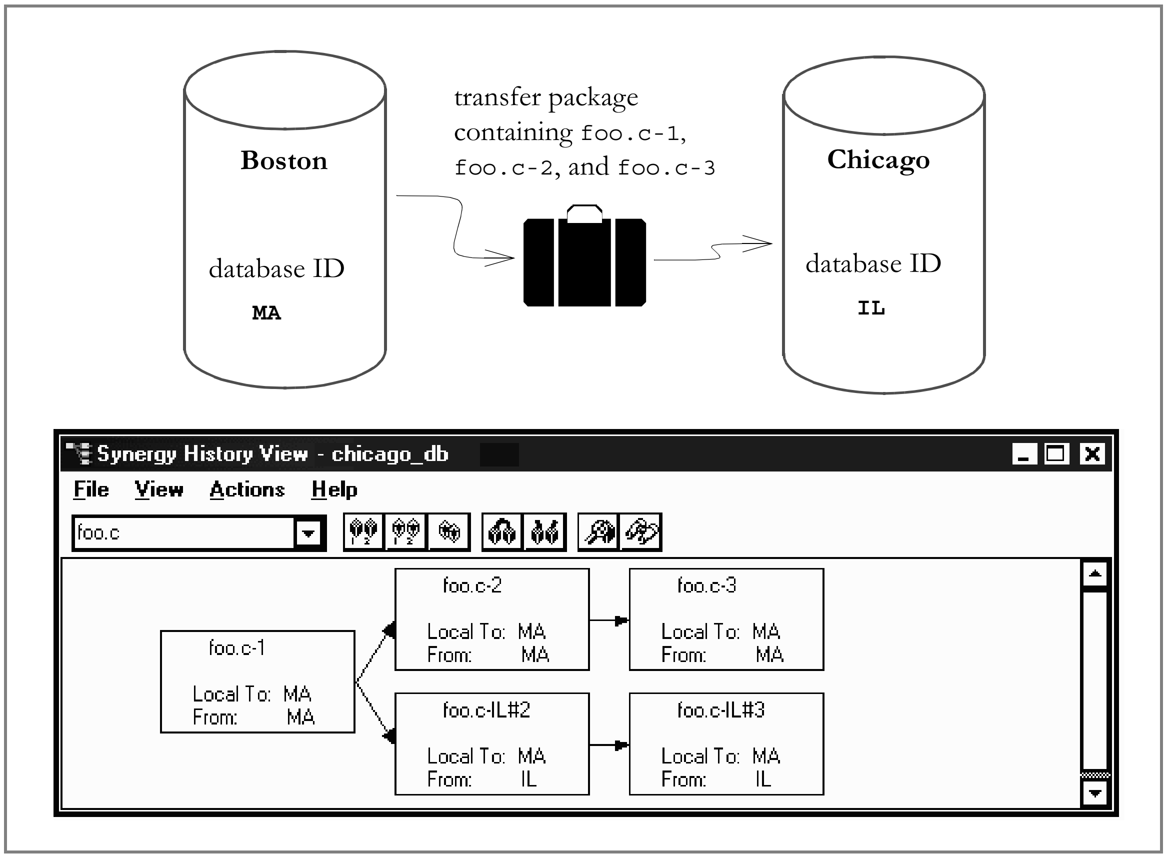
Local To – The value of the Local To property (the `local_to` attribute) is the database ID of the database in which the object is either initialized or created. The Local To value remains with all versions of an object. It never changes during the entire history of the object. All successors of a DCM object inherit its Local To value regardless of where the new versions are checked out.

From – The value of the From property (the `created_in` attribute) is the ID of the database **from which** the object version is checked out. The From value is assigned only to a particular version; it is not inherited by successors.

Note When an object is created (by a user or during DCM initialization), the value of its From property is set to the ID of the database in which it is created. Thus, the Local To and From values are always identical for the first version of a DCM object.

Modifiable In or ***Work in DB*** - The value of the Modifiable In property (the `modifiable_in` attribute) is the ID of the database that controls the object. DCM only allows a specific object to be modified in one database in the DCM cluster. In all other databases, that object cannot be modified, with the exception of transitioning it to later states. If the Modifiable In property does not exist, then the controlling database is determined by the From (`created_in`) property. Usually, files, directories, and projects do not have a Modifiable In property by default. The concept of controlling database is discussed further in Database and handover of control.

The figure below shows a Rational Synergy Class **History View** of an object called `foo.c`. It was created in the Boston database and, therefore, the Local To and From values for the first version (`foo.c-1`) are both set to the ID of the Boston database (MA) — see preceding Note for further explanation. When `foo.c` is checked out **from** the Chicago database, the From value is set to the ID of the Chicago database (IL). When `foo.c` is checked out **from** the Boston database, the From value is set to the ID of the Boston database (MA). As shown in the figure, the Local To value never changes during the entire history of an object.


Boston
database ID
MA
transfer package containing foo.c-1, foo.c-2, and foo.c-3
Chicago
database ID
IL
Synergy History View - chicago_db
File View Actions Help
foo.c
foo.c-1 Local To: MA From: MA
foo.c-2 Local To: MA From: MA
foo.c-3 Local To: MA From: MA
foo.c-IL#2 Local To: MA From: IL
foo.c-IL#3 Local To: MA From: IL

In addition to these properties, all DCM objects are assigned attributes that ensure their uniqueness. For example, each object version is linked to a unique cluster ID. These and other object traits provide security features that do the following:

- Protect objects from being overwritten in their controlling database by data from other databases.
- Allow the control of an object to be handed over from one database to another. This is accomplished using the modifiable_in attribute.

- Prevent changes, made to a non-static object that has been copied across databases, from being lost or overwritten.

Master builds, master projects, and master folders

Before DCM is used, a plan for the flow of information between databases must be developed. This is called the methodology. There are many standard methodologies from which to choose. For example, Master and Satellite is a typical DCM methodology in which one database (the master) performs the builds. These builds, called *master builds*, combine the changes that have been made in all the other databases (the satellites) in the cluster.

Projects that are used to perform the master builds and tests are called *master projects*. Most DCM installations include a master integration *prep* project. In addition, they may have master *prep* projects that represent other levels of building and testing.

When task-based CM is used, folders are included in the project update properties. Folders that are in the update properties of master projects are called *master folders*.

Note More DCM terms are defined in the "Glossary" on page 163.

Object naming conventions

DCM uses special naming conventions for objects that have been initialized or created in a DCM database. The naming convention that is used depends on the type of the object, and whether or not the object is local to the database in which it is being used.

As explained below, DCM-specific naming conventions are used in the following object attributes: versions, instances, and task names.

Versions

When you check out an object that is not local to the database from which it is checked out, DCM prepends the database ID and DCM delimiter to the new version number. For example, if you are in the `IL` database and check out from `foo.c-1`, where `foo.c-1` is local to the `MA` database, the next version is `foo.c-IL#2` (where `IL` is the database ID, and `#` is the DCM delimiter).

Note The database ID and DCM delimiter are set when the database is initialized to use DCM. The default DCM delimiter is "`#`". The following characters can also be used: "`!`", "`~`", or "`=`".

The next version of an object depends on the current version, and whether or not the database from which it is checked out is the home database of the object. The following table shows how versions are assigned to local and non-local objects.

Local To	**Initial Object Version**	**Checked Out From**	**New Object Version**
MA	`foo.c-1`	MA (home)	`foo.c-2`
MA	`foo.c-1`	IL	`foo.c-IL#2`
MA	`foo.c-IL#2`	MA (home)	`foo.c-3`
MA	`foo.c-IL#2`	IL	`foo.c-IL#3`

Instances

DCM prepends the home database ID and DCM delimiter to all instances of versioned DCM objects. For example, an object that is local to the MA database might have an instance value of `MA#1` (where `MA` is the database ID, `#` is the DCM delimiter, and `1` is the instance).

Task names

For display purposes, DCM prepends the home database ID and DCM delimiter to all task numbers.

Projects and projectspecs

Projects in a DCM-initialized database use an instance value that includes the DCM database ID. For example, a project created in the `MA` database will have an instance value of `MA#1`, whereas a project created in the `IL` database will have an instance value of `IL#1`.

If a project is specified by its name and version only, Rational Synergy assumes a default instance of the current DCM database ID plus the DCM delimiter plus "1". For example, a project named `hello` with version `1` created in the `MA` database, will be displayed as and can be specified by `hello-1`. The same project when replicated to the `IL` database will be displayed there as `hello-1:project:MA#1` and has to be specified there by its full objectname. This is necessary because there could be a project named `hello` with version `1` that was created in the `IL` database that is a different and unrelated project.

Roles

Some DCM operations require the *dcm_mgr* role while others need the *ccm_admin* role. If a user does not have at least one of these roles, all DCM dialogs are read-only.

The *dcm_mgr* role is required for the following operations:

- Defining destination databases
- Defining transfer sets
- Adding objects to transfer sets
- Generating a transfer package

The *ccm_admin* role is needed for any operation that might change the properties of non-modifiable objects. These operations include:

- Initializing a database to use DCM – During this operation, DCM might **add** properties to non-modifiable objects
- Changing the DCM database ID or DCM delimiter
- Receiving a transfer set – During this operation, DCM might **add** or **change** properties of non-modifiable objects

If you try to perform a DCM operation while in the wrong role for that operation, but you have access to the role in the database, DCM automatically switches your role to the correct one. This is called *dynamic role switching*. When the operation is finished, DCM switches back to the original role.

3 Planning a DCM Cluster

Introduction

This section explains how to plan a DCM implementation. The following topics are discussed:

- "Developing a DCM methodology and replication topology" on page 20
- "Establishing disk space requirements" on page 25
- "DCM planning worksheet" on page 27
- "Establishing common database properties" on page 33
- "Establishing common database parameters" on page 34
- "Synchronizing engines and servers" on page 36

Developing a DCM methodology and replication topology

Before DCM can be used effectively, a plan for the flow of information between databases must be developed. There are two related parts to this:

- A *DCM methodology* defines what information is to replicated between which database(s), for what purpose, and how the software component or application is built from such shared changes. Some typical DCM methodologies include:

  "Master and Satellite methodology" on page 22

  "Publish and Subscribe methodology" on page 23

  "Peer-to-Peer methodology" on page 24

- A *replication topology* defines whether the databases replicate data directly with each other, or through intermediate hub databases. Some typical replication topologies include:

  Point-to-point topology

  Each database generates a transfer package for a database to which information is to be sent. To learn how this topology is used to transfer change requests, see Point-to-Point topology.

  Hub and spoke topology

  Spoke databases do not replicate directly between each other. Instead, they replicate via one or more hub databases. To learn how this topology is used to transfer change requests, see Hub and Spoke topology. In a DCM cluster that contains databases running on earlier releases of Rational Synergy, any hub databases should be at the latest release of Rational Synergy. See Supported transfers from Rational Synergy Distributed releases prior to 7.1 for more specific details.

Together, the DCM methodology and replication topology define the nature and direction of transfers. To help you define your methodology and topology, answer the following questions:

- What is the purpose of the DCM cluster?
- What is the purpose of each database in the DCM cluster?
- What is the nature of the object sharing (for example, to distribute tested projects, or to share products)?
- Which objects are local to a given database and are not shared?
- Which objects are local to a given database and are shared with other databases?

- Which objects are not local to a given database, but are used by it?
- How often does each database require a transfer?
- Does the DCM cluster require central control of objects?
- Will one person manage the DCM cluster (for example, a DCM administrator)?

The contents of transfer packages are not addressed by the DCM methodology. Transfer packages are part of the *implementation* of the methodology and are controlled by each source database.

Note Within a given DCM cluster, different software components may use different methodologies. For example, an application that is being developed across multiple sites might use the Master and Satellite methodology. In the same cluster, however, a software feature that is shipped as a tested released component might be transferred using the Publish and Subscribe methodology.

Master and Satellite methodology

In the Master and Satellite methodology, one DCM database is designated as the *master*. All other databases in the cluster are called *satellites*. The master performs master builds that combine changes made in the satellites.

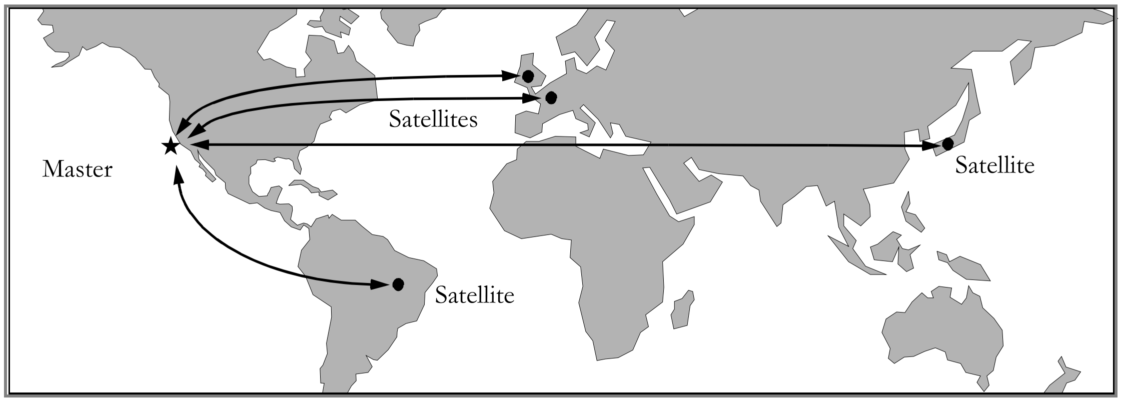
A Master and Satellite methodology is used when software components of an application are developed in different databases, but must be built, tested, and released together.

A Master and Satellite methodology is shown in the figure below. Typically, the satellites replicate directly with the master using point-to-point transfers. Satellites may replicate with each other as explained below:

- Directly using a Point-to-Point topology

  This gives the quickest propagation of changes from satellite to satellite.
- Indirectly through one or more hub databases

  The master database is often the hub around which the satellites form the spokes.



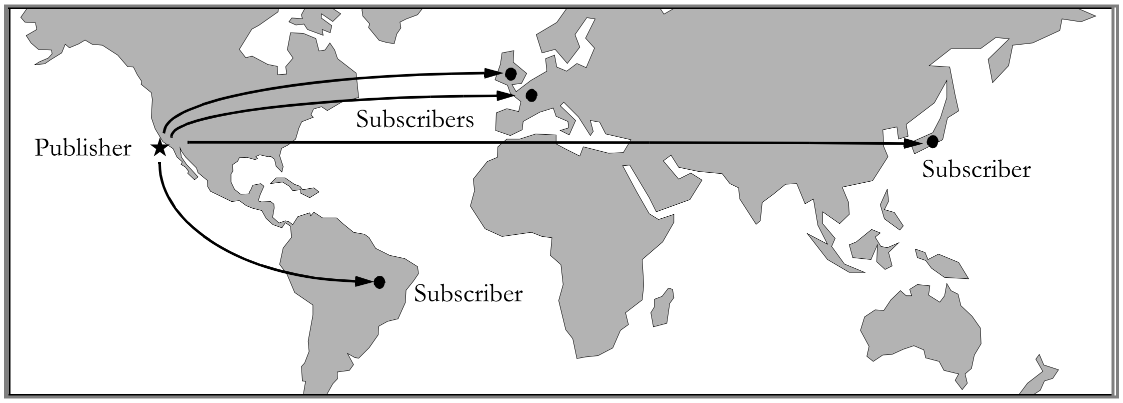
Publish and Subscribe methodology

In the Publish and Subscribe methodology, a central database is designated as the *publisher* – this database contains all shared objects. *Subscriber* databases request subsets of the data available from the publisher, and the publisher sends the data.

The transfer of data is one way – from the publisher to each subscriber. However, the publisher can collect shared data from other databases using DCM operations or the Rational Synergy migrate tool.

A Publish and Subscribe methodology is used when many databases must share software components (such as libraries, DLLs and header files), and the shared components are not developed in the subscriber databases.

A Publish and Subscribe methodology is shown in the figure below.

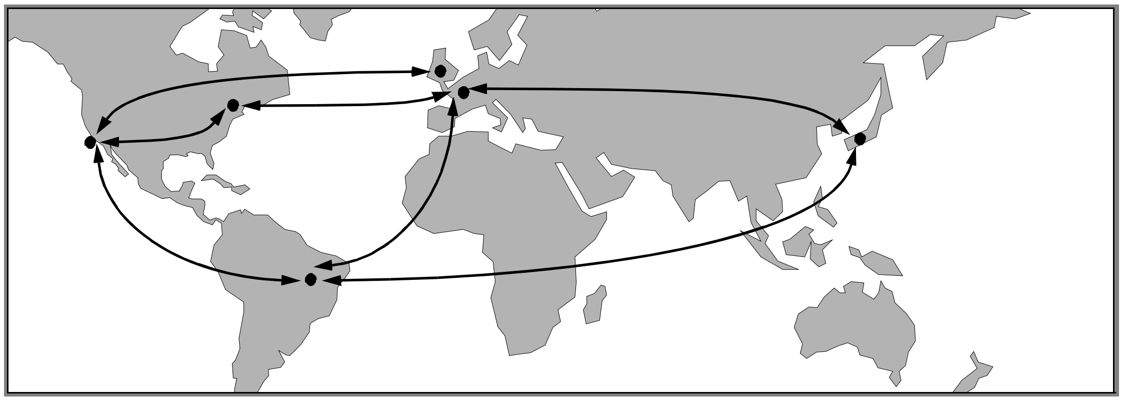


Peer-to-Peer methodology

In the Peer-to-Peer methodology, a master database is not designated. Each DCM database:
1) Controls the flow of information to each of its peers,
2) Maintains its own projects for integration testing, SQA testing, and releases,
3) Rebuilds its projects using objects that it receives from source databases, and
4) Sends new objects to destination databases that request them.

A Peer-to-Peer methodology is used when a flexible method of software distribution is needed and a central repository is either not feasible or not required.

A Peer-to-Peer methodology is shown in the figure below.



Establishing disk space requirements

Whenever data is sent and received, DCM writes to the file systems of the source and destination databases. Thus, these file systems must have plenty of free space before transfer packages can be generated, transferred, and received. The following subparagraphs explain how DCM uses source and destination file systems during the generate, transfer, and receive operations, and how to estimate the needed disk space.

Source database

The DCM generate operation writes temporary and transfer package files to the generate_directory. Following is a description of how this works: The generate operation creates a temporary export directory in the `generate_directory`. of the source database. The transfer set objects are temporarily stored in a compressed `tar` file in this export directory. To perform a successful generate operation, the source file system must have enough disk space for the export directory and the compressed `tar` file.

The generate operation also writes the preview of the transfer package and information files to the `generate_directory`; however, these files are too small to consider when estimating disk space requirements.

To estimate the disk space needed for a given generate operation, use the "DCM planning worksheet" on page 27.

Destination database

The DCM transfer and receive operations write transfer package files to the receive_directory. Following is a description of how this works: During the *generate* operation, a compressed `tar` file is written to the `generate_directory`. The transfer operation writes this compressed `tar` file to a temporary import directory in the `receive_directory`. of the destination database. If the package is to be zipped, the receive operation uncompresses and extracts the `tar` file, and deletes the compressed file. To perform a successful receive operation, the destination file system must have enough disk space for the import directory and compressed `tar` file.

The receive operation also writes the preview of the transfer package and information files to the `receive_directory`; however, these files are too small to consider when estimating disk space requirements.

To estimate the disk space needed for the transfer and receive operations, use the "DCM planning worksheet" on page 27.

Note DCM deletes temporary and transfer package files when it successfully completes a generate, transfer, or receive operation, but sometimes these files are not removed. Therefore, periodically check the *`generate_directory`* and *`receive_directory`* to make sure they do not contain leftover files.

DCM planning worksheet

Before sending and receiving data, use the following worksheet to estimate how much free disk space is needed in the `generate` and `receive` directories. As explained earlier, these directories are typically located at the source and destination databases, respectively.

DCM Disk Space Planning Worksheet
1. Source Objects
a. Estimate the average size of the source objects to be transferred. For example, if your project hierarchy contains several 100 KB source files and the remainder of the source files are 30 KB or smaller, you might choose 80 KB as a high (safe) estimate of the average source object size. average source size = ____________________ KB
b. Estimate the number of objects that will be in a regular transfer. For example, you might transfer 5 projects, each of which contains 200 source objects. If you expect an average of 150 members of each project to change between daily transfers, each transfer will contain 750 source objects. average number of sources = ____________________ objects **Note** An initial transfer usually contains many more objects than are transferred on a regular basis. For example, 5 projects with 200 files each will yield a minimum of 1,000 source objects in the initial transfer. If the objects are transferred with history, and each source file has an average of 6 versions, the transfer will contain 6,000 source files.

DCM Disk Space Planning Worksheet (Continued)
c. If objects are being transferred with history, at least two objects are transferred for each object that has changed. Therefore, multiply the average number of source objects by a history factor that is equal to 1 if you are transferring without history, or 2 if you are transferring with history. history factor = ____________________ (1 or 2)
d. Estimate the source size. source size = average source size (1.a) x average number of sources (1.b) x history factor (1.c) = ____________________ KB
e. Compute the total source size by adding 10% overhead for other attributes that are associated with the source objects. total source size = source size (1.d) x 1.10 = ____________________ KB
2. Product Objects
a. Estimate the average size of the product objects to be transferred. For example, if your project hierarchy contains several 500 KB product files and the remainder of the product files are 150 KB or smaller, you might choose 300 KB as a high (safe) estimate of the average product object size. average product size = ____________________ KB

DCM Disk Space Planning Worksheet (Continued)
b. Estimate the number of products that will be in a regular transfer. For example, you might transfer 5 projects, each of which contains 3 product objects. If you expect an average of 2 products of each project to change between daily transfers, each transfer will contain 10 product objects. average number of products = ____________________
c. If you are transferring objects with history, at least two objects are transferred for each object that has changed. Therefore, multiply the average number of products by a history factor that is equal to 1 if you are transferring without history, or 2 if you are transferring with history. history factor = ____________________ (1 or 2)
d. Estimate the product size. product size = average product size (2.a) x average number of products (2.b) x history factor (2.c) = ____________________ KB
e. Compute the total product size by adding 10% overhead for other attributes that are associated with the product objects. total product size = product size (2.d) x 1.10 = ____________________ KB

DCM Disk Space Planning Worksheet (Continued)

Note You may need to allow much more than 10% overhead for attributes associated with product objects (for example, if you generate Bills of Materials, as explained in *Help*).

3. Tasks

Multiply the number of tasks by 1 KB to estimate the space required for the average number of completed tasks in a transfer.

total task size = number of tasks x 1 KB

= ____________________ KB

4. Folders

If folders are being transferred, assume that a folder with 50 tasks requires 1 KB of disk space. Divide the number of tasks by 50 and multiply by 1 KB to estimate the space required for the average number of folders in a transfer.

total folder size = number of tasks / 50 x 1 KB

= ____________________ KB

Note Query-based folders increase in size over time. Consider this in your estimate.

5. Projects and Dependencies

a. Use the `ccm export` command to export each project object:

```
ccm export /t export_directory /h project_name-version:project:dbid#1
```

DCM Disk Space Planning Worksheet (Continued)

b. Use the command of your choice (such as, `dir` in Windows or `du -k` in UNIX) to obtain the size (in kilobytes) of each export directory.

export directory 1 size +
export directory 2 size +
⋮
export directory N size

____________________ KB

6. Total Size

Compute the total disk space required.

total size =
total source size (1.e) +
total product size (2.e) +
total task size (3) +
total folder size (4) +

____________________ KB

7. Total generate_directory Size

For the *`generate_directory`*, allocate at least 2.5 times the total size computed in Step 6.

total *`generate_directory`* size = total size (6) x 2.5
=
____________________ KB

If one transfer package is sent right after another (that is, if the transfers are queued), space must be allocated for each transfer package. Also, keep in mind that a failed transfer operation requires more disk space than a successful one.

DCM Disk Space Planning Worksheet (Continued)

8. Total receive_directory Size

For the *`receive_directory`*, allocate at least 2.0 times the total size computed in Step 6.

total *`receive_directory`* size = total size (6) x 2.0

= ____________________ KB

If one transfer package is received right after another (that is, if the receives are queued), disk space must be allocated for each transfer package. Also, keep in mind that a failed receive operation requires more disk space than a successful one.

Note If a database is used for both the generate and receive operations, space must be available in its *`generate_directory`* and *`receive_directory`*.

Establishing common database properties

By design, all databases in a DCM cluster must have: unique database IDs, identical DCM delimiters, and compatible models. These properties are described below.

- Unique database IDs

  Each DCM database must be manually given a unique database ID (`dbid`). The ID is set when the database is initialized to use DCM (see “Initializing a database” on page 39). The database ID is *case sensitive* and is a logical identifier for the database. It can be the same as the name of the database, but it does not have to be.

- Identical DCM delimiters

  The DCM delimiter is used in the names of all non-local objects in a DCM database. Each database in a DCM cluster must be given the same delimiter so that object naming is consistent. The delimiter is set when the database is initialized to use DCM. However, the delimiter can be changed, as explained in *Help*. For more information on DCM delimiters, see “Object naming conventions” on page 15.

- Compatible database case settings

  Case settings of the source and destination databases must be compatible as explained in Case settings.

- Identical release delimiters

  For consistent processing of release definitions and operations related to releases, the release delimiter must be identical across all databases in a DCM cluster.

- Compatible models

  The life cycles and types of all shared objects must be compatible across all databases in a DCM cluster. Also, to ensure compatibility during a receive operation, DCM automatically creates any user-defined type definitions that are not already defined in the destination database (see “Type definitions” on page 150).

Establishing common database parameters

Before using DCM, make sure your source and destination projects have the same update properties. Otherwise, transferred objects may not be correctly configured into their projects in the destination database(s).

The update properties do **not** have to be the same for all databases in a DCM cluster. However, each pair of source and destination databases must have the same values for the project purpose list, platform, release delimiter, release definitions, and task attribute value range settings. These properties are described below.

- Project purpose list

  The destination database must be able to recognize the project purpose of a transferred *prep* project. The possible values for this property are stored in the `project_purpose_list` attribute of a base model object. To modify the project purpose list, use the **Project Purpose Table** dialog or the `project purpose` command. This property is used when a *prep* project is created, checked out, or updated. See *Help* for more information about the project purpose list.

- Platform

  The destination database uses the platform setting of transferred objects to update projects. The possible values for this property are stored in the `etc\om_hosts.cfg` file in the Rational Synergy installation area.

  The platform setting is used by task-based CM. For more information, refer to the IBM Rational Synergy Administration Guide for UNIX or the IBM Rational Synergy Administration Guide for Windows.

- Release delimiter

  The release delimiter is used to construct release values from the component name and component release part of a release definition. The release delimiter must be identical in all databases in a DCM cluster for a release definition to generate the same release value or for a release value to reference its corresponding release definition.

- Release definitions

  The destination database uses the release setting of transferred objects to update the membership of projects. Therefore, the release definitions that correspond to the values that are being used with the transferred objects must be equivalently defined. This is most easily achieved by allowing the release definitions to be replicated, and by including release definitions in the transfer set(s) used to replicate. Each transfer set normally includes eligible release definitions by default. For more information, see "Replication of generic and release-specific processes" on page 118.

- Task attribute value range settings

  The task attribute value range settings define the possible values for task attributes. When using task-based CM, the destination database must be able to recognize at least some attribute values of the tasks in order to update the projects of the destination database.

By default, only the `release` attribute of a task is used in a task-based update operation. The `platform` and `task_subsys` attributes can also be used. In the destination database, folder queries can be set up based on the values of these attributes. To accomplish this, the attribute value range settings for the destination database must include the values of the `release`, `platform`, and `task_subsys` attributes, as applicable.

The task attribute ranges are defined in the *`database_path`*`\pt\attrange.dft` file. If the `release`, `platform`, or `task_subsys` attributes are defined in terms of other attributes, those attributes must be included in the `attrange.dft` file for the destination database.

At a minimum, the `release` values that are used by transferred objects must be defined in the destination database. Moreover, if the queries being used by the transferred objects contain references to `platform` or `task_subsys` values, these values must also be defined in the destination database.

Note Rational Change has its own administration interface that lets you set attribute range values.

Synchronizing engines and servers

The times on the Rational Synergy engine and server machines must be synchronized. By default, DCM allows machines accessing the same Rational Synergy database to have time discrepancies of up to 60 seconds. If a machine has a time difference that is greater than 60 seconds with another machine, the object may not be included in DCM transfer packages. The following scenario shows how this would occur.

1. A DCM generate is performed on a server whose time is accurate.
2. A few seconds later, a user checks in a file using a Rational Synergy session whose engine process is running on a machine that is 100 seconds behind the actual time.
3. A few seconds later, another DCM generate is performed on the machine that has the accurate time. The object will not be included in the first DCM transfer package since it was not checked in.

   However, it will also not be included in the second DCM transfer package because it is older than the last generate time.

When a DCM generate is performed, DCM subtracts a small time value from the current time (by default, 60 seconds) to use as the generate time. This allows for a time difference of 60 seconds between machines accessing the same database. This may mean that an object that is modified or becomes a direct or indirect member of a transfer set within 60 seconds of the start of a DCM generate may be included in that DCM transfer package and in the next one.

The synchronization process varies from one installation to another. For specific information about this operation, contact your system administrator.

4 Setting Up a DCM Database

Introduction

Before objects can be transferred between databases, the following steps must be performed: 1) Initialize both databases to use DCM, 2) Determine which databases are going to send transfer packages and which are going to receive them, and 3) Define the transfer sets that each database will use when sending data to other databases. This section explains how to perform these tasks and includes the following topics:

Methodology

In order for a database to be usable by DCM, a *DCM initialization* must be performed on the database. This is accomplished using the **Initialize DCM** dialog or the `ccm dcm -init` command.

Note Before you initialize a database, review the information in "Advanced DCM Topics" on page 115. Based on how you are using DCM, that section will help you decide if you need to change the default settings of any DCM parameters.

If a database will be used only as a destination, the setup is complete once the database is initialized and any DCM settings that you want to change have been modified. However, if a database will be used as a source (or a source **and** destination), the following steps must be performed: 1) Define the DCM database definitions, 2) Define the transfer sets, and 3) Add objects to the transfer sets. These actions are described below.

- Defining DCM database definitions

  Transfer sets can be sent only to databases that have been defined as potential destinations. These databases can be defined any time before starting a transfer.

- Defining the transfer sets

  A transfer set defines which objects are to be sent from a source database. A separate transfer set can be defined for each logical group of objects that need to be transferred.

- Adding objects to the transfer sets

  Once transfer sets are defined, objects are added to them.

Initializing a database

Before you can perform DCM operations on a database, you must initialize the database to use DCM. This procedure: 1) Enables all DCM operations, 2) Adds DCM options to Synergy Classic menus, and 3) Enables the versioning with which DCM ensures unique object names within a DCM cluster.

Before you start

Before initializing a database to use DCM, do the following:

1. Read the earlier Note.
2. Make sure that your session is the only one running on the database.
3. Protect the database (see the `ccm monitor` and `ccmdb protect` commands in the IBM Rational Synergy Administration Guide for UNIX or the IBM Rational Synergy Administration Guide for Windows).,
4. Have a valid backup of the database.

The initialize operation

This operation can be performed from the GUI or command line. When performing this operation, you must give each database in a DCM cluster a unique database ID, and a DCM delimiter that is the same in all the databases. Enter a description, a location and some details of the administrator(s) of this database. This information can be replicated across the cluster and assist others in determining what the database is used for, where it is located and who to contact for any administration issues - see "Replication of DCM database definitions" on page 123.

When you initialize the database, each object in the database is automatically assigned a cluster ID. The purpose of this attribute is to track name changes across databases, as explained in "Cluster IDs" on page 149.

This operation is described in *Synergy Classic Help*, specifically in Initialize a database for DCM.

Reviewing and selecting a process

In a non-DCM-initialized database, by default, Rational Synergy uses a default process named **Standard**. This process includes built-in generic process rules that do not treat tasks from other databases any differently from those completed locally. For example, for Integration Testing, all completed tasks are collected, and this includes both local and non-local tasks.

When a database is DCM initialized, Rational Synergy also creates an additional process named **Distributed**. This process differs from the **Standard** process in that it supports a two-tier form of development and testing:

- **Local Collaborative Development** is used for collaborative development that includes locally completed changes from other developers but not completed changes from other databases.
- **Local Integration Testing** is used to test that a component builds and is viable using locally completed changes. This does not include changes from other databases.
- **Master Integration Testing** is used to test that a component builds and is viable using changes from all databases.

The advantage of this two-tier approach is that changes that break the build only affect that local build and do not directly impact developers on remote sites. This can be a more controllable process especially when there is a significant time zone difference between sites. The disadvantage is that developers may have to wait until a **Master Integration Baseline** has been created before their projects can be updated with non-local changes.

In a DCM-initialized database, you can use the default **Standard** or **Distributed** process, modify either process, or create a custom process of your own design.

Existing release definitions and process rules are not modified during the DCM initialization. If the database that was DCM initialized has already been used for development, then you should review the process used for current releases to see whether the current process is still appropriate for those releases. You may wish to modify the process rules used for the release and/or copy an alternative generic process rule to a release specific process rule to implement a revised process for that release.

If the database that you have DCM initialized has not yet been set up with release definitions, then you should review the two default processes. If neither are appropriate for a release, you can modify either one or create a new custom process, When you create a release from the Rational Synergy (not Synergy Classic), you can select which process should be used.

Creating a DCM database definition

Before you send a transfer package to a given database, you must create a DCM database definition for it. DCM uses this information whenever a transfer package is sent from the source to the destination database. This operation is performed using the **Create DCM database Definition** dialog.

If the database is to be used directly as a destination database, the **Generate Allowed** option should be selected (the default setting). If data from the source database is replicated to this database indirectly via one or more hub databases and you wish to handover control of objects from the source database to the destination database, then you should select the **Handover allowed** option.

If the destination database is at a prior release of Rational Synergy, please see “Supported transfers from Rational Synergy Distributed releases prior to 7.1” on page 157.

This operation is described in *Synergy Classic Help*, specifically in Create a DCM database Definition.

Creating a transfer set

A transfer set is a related group of objects that can be sent to a destination database. A transfer set has properties such as: the name of the transfer set, where email notification is sent when the transfer occurs, problem scope, the generate_directory, which object types are excluded from the transfer, and whether additional information is replicated or included (see "Replication of generic and release-specific processes" on page 118, "Replication of DCM database definitions" on page 123, "Fully-expanding reconfigure properties" on page 125 and "Transfer of associated baselines" on page 127). This operation is performed using the **Create transfer set** dialog.

When you create a transfer set, you define only its properties (that is, the transfer set name, any excluded objects, and so on). Later, you add objects to the transfer set using the **Show DCM Properties** dialog for each object (see "Adding objects to a transfer set" on page 43).

This operation is described in *Synergy Classic Help*, specifically in Create a transfer set.

Adding objects to a transfer set

When you create a transfer set, you define its properties only (that is, the transfer set name, any excluded objects, and so on). Later, you add objects to the transfer set using the **Show DCM Properties** dialog for each object or the command line.

When you add a project, task, or folder to a transfer set, DCM automatically adds all members of that object. You have the option to fully expand update (reconfigure) properties during the Generate operation, as explained in "Creating a transfer set" on page 42.

The following table shows what is included when each type of object is added to a transfer set, and the **Fully Expand Reconfigure Properties** option is not selected. To learn what objects are added when this option is selected, see "Fully-expanding reconfigure properties" on page 125.

Object type	Objects added to the transfer set	History
folder	The folder itself, its member tasks, and all objects associated with the tasks.	NA
task	The task itself and all objects associated with the task. If the task is a component task, this includes associated projects or products. If the Include associated baselines option is selected, any baselines that include the task will also be included. For more information, see "Transfer of associated baselines" on page 127	NA

Object type	Objects added to the transfer set	History
project	The associated task of a project excluding any automatic tasks. If there is a component task or user-created task associated with this project, that task is included.. For non-static projects such as *prep* projects, the following are also added: 1) Any project baseline that the project is using, 2) All folders and tasks in the update properties of the project, and 3) All tasks in the folders. If the Include associated baselines option is selected, any baselines that include the project are also included. The associated objects of tasks are **not** included unless they are members of the project hierarchy. Subprojects of any project baseline are **not** included unless these subprojects are used as a baseline in the project hierarchy.	optional
project grouping	The project grouping itself, its member projects, and all tasks and removed tasks of the project grouping. The projects and tasks in a project grouping are expanded as described above. Only project groupings for *prep* projects are eligible for replication. Project groupings for projects in other non-static states are not replicated since those projects are ineligible.	NA
baseline	The baseline object itself, any component tasks that were created from the baseline, the project members, and the task members of the baseline. The component tasks, project members and task members of the baseline are further expanded as described above.	NA
process rule	The process rule itself, and the folder and folder template members of the process rule.	NA
process	The process definition itself, and the generic process rules that are used by the process. Each generic process rule is expanded as described above.	NA

Object type	**Objects added to the transfer set**	**History**
product	The object itself and any non-automatic task associated with the product. This includes any component task associated with the product.	NA
source	The object itself and its associated task(s).	optional

Objects that you explicitly add to a transfer set using a DCM add operation are called *direct members*. Objects that DCM automatically adds, because they are associated with a direct member, are called *indirect members*.

To transfer all eligible objects in a DCM database, use the built-in `Entire database` transfer set.

Note Even though an object is added to a transfer set, it may be excluded from the transfer. To see which objects are excluded from a transfer set when a transfer package is generated, refer to "Structure of a transfer package" on page 128.

This operation is described in *Synergy Classic Help*, specifically in Add an Object to a transfer set.

5 Using DCM

Introduction

Before DCM can be used, the steps in the preceding chapter must be performed. Once those tasks are complete, data can be transferred and received, as explained in the following topics:

- "Methodology" on page 48
- "About the generate operation" on page 49
- "Generating the transfer package" on page 53
- "About the receive operation" on page 55
- "Receiving a transfer package manually" on page 57

Methodology

DCM lets you send a shared data from a database to other databases and receive shared data from other databases. The procedure for sending and receiving data is as follows:

1. Generate the transfer package

   Whenever you need to include new objects in a transfer set, you must generate a new transfer package. To send these same objects to another database, you must generate a new transfer package using the same transfer set, and the database ID of the other database.

2. Send the transfer package

   DCM automatically sends the transfer package to the destination database unless you select the Manual Copy transfer mode.

3. Receive the transfer package (optional)

   To receive data manually, you must initiate a receive operation from the destination database.

   To perform an automatic receive, the following are needed: your chosen transfer mode must allow DCM to execute a remote command at the destination machine, and the source and destination machines must be correctly set up (as explained in "How to use automatic receive" on page 142).

About the generate operation

When you generate the transfer package, DCM automatically performs the following tasks:

- Computes and creates the transfer package
- Sends the transfer package to the destination database (optional)
- Receives the transfer package at the destination database (optional)

If you select email notification when you create a transfer set, DCM automatically sends the addressee information about the generate, transfer, and receive results when these operations are finished. This data is also available in the DCM Event Log.

Parts of a transfer package

A DCM transfer package may contain up to five parts:

- A data part, containing information about user objects such as files, tasks, and projects.
- An optional releases part, containing information about release definitions.
- An optional templates part, containing information about folder templates.
- An optional process part, containing information about processes and process rules.
- An optional types part, containing information about type definitions.

How DCM computes and creates a transfer package

DCM creates a transfer package using:

- The time the transfer package was last generated
- The objects contained in the transfer set
- The destination database definition.

By default, DCM saves the 30 most recent times at which the transfer package was generated (for more information, see "How DCM stores generate times" on page 152). The modification time of each object version in the transfer set is compared with the last generated time. Each object that is newer than the last generated time is then added to the transfer list. The time that the transfer package was last generated is shown in the **Last Generated** field of the **Generate** dialog.

After DCM compares the modification times, it removes from the transfer list any objects that are excluded or ineligible. This is accomplished using built-in exclusion rules of DCM. The **Exclude** dialog can also be used to further restrict which objects are transferred. The final transfer list is shown in the **Preview Generate** dialog. All of these objects are included in the data part of the DCM transfer package.

The following objects are excluded from the transfer list when DCM applies its built-in exclusion rules:

- Projects that are in neither a non-modifiable nor *prep* state
- Automatic tasks
- Project groupings other than for *prep* projects
- Admin objects, transfer sets, and models
- Modifiable objects, except as noted above
- Change request objects if Rational Change Distributed is not licensed
- Built-in predefined folder templates
- Built-in predefined generic process rules

There are no restrictions on the transfer of folders.

You can also exclude objects from the transfer list using one or more of the following options in the **transfer set Exclude** dialog:

- Exclude all products
- Exclude all imported objects (objects that were created in other databases)
- Exclude all objects of a specified type

The objects in the final transfer list are sent to the destination database. For details on the contents of a transfer package, see “Structure of a transfer package” on page 128.

You can include or exclude objects from the releases, templates or process parts of a transfer package using one or more of the following options:

- Exclude Type Definitions - an option in the **transfer set Exclude** dialog.
- Exclude database Info - an option in the **transfer set Exclude** dialog.
- Release Scope, Release Query - options in the **Create transfer set** and **Edit transfer set** dialogs.
- Replicate Reconfigure and Folder Templates - an option in the **Create transfer set** and **Edit transfer set** dialogs.

Sending a transfer package with DCM

The generate operation automatically sends the transfer package to the destination database unless one of the following transfer modes is used: Manual Copy, or a user-defined transfer mode that requires a manual copy.

After the transfer package successfully arrives at the destination database, DCM removes the transfer package from the source database (except when the Direct transfer mode is used). If email notification was selected when the destination database was defined, DCM also sends transfer status email.

Receiving a transfer package with DCM

You can manually receive a transfer package into a destination database (see “Receiving a transfer package manually” on page 57), or the source database can automatically initiate the receive (see “How to use automatic receive” on page 142). The type of receive operation (either Manual or Automatic) is set up when you create or edit a DCM database definition.

DCM performs automatic receives as user *ccm_root*. This is because the Rational Synergy engine process must perform the receive, and the engine process runs as *ccm_root*. Also, the *ccm_root* user has the *ccm_admin* role, which is required to change non-modifiable objects.

A manual DCM receive requires that the user has the *ccm_admin* role available. Auto-receive is optionally performed as part of a DCM Generate and is run by the engine process performing remote execution on a specified host. On Unix servers, this is done by the *ccm_root* user. On Windows servers, this is done by the CM administrator user. In each case, this user always has access to *ccm_admin* role. Such remote execution requires trusted hosts or logins and in secure environments an organization's security policy may forbid such trusts from being established.

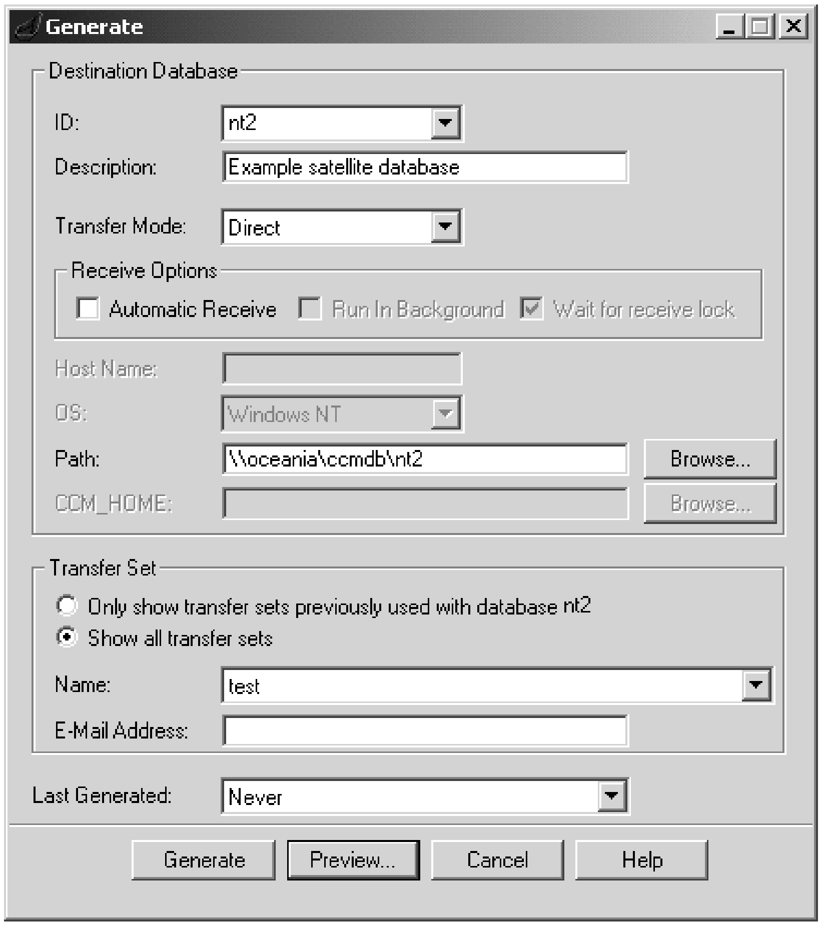
If Automatic Receive is enabled, DCM loads the data into the destination database as part of the generate operation. DCM performs the automatic receive by sending a command that starts a Rational Synergy session on the destination machine, loads the data, and stops the Rational Synergy session. For more information on how DCM checks and loads the data, see “About the receive operation” on page 55.

If you select email notification when you create a transfer set, DCM automatically sends the addressee receive status information when a receive operation is finished on that transfer set. The information is also available in the DCM Event Log in the receiving database.

If an automatic receive fails, error information may be found in the `*temp_dir*\dcm_ui.log` and `*temp_dir*\dcm_eng.log` files on the destination machine. On Windows, *`temp_dir`* is `%CCM_HOME%\temp`. On UNIX, *`temp_dir`* is the temporary directory used by the operating system of the machine on which the receive operation is running.

Generating the transfer package

You can generate a transfer package from the GUI or command line. The **Generate** dialog is shown in the figure below.


Generate

Destination Database

ID: nt2

Description: Example satellite database

Transfer Mode: Direct

Receive Options

Automatic Receive [ ] Run In Background [ ] Wait for receive lock [✓]

Host Name:

OS: Windows NT

Path: \\oceania\ccmdb\nt2

CCM_HOME:

Transfer Set

Only show transfer sets previously used with database nt2 [● Show all transfer sets]

Name: test

E-Mail Address:

Last Generated: Never

Before starting this operation, do the following:

- Confirm that the source database is initialized to use DCM.
- Make sure that the source database has a transfer set and a defined destination database (see "Creating a transfer set" on page 42 and "Creating a DCM database definition" on page 41).

- Ensure that the destination database is configured to accept the transfer package (for example, the destination database security and hosts files must be set up so that DCM can perform the transfer as user *ccm_root)*.

This operation is described in *Synergy Classic Help*, specifically in Generate and Send a transfer package.

About the receive operation

You can manually receive a transfer package into a destination database (see "Receiving a transfer package manually" on page 57), or the source database can automatically initiate the receive (see "How to use automatic receive" on page 142). The type of receive operation (either Manual or Automatic) is set up when you create or edit a DCM database definition.

Note Only one receive operation can be run on a specific database at a given time (that is, DCM does not allow multiple concurrent receives to be performed on a single database).

This section describes a manual receive operation. However, data is loaded and checked the same way whether the transfer package is received manually or automatically.

When a transfer package is received, DCM automatically performs the following tasks:

- Loads any type definitions that are included in the transfer package and are **not** defined in the destination database
- Loads the transferred objects
- Checks for parallel versions of the transferred objects
- Compares the source and the release information of the destination database

If you select email notification when you create a transfer set, DCM automatically sends the addressee the following when the receive operation is finished:
1) Information on the data loading, 2) Parallel version checking results, and
3) Additional reports for skipped objects and conflicts. This information is also included in the DCM event log.

Loading type definitions

If type definitions are **not excluded** from a transfer set, then all type definitions in the transfer set that are not defined in the destination database and which are required for objects in the data part of the transfer package are imported. If a type definition that is in the transfer set already exists in the destination database, it is not modified. For more details, see "Type definitions" on page 150.

Loading release definitions

If release definitions are **not excluded** from a transfer set, then release definitions that are included in the releases part of a transfer package are imported. For more details, see “Replication of generic and release-specific processes” on page 118.

Loading processes, process rules and folder templates

If update (reconfigure) and folder templates are **not excluded** from a transfer set, then processes, process rules and folder templates that are included in the process and templates parts of a transfer package are imported. For more details, see “Replication of generic and release-specific processes” on page 118.

Loading the transferred data

DCM loads each transferred object into the destination database by "cloning" it. If the object is new to the destination database, DCM creates a new object whose properties are identical to those of the transferred object. If the object already exists in the destination database, DCM updates all object properties so that they are identical to those of the transferred version. These properties include any relationships to other objects, such as history relationships.

Note Properties of non-modifiable objects that are in the destination database may change when new objects are loaded. However, the source contents of non-modifiable objects that already exist are not updated.

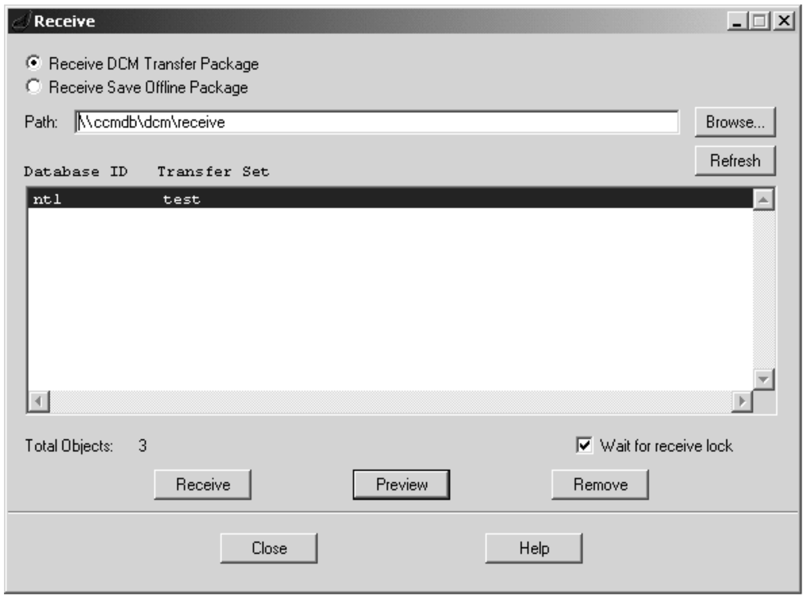
Checking for parallel versions

During the receive operation, DCM checks each newly-created object to see if there is an object in the destination database that has the same predecessor. This is only performed when the predefined **Entire database** transfer set is used to generate the package, or if the object is a history member of the transfer set used to generate the package. An object version in the destination database can have the same predecessor as a received object version if: 1) A parallel version was created in the destination database, or 2) Parallel versions are received from two or more databases.

If Local Parallel Notification is specified in a transfer set, then, for each received object, DCM makes a list of parallel versions that were created in the destination database. Each user who owns or created such a version receives an email. This notice describes the received object and its parallel versions, one of which is owned by the email recipient. For more information, see “Parallel checking” on page 151.

Receiving a transfer package manually

Once you initialize a database to use DCM, the database can receive a transfer package. You can perform a manual receive operation from the GUI or command line. The **Receive** dialog is shown in the figure below.



[✓ Receive DCM Transfer Package]

Path: \\ccmdb\dcm\receive

```
Database ID   Transfer Set
ntl           test
```

Total Objects: 3

[✓ Wait for receive lock]

This operation is described in *Synergy Classic Help*, specifically in Receive a transfer package.

6 Administering DCM Databases

Introduction

DCM administrators can oversee one or more databases in a cluster. These persons are typically responsible for monitoring and analyzing data transfers, communicating database changes to other administrators, and creating and updating one or more databases in the cluster.

The DCM administrator performs a variety of operations. For instance, to change the directory in which `ftp` transfers are received, the administrator must update the destination database definitions for all source databases that perform `ftp` transfers to that directory. In addition, if a new site wants to start with a copy of an existing DCM database, the administrator must duplicate the database and change its ID.

To help administrators perform these and other tasks, the following topics are included in this section:

- “Communicating with other DCM administrators” on page 60
- “Changing DCM settings after an upgrade” on page 61
- “Changing a database ID” on page 62
- “Changing the DCM delimiter” on page 65
- “Editing a DCM database definition” on page 66
- “Deleting a DCM database definition” on page 67
- “Changing transfer set parameters” on page 68
- “Deleting a transfer set” on page 69
- “Adding an object to a transfer set” on page 70
- “Removing an object from a transfer set” on page 71
- “Duplicating a DCM database” on page 72
- “Transferring all DCM members of a database” on page 76
- “Creating and maintaining "mirrored" databases” on page 77
- “Maintaining DCM directories” on page 78

Communicating with other DCM administrators

DCM administrators must inform each other of any issues that affect shared data. For example, if the update properties of a *prep* project change, the build manager must notify all build managers in the DCM cluster. Following are some of the changes that may need to be conveyed to all build managers in the DCM cluster who use this *prep* project or a similar local *prep* project:

- Addition of a new type in one of the databases (if type definitions are excluded from transfer packages)
- Changes to release or platform values
- Addition of a folder to the update properties of a project (if not using process rules or process rules are excluded from transfer packages)
- Changes to a project baseline (if not using process rules or process rules are excluded from transfer packages)
- Change of a process rule or folder template (if templates are excluded from transfer packages)
- Changes to folder query criteria if folder templates are not used

Changing DCM settings after an upgrade

Once you upgrade a database to a more current release, you must change the *`CCM_HOME`* field in the definitions in all databases that send data to the upgraded database. You must also review the settings of DCM database definitions in the upgraded database and ensure they are compatible with the Rational Synergy release used with those databases.

For complete instructions, refer to the appropriate *Rational Synergy Upgrade Instructions* located on the Rational Synergy Information Center Web site (`http://publib.boulder.ibm.com/infocenter/rsdp/v1r0m0/index.jsp/`. These *Upgrade Instructions* contain information specific to DCM functionality in the section *"Upgrading a DCM Cluster."* Further information may also be found in "Supported transfers from Rational Synergy Distributed releases prior to 7.1" on page 157.

Changing a database ID

A database ID can be changed after the database has been initialized to use DCM. This change can optionally be applied to objects that were created in the old database ID to convert them to the new database ID.

Changing a database ID without updating objects

This feature is useful when a database is added to a DCM cluster by copying one of the existing databases in the cluster (see “Duplicating a DCM database” on page 72). When the this option is used, all of the existing objects retain their former name and other properties. New objects created after the change will use the new database ID and therefore appear to have been created in a different database.

It is vital that the new database ID is unique in the DCM cluster. You should verify this before proceeding and before creating any new objects.

Note that changing a database ID may have an impact on the work area paths for projects in that database - see “Work area subdirectory templates and paths” on page 155 for details.

This operation is described in *Synergy Classic Help*, specifically in Change a DCM database ID.

Changing a database ID and updating affected objects

This feature is useful when the database ID that has previously been used needs to be changed, and all the objects that were created in that database need to be updated to reflect the new database ID.

Caution If a database is a member of an existing DCM cluster, changing its database ID may adversely affect other databases in the cluster.

In databases in which many objects have been created, there could be many objects that require updating, so this operation could take some time. It is vital that the new database ID is unique in the DCM cluster. You should verify this and that you ensure you have a valid backup of the database before proceeding.

Note that changing a database ID may have an impact on the work area paths for projects in that database - see “Work area subdirectory templates and paths” on page 155 for details.

All DCM administrators of a cluster **must be informed** when the ID of one of the databases in the cluster is to be changed. DCM does **not** automatically notify DCM administrators of the change.

The steps to perform a change of database ID and update objects are:

- Notify all DCM administrators of the intended change of database ID and co-ordinate a good time to perform the change.
- Suspend DCM operations in the database whose ID is being changed and all other databases that have received objects from that database.
- In the database to be changed, use the **Change database ID and update affected objects** option.
- In all other databases in the DCM cluster that may have received objects from the database whose ID has changed, use the **Convert the database ID of objects created in another database** option. You may need to co-ordinate with remote administrators who will perform this operation on the databases at their location.
- In all other databases, change the database ID in the DCM database definition for that database. You may need to co-ordinate with remote administrators who will perform this operation on the databases at their location.
- When you have received confirmation from all DCM administrators that all databases requiring change have been successfully updated, notify all administrators that the change is complete and that they may resume DCM operations.
- Resume DCM operations from the database whose ID has been changed.

Note that this operation preserves the modify time of objects in the database so that subsequent DCM replications should not be adversely affected.

This operation is described in *Synergy Classic Help*, specifically in Change a DCM database ID.

Converting the database ID of objects received from another database

This option is used after another database ID has been changed and converts any objects referencing the old database ID to reflect the new database ID. See “Changing a database ID and updating affected objects” on page 62 for the complete set of steps for changing a database ID. This operation preserves the modify time of objects in the database so that subsequent DCM replication should not be adversely affected.

Note that changing a database ID may have an impact on the work area paths for projects in that database - see “Work area subdirectory templates and paths” on page 155 for details.

This operation is described in *Synergy Classic Help*, specifically in Change a DCM database ID.

Changing the DCM delimiter

The DCM delimiter can be changed after the database has been initialized to use DCM. This change **must be applied to all databases in the DCM cluster.** Since this may require that many objects are updated, ideally it should be performed early when the number of databases and objects requiring update in the cluster is small. If you have a large cluster or large databases, this operation could take some time and may require careful planning. You should verify that you have a current and usable backup of each database in the cluster before proceeding.

The steps to perform a change of DCM delimiter are:

- Notify all DCM administrators of the intended change of DCM delimiter and co-ordinate a good time to perform the change.
- Suspend all DCM operations in the cluster.
- In each database in the cluster, use the **Change DCM Delimiter** option. You may need to co-ordinate with remote administrators who will perform this operation on the databases at their location.
- When you have received confirmation from all DCM administrators that all databases have been successfully updated, notify all administrators that the change is complete and that they may resume DCM operations.
- Resume any DCM operations local to your site.

Note that changing a database ID may have an impact on the work area paths for projects in that database - see “Work area subdirectory templates and paths” on page 155 for details.

Note that the change DCM delimiter operation preserves the modify time of objects so that subsequent DCM replication should not be adversely affected.

Editing a DCM database definition

Your access to a destination database is controlled by the DCM database definition in the source database. If the destination database properties change, you may need to edit its DCM database definition in the source database.

For example, suppose you can now perform a remote copy to a database to which you have performed only manual transfers in the past. By editing its destination database definition, you can change its transfer mode from Manual Copy to Remote Copy (see “Transfer modes” on page 131).

As a second example, suppose a database is moved to a different machine or its host name is changed. In such cases, it is necessary to edit the destination database definition in all databases that send data to it.

You can change the following properties in a DCM database definition:

- Database ID
- Database description, location, admin contact info
- Database ID hidden or visible settings
- Database can be used as a destination database setting
- Handover allowed
- Receive options
- Transfer mode (and its related parameters)
- Zip setting

Caution Changes to an existing database definition are typically the result of a change notification from the destination database administrator. To ensure data integrity, inform this person of any changes made to the destination database definition.

This operation is described in *Synergy Classic Help*, specifically in Edit a DCM database definition.

Deleting a DCM database definition

When you no longer want to use a destination database definition, you can remove it. If the database exists but you wish to exclude its database ID from various choice lists, you may prefer to mark it as hidden - see "Editing a DCM database definition" on page 66.

This operation is described in *Synergy Classic Help*, specifically in Delete a DCM database definition.

Changing transfer set parameters

As your needs change, you may want to change transfer set parameters. For example, if you no longer need to include products in a given transfer set, you can change the transfer set so that products are excluded from future transfers.

You can change the following properties in transfer sets:

- Transfer set name
- Email address for email notification
- Email policy
- Problem scope
- Cumulative problem scope
- generate_directory
- Release scope and release query
- Local parallel notification
- Inclusion of associated baselines
- Excluded objects

This operation is described in *Synergy Classic Help*, specifically in Edit transfer set parameters.

Deleting a transfer set

When you no longer want to use a transfer set, you can delete it.

Caution Once you delete a transfer set, all of its members are disassociated from it, its Last Generated information is deleted, and you can no longer use it to generate or send transfer packages.

This operation is described in *Synergy Classic Help*, specifically in Delete a transfer set.

Adding an object to a transfer set

One way to change a transfer set is to add a new object to it. This operation is described in “Adding objects to a transfer set” on page 43.

Removing an object from a transfer set

You can remove an object from a transfer set only if it is a direct member. An indirect member cannot be explicitly removed from a transfer set. When a direct member such as a project, task, folder, or baseline is removed from a transfer set, the transfer set is marked as needing to be recomputed. The indirect members are not removed at that time. Later, when indirect members of the transfer set are recomputed, the indirect members are automatically removed – unless they are used elsewhere in the transfer set. Recomputing the indirect members of the transfer set may be explicitly performed, or will be automatically performed when the transfer set is next used for a DCM Generate.

Caution If you remove an object that was added with history, the entire history of that object is removed from the transfer set.

This operation is described in *Synergy Classic Help*, specifically in Remove an object from a transfer set.

Duplicating a DCM database

You can create a new DCM database by duplicating an existing DCM database. This is an easy way to add a database to a DCM cluster if the intent is to replicate using the predefined `Entire database` transfer set.

The two ways that you can duplicate a DCM database are:

- Copy the database using pack and unpack commands
- Transfer all DCM members of a database

Copying a DCM database using pack and unpack

You can use the Rational Synergy pack and unpack commands to duplicate an existing DCM database. The database that is to be packed must be initialized to use DCM. Also, there must be an active server and sufficient disk space where the database is to be unpacked.

Note that you cannot pack a database from a Windows server and unpack it on a UNIX server.

The pack operation includes the generate_directory and receive_directory. Therefore, all outstanding transfers or receives must be resolved before packing the database.

Caution When a Rational Synergy database is unpacked or copied, the project work area paths of new database are the same as those in the original database. To avoid possible conflicts, these paths must be changed before the new database is used.

For information on how to change work area paths, see the following in *Synergy Classic Help*: Set work area properties.

To create a new DCM database using the pack and unpack commands, do the following:

1. Make sure there are no active sessions running on the database that is to be packed.

2. Protect the database that is to be packed (see the `ccm monitor` and `ccmdb protect` commands in the IBM Rational Synergy Administration Guide for UNIX or the IBM Rational Synergy Administration Guide for Windows).

3. Pack the DCM database:

```
> ccmdb pack database_path /to packfile
```

For Windows, `database_path` must be a UNC path.

The resulting packed file is called `database_name.cpk`.

4. Unpack the DCM database, and give it a new name:

   ```
   > ccmdb unpack database_name.cpk /t
   new_database_path\new_database_name /s server_name
   ```

   For Windows, `new_database_path` must be a UNC path.

5. Change the work area paths.

6. Change the DCM database ID:

   a. Start a session on the new database as *ccm_admin*.

   b. Protect the new database.

   c. Bring up the **Change DCM database ID or Delimiter** dialog.

   d. Select the option **Change database ID without updating any objects**.

   e. Change the database ID to an ID that is unique in the DCM cluster.

   f. Click **OK**.

   g. Exit from the session.

      For more information, refer to "Changing a database ID" on page 62.

7. Set up the original database to send transfer packages to the new database:

   a. Unprotect the original database (see the `ccmdb unprotect` command in the IBM Rational Synergy Administration Guide for UNIX or the IBM Rational Synergy Administration Guide for Windows).

   b. Start a session on the original database.

   c. Protect the original database so that no one else can start a session on it.

   d. Create a destination database definition for the new database. For more information, refer to "Creating a DCM database definition" on page 41.

   e. Create and add objects to transfer sets for any transfer packages that will be sent from the original to the new database. For more information, refer to "Creating a transfer set" on page 42.

   f. Initialize the new transfer sets.

   g. Generate transfer packages for each new transfer set, with Last Generated Time set to `Current Time`. This prevents unnecessary transfers (that is, objects in the newly unpacked database will not be sent by the original database). Moreover, only objects that have changed since the original database was unprotected are eligible to be transferred to the new database.

   h. Exit from the session.

8. Unprotect the new database.
9. Set up the new database to send transfer packages to the original database (optional).
   a. Start a session on the new database.
   b. Protect the new database so that no one else can start a session.
   c. Create a destination database definition for the original database. For more information, refer to "Creating a DCM database definition" on page 41.
   d. Create and add objects to transfer sets for any transfer packages that will be sent from the new to the original database. For more information, refer to "Creating a transfer set" on page 42.
   e. Initialize the new transfer sets.
   f. Generate transfer packages for each new transfer set, with **Last Generated Time** set to **Current Time**. This prevents unnecessary transfers (that is, objects that were obtained from the original database will not be sent back to it). Moreover, only objects that have changed since the new database was unprotected are eligible to be transferred to the original database.
   g. Exit from the session.

10. Unprotect the new database.
11. Unprotect the original database.

For more information on server commands, refer to the IBM Rational Synergy Administration Guide for UNIX or the IBM Rational Synergy Administration Guide for Windows.

Transferring all DCM members of a database

You can use a built-in transfer set named **Entire database** to copy all eligible objects from one DCM database to another. The destination database does not need to be new nor empty. All eligible objects in a DCM database are members of the `Entire database` transfer set. Therefore, no more objects need to be added to it.

Note that the `Entire database` transfer set does **not** replicate all objects in the database. Objects in the working state and objects related to the model that are vital to the correct functioning of the database may be excluded. Hence you should not use this transfer set as an alternative to performing regular backups of the database. You cannot fully restore a database using DCM alone.

The **Entire database** transfer set is useful for small databases, when a small team developing an application or component needs to perform distributed development across multiple locations. However, for large databases and complex development, using this transfer set can result in a lot of data being replicated and in databases growing in size very rapidly since each new static object version is replicated to the other database(s). Also note that if the source database has been in use for some time, the first DCM transfer package may be very large and take a long time to generate and to receive.

Before starting this procedure:

- The source and destination databases must be initialized to use DCM.
- The destination database must be configured to accept the transfer package (for example, the destination database security and hosts files must be set up so that user *ccm_root* can perform the transfer).

To transfer all eligible objects from the source database to the destination database, do the following:

1. Create the destination database definition.

   This operation is described in *Synergy Classic Help*, specifically in Add a destination database definition.

2. Generate the transfer package.

   Select the destination database and the **Entire database** transfer set.

   This operation is described in *Synergy Classic Help*, specifically in Generate and send a transfer package.

3. Receive the transfer package.

   This operation is described in *Synergy Classic Help*, specifically in Receive a transfer set.

Creating and maintaining "mirrored" databases

To keep two DCM databases synchronized with the same data, you first create a "mirrored" database from another database. You then keep the databases synchronized by transferring their **Entire database** transfer sets to each other.

Before starting this procedure, do the following:

- Make sure that the database to be packed is initialized to use DCM.
- Resolve all outstanding transfers or receives before packing the database. You need to do this because the pack operation includes the `*database_path*\dcm\generate` and `*database_path*\dcm\receive` paths that are used as the default generate_directory and receive_directory, respectively.
- Verify that there is an active server and sufficient disk space where the database is to be unpacked.

To create mirrored databases, do the following:

1. Set up the databases as described in “Copying a DCM database using pack and unpack” on page 72, but replace two of the steps as explained below.
   a. In place of step 7.f, initialize the transfer set of the original database as follows:

      Perform a generate using the **Entire database** transfer set for the mirror database, with **Last Generated Time** set to `Current Time`. Thereafter, only objects that have changed since the original database was unprotected will be eligible for transfer to the mirror database using the **Entire database** transfer set.

   b. In place of step 9.e, initialize the transfer set of the mirror database as follows:

      Perform a generate using the **Entire database** transfer set for the original database, with **Last Generated Time** set to **Current Time**. Thereafter, only objects that have changed since the mirror database was unprotected will be eligible for transfer to the original database using the **Entire database** transfer set.
2. Periodically synchronize the databases by transferring their **Entire database** transfer sets to one another.

Maintaining DCM directories

DCM uses temporary files to send and receive transfer package information. As explained in “Source database” on page 25 and “Destination database” on page 25, these files are stored in the generate_directory and the receive_directory.

DCM creates the *generate_directory* the first time a generate operation is performed in a database. DCM creates the *receive_directory* during the DCM initialization of a database so it can receive transfers.

During normal operations, DCM uses these directories for temporary creation and extraction of the transfer package. When DCM completes an operation, it removes the temporary files. Occasionally the transfer package files may not be removed completely. Therefore, periodically check the *generate_directory* and *receive_directory* and remove any unneeded files.

Caution Do not delete the following directory nor its contents: *database_path*\dcm\receive\history. DCM uses this data to identify missing transfer sets, as explained in “Missing transfer sets” on page 145.

For details on the files that DCM stores in the *generate_directory* and *receive_directory*, see “Structure of a transfer package” on page 128.

7 Sample DCM Scenario: Master and Satellite

Introduction

This section describes a sample DCM scenario. Although your methodology may be different, this example provides a useful guideline. Use the information in this section as a checklist of things to consider when implementing DCM. The following topics are covered:

Prerequisites

To perform the steps in this scenario, you must meet the following requirements:

- Each machine receiving a transfer must be set up to permit remote command execution. For example, on UNIX the receiving machine must have an entry for the sending machine in its system `hosts.equiv` file or in the *ccm_root* `.rhosts` file, and on Windows the receiving machine must be running `ccm_remd`.
- All databases in your DCM cluster must have compatible models.
- You must have a working knowledge of task-based CM.
- You must have the *build_mgr*, *dcm_mgr*, and *ccm_admin* roles. For more information, see "Roles" on page 17.

Scenario of the DCM methodology

This scenario includes two independent databases: one is located in Boston, Massachusetts (MA), and the other is in Chicago, Illinois (IL). These two sites will concurrently develop version 2.0 of a software application that displays "Hello world". This will use a release value of `hello/2.0`. The first release of the application was developed and released by the Boston site. A released baseline for this was created using a release of `hello/1.0` and this has a member project of `hello-1.0`.

Each site will develop a part of the application and maintain its own integration *prep* project. The Master and Satellite methodology will be used. MA is the master and controls the master integration and released projects. IL is the satellite. For more information on this methodology, see "Master and Satellite methodology" on page 22.

The following assumptions are made:

- The sites use the same, or nearly the same, model.
- Both sites use the default DCM delimiter ("#").
- Both sites use the default release delimiter ("/").
- Both sites use task-based CM.
- Both sites use a `hello/2.0` release value using the **Distributed** process.
- Both sites use automatic receive, and security is set up at each site to support automatic receives.
- DCM sends email notification to both the Boston and Chicago DCM managers when it generates or receives a transfer package.
- The sites are connected by a wide area network that lets them transfer data using remote copy commands.

There are two DCM managers: one in Boston and one in Chicago. Each manager has responsibilities that involve both databases. The build manager at each site is also involved in this scenario.

Master responsibilities

The Boston build manager is responsible for:

- Establishing the master integration projects
- Building the master projects
- Establishing the local integration projects for Boston

- Building the local integration projects

The Boston DCM manager is responsible for:

- Initiating the first transfer
- Collecting data from the satellite databases
- Distributing changes back to the satellites

Satellite responsibilities

The Chicago build manager is responsible for:

- Establishing local integration projects for Chicago
- Building local integration projects for Chicago

The Chicago DCM manager is responsible for:

- Sending changes from Chicago back to the master

The following topics are covered in this sample scenario:

1. Initialize the databases.
2. Set up the master.
3. Perform the initial transfer to the satellite.
4. Set up the satellite.
5. Perform periodic satellite-to-master transfers.
6. Perform periodic master-to-satellite transfers.
7. Continue to develop software.
8. Add a satellite.

Initializing the databases

In this scenario, the databases have not yet been initialized to use DCM. To initialize the databases, do the following:

1. In the MA database, make sure the release value `hello/1.0` is defined and are allowed DCM transfer. Do not define these releases in the IL database.
2. In the MA database, ensure that a released baseline has been created for the `hello/1.0` release and includes the `hello-1.0` project.
3. Initialize the Boston database to `MA`.

   Use the **DCM Initialization** dialog to set the database ID to `MA` and the DCM delimiter to "#".
4. In the MA database, using Rational Synergy (not Synergy Classic), copy the `hello/1.0` release to the new `hello/2.0` release, selecting the **Distributed** process for the new release. Ensure that the `hello/2.0` release is allowed DCM transfer. If release `hello/2.0` was already defined before the database was DCM initialized, then using Rational Synergy (not Synergy Classic), show the properties of release `hello/2.0` and on the process rules tab press the add button to add the generic process rules from the **Distributed** process.
5. Initialize the Chicago database to `IL`.

   Use the **DCM Initialization** dialog to set the database ID to `IL` and the DCM delimiter to "#".

Setting up the master

The MA database is the master. Set up this database as follows:

Note The following procedure assumes that projects use process rules (update or reconfigure templates) by default. If process rules are not used, the appropriate folders need to be manually created and added.

1. Create the master integration project.

   A master integration project is needed in order to combine integration builds for the completed tasks of the two databases. Check out the master integration *prep* project from `hello-1.0`. Set the version to `MasterInt` and select `Master Integration` from the **Purpose** list in the **Check Out Project** dialog. Set the release value to `hello/2.0`. The new project is `hello-MasterInt`.

   The update properties for the new project are automatically set up using the process rules. For more information on process rules, see the Build Manager's Guide.

2. Create the MA local integration project.

   One local integration testing project is needed for the MA local integration builds. Check out the `MA` integration *prep* project from the `hello-1.0` project, setting the version to `MAInt` and selecting **Integration Testing** from the **Purpose** list in the **Check Out Project** dialog. Set the release value to `hello/2.0`. The new project is `hello-MAInt`.

   The update properties for the new project are automatically set up using the process rules.

3. Create the DCM database definition for IL.

   Use the **Create DCM database Definition** dialog to add the destination database ID (`IL`), its description, and its properties. Choose Remote Copy as the transfer mode, and select the **Automatic Receive** check box.

4. Create the master-to-satellite transfer set.

   a. Set the transfer set properties.

      Use the **Create Transfer Set** dialog to name the transfer set (`Master Transfer of hello Project`), and set the email address for generate and receive notification. Ensure that the `Release Scopes` is set to **Releases and Templates**. Check that the transfer set exclusions do not exclude DCM database definitions.

b. Choose **Admin > Browse Baselines to** open the **Browse Baselines** dialog. Click **DCM Properties**, then add the released baseline for the previous release, `hello/1.0` to the transfer set. This will automatically include the `hello-1.0` project and its members.

c. Add folders to the transfer set.

   Choose **Tools > Folder > Browse** `to open the` **Browse Folders** `dialog. Click` **DCM Properties**, then add the `All completed tasks for release hello/2.0` folder and the `Master Integration Tested Tasks for hello/2.0` folder to the transfer set. This will automatically include all completed changes from all databases and the master integration tested tasks for `hello/2.0`.

Note that is not necessary to add the master project to the transfer set. All of the tasks and their associated source changes will be replicated through the folders that are members of the transfer set.

Performing the initial transfer to the satellite

Before you can create an integration testing project in the satellite, you must send the master `hello` project and associated data to the satellite. This is the first transfer that you will perform to each satellite.

Note This action requires the *dcm_mgr* role. DCM performs dynamic role switching for this and all other DCM operations.

Generate, send, and receive the initial transfer package using the **Generate** dialog, as follows:

1. Choose the generate options.

   Select the `IL` destination database and the `Master Transfer of hello Project` transfer set.
2. Click the **Preview** button, and verify that the correct objects are in the transfer list (optional).
3. Generate the transfer package.

   This operation generates, transfers, and initiates the automatic receive of the transfer package.

When the generate and send operations are completed, IL automatically receives the transfer set. IL will then have a copy of the baseline, a copy of the `hello-1.0` project, and copies of the `All completed tasks for release hello/2.0` folder and the `All completed tasks for hello/2.0 from database MA` folder.

The IL database will also have the process rules for the `hello/2.0` release, and release definitions for `hello/1.0` and `hello/2.0` since these were included in the transfer package. You can check these using the **Browse Reconfigure Templates** and **Browse Releases** dialogs.

The IL database also sets up a skeleton DCM database definition for MA. You can check this using the **Browse DCM database Definitions** dialog.

Setting up the satellite

IL is a satellite in this scenario. It should be set up after you set up the master and perform the initial transfer to the satellite.

To set up the satellite, do the following:

1. Create the IL local integration *prep* project.

   One *prep* project is needed for IL local integration builds. Check out the IL integration *prep* project from the `hello-1.0` project. Set the version to `IL#Int`, and select `Integration Testing` from the **Purpose** list in the **Check Out Project** dialog. The new project is `hello-IL#Int:project:MA#1`.

2. Edit the satellite DCM database definition for MA.

   A DCM database definition for MA should already have been created by DCM receive since DCM database definitions were not excluded from the transfer package. Use the **Edit Destination database** dialog to set up the remaining properties. Select **Generate Allowed**, choose **Remote Copy** as the transfer mode, and select **Automatic Receive**.

3. Create the satellite-to-master transfer set (this transfer set contains the tasks that have been completed in the satellite database).

   Note This action requires the *dcm_mgr* role. DCM performs dynamic role switching for this and all other DCM operations.

   a. Set the transfer set properties.

      Use the **Create transfer set** dialog to name the transfer set (`IL Transfer of Completed Tasks for hello Project`), and set the email address for generate and receive notification.

   b. Exclude objects that are not local to the database.

      By excluding non-local objects, the transfer package will not include the objects that IL received from the master. In the **Create Transfer Set** dialog, choose **Exclude**. Select **Exclude Imported Objects**. Click **OK**.

   c. Add the folder for the satellite.

      From the Synergy Classic Project View **Tools** menu, choose **Folder > Browse** to bring up the **Browse Folders** dialog. Locate and select the `All completed tasks for Release hello/2.0 from database IL` folder, and click **DCM Properties**. Using the **Show DCM Properties** dialog, add the folder to the transfer set.

Initializing the satellite transfer set

You need to send to the master only those objects that have changed since the first transfer from the master to the satellite was received. That is, you do not need to send the "seed" objects that the master originally sent to the satellite (see “Performing the initial transfer to the satellite” on page 86). Therefore, select **Current Time** in the **Last Generated** list. The next time you generate a transfer set at the satellite, only those objects that have changed since the last generated time will be included in that transfer set.

To "initialize" the satellite transfer set, perform a generate operation from the **Generate** dialog:

1. Select the generate options.

   Choose `MA` as the destination database, and select the `IL Transfer of Completed Tasks for hello Project` transfer set.

2. Select the generate time for the transfer package.

   Set the **Last Generated** time to **Current Time**. This creates an empty transfer set and sets an initial last generate time — the next transfer package will be created relative to this time.

3. Exclude objects that are not local to the satellite database.

   Turn on **Exclude Imported Objects**.

4. Click **Preview**, and confirm that there are no objects in the transfer list (optional).

5. Generate the transfer package.

This operation produces an empty transfer package; no data is transferred to the master.

Performing periodic satellite-to-master transfers

After IL receives the initial transfer from the master, developers can check out insulated development *working* versions of the transferred project and perform their normal development tasks.

The build manager updates the local integration testing project, performing conflict detection. On a successful build that produces usable products, the build manager creates a new integration testing baseline for release `hello/2.0`. This makes these tested changes available to developers as a new baseline. Developer projects will automatically use this latest baseline when they are updated.

After the integration testing *prep* project has been rebuilt and tested, the DCM manager can send the completed tasks to the master using the `IL Transfer of Completed Tasks for hello Project` transfer set. This transfer set includes the tasks in the `All completed tasks for Release hello/2.0 from database IL` folder.

Note This action requires the *dcm_mgr* role. DCM performs dynamic role switching for this and all other DCM operations.

Use the **Generate** dialog to generate, send, and receive the transfer package:

1. Select the generate options.

   Select the `MA` destination database and the `IL Transfer of Completed Tasks for hello Project` transfer set.

2. Click the **Preview** button, and verify that the correct objects are in the transfer list (optional).

3. Generate the transfer package.

   This operation generates, transfers, and initiates the automatic receive of the transfer package.

Performing periodic master-to-satellite transfers

After MA receives transfers from the satellites, you must update the master project `hello-MasterInt`. During this operation, Rational Synergy reevaluates the `All Completed Tasks for Release hello/2.0 from database MA` folder. All of the completed tasks that were transferred from the satellite, as well as the local completed tasks, are added to the folder through the folder query.

You then complete testing on `hello-MasterInt` and update your `Master Integration Tested Tasks for Release hello/2.0` folder by copying all of the tasks from the `All Completed Tasks for Release hello/2.0` folder into it.

After this, the DCM manager sends the updated master integration tested tasks folder to the satellite using the `Master Transfer of hello Project` transfer set.

Note This action requires the *dcm_mgr* role. DCM performs dynamic role switching for this and all other DCM operations.

If you were sending data to more than one satellite, you would regenerate the transfer package for each satellite. The reason is this: When a transfer package is regenerated before it is sent to a database, DCM generates and sends only the objects that have changed or become new members of the transfer set since the transfer package was last sent to that database.

You periodically send the updated `Master Integration Tested Tasks for Release hello/2.0` folder to each satellite. This is usually done each time the master integration test is successfully completed.

Use the **Generate** dialog to generate, send, and receive each transfer package, or each database:

1. Select the generate options.

   Select the `IL` destination database and the `Master Transfer of hello Project` transfer set.

2. Click the **Preview** button, and verify that the correct objects are in the transfer list (optional).

3. Generate the transfer package.

   This operation generates, transfers, and initiates the automatic receive of the transfer package.

When the generate and send operations are finished, IL automatically receives the transfer set. At that point, IL has the latest integration-tested tasks of the master and can configure them into the `hello-IL#Int:project:MA#1` *prep* project.

Continuing to develop software

To create an insulated development *working* project in the satellite database, a developer checks out from `hello-1.0:project:MA#1` using a release value of `hello/2.0` and the purpose `Insulated Development`. The update properties will be automatically set up using the process rule `hello/2.0:Insulated Development` that was received from the `MA` master database.

The build manager periodically updates and builds the `hello-IL#Int:project:MA#1` integration testing *prep* project (for example, nightly). The update (reconfigure) operation collects the developers' latest checked-in objects and tasks and also includes the master integration tested tasks through the `Master Integration Tested Tasks for Release hello/2.0` folder.

After the build manager successfully rebuilds and tests the integration testing project, he creates a new baseline for release `hello/2.0` and purpose `Integration Testing`. The DCM manager then transfers the tasks that were completed in the satellite database to the master database using the `IL Transfer of Completed Tasks for hello Project` transfer set.

Baselines overview

Although it is possible to replicate baselines with DCM, the most common DCM methodology is to transfer folders and let each database create its own local baselines. Folders are more lightweight than baselines, therefore DCM transfers will be faster with folders.

If you do transfer baselines, note that the satellite database projects update members based on the latest baseline, regardless of which database it came from. Therefore, you need to be careful not to receive from a remote database a newer baseline that doesn't include your local tasks.

If you choose to transfer baselines, the best method is to create all baselines in the master database, and configure the satellite database process rules to be based on the latest baseline plus all local completed tasks.

Adding a satellite

You can add databases to a Master and Satellite DCM cluster by creating new satellites. For example, you could add a new database in Irvine, California with a `CA` ID. To add this satellite to the existing DCM cluster, use the same procedure as the one used to set up the destination database definition for the master and the first satellite:

1. Initialize the Irvine database to `CA`.

   For more information, refer to "Initializing the databases" on page 83.

2. Create the master destination database definition for `CA`.

   In the MA **Create DCM database Definition** dialog, add the destination database ID (`CA`), its description, and its properties.

3. Perform the initial transfer to the `CA` satellite.

   Use the same transfer set that was used for the IL database, and generate the transfer package for the new destination.

   For more information, refer to "Performing the initial transfer to the satellite" on page 86.

Summary

After completing this section, you should know how to initialize DCM databases, set up the transfers between them, and perform periodic updates using a Master and Satellite methodology. You are not limited to the methods used in this example. You can combine different methodologies and create whatever transfer sets are needed to meet your needs.

Note DCM also lets you generate, transfer, and receive transfer packages from the command line. Thus, DCM operations can be automated using UNIX or Windows scripting and scheduling capabilities.

8 Tracking Distributed Change Requests

Introduction

This section explains how to manage change requests using distributed configuration and change management tools. The following topics are discussed:

- "Overview of non-distributed change request tracking" on page 98
- "Overview of distributed change request tracking" on page 100
- "Planning a distributed change request methodology" on page 102
- "Planning a distributed change request methodology" on page 102
- "Types of DCM replication topologies" on page 108
- "Overview of different distributed change request methodologies" on page 103
- "Overview of Rational Change dialogs and usage" on page 112
- "Rational Change Distributed model parameters" on page 113

Overview of non-distributed change request tracking

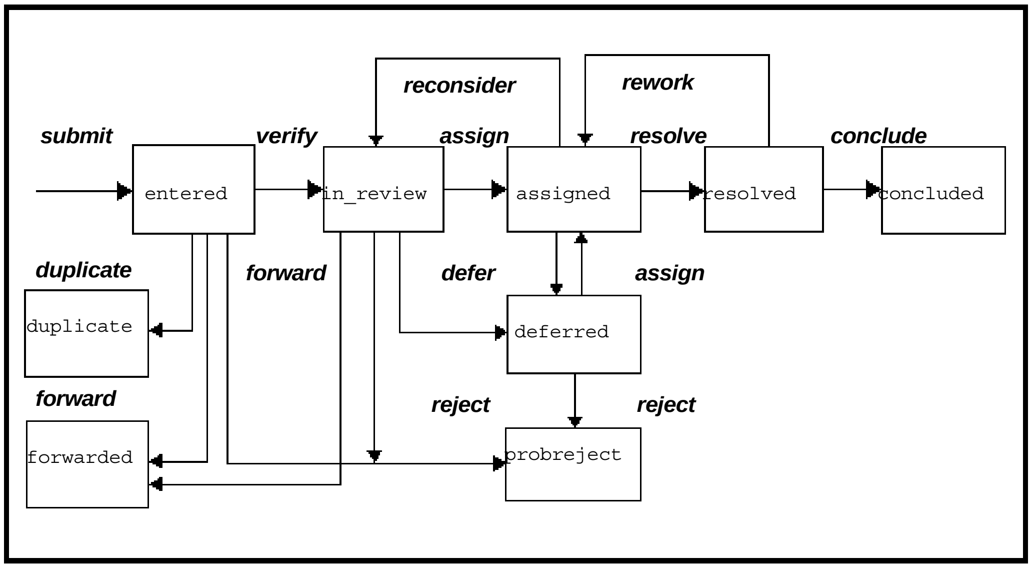
This section briefly describes IBM® Rational® Change, which is the non-distributed change request tool.

Rational Change

Rational Change is a Web-oriented, browser-based application for submitting and tracking change requests. This tool is integrated with Rational Synergy Distributed and is accessed from the Synergy Classic **Tools** menu. Rational Change lets you:

- Submit change requests
- Track change requests
- Run reports

Rational Change uses special objects called change requests to keep track of problems or issues submitted by users. A *change request object* describes a specific change request — its attributes include why and when the request was made, and who submitted it. The figure below depicts a sample Rational Change lifecycle; it shows some possible transitions and states of a change request.



Rational Change does not provide any means of replicating change request data. For more information about Rational Change, see the documentation Web site (`http://publib.boulder.ibm.com/infocenter/rsdp/v1r0m0/index.jsp`).

Overview of distributed change request tracking

This section briefly describes Distributed Rational Change (DCS), which is the distributed change request tracking package.

Supported functionality

DCS enhances Rational Synergy Distributed and Rational Change functionality as follows:

- Change requests and tasks in any state are eligible for replication.
- A given change request or task may be modified or transitioned in only one database at a time within a DCM cluster. This is controlled by the `modifiable_in` attribute and security rules.
- Permission for the controlling database to modify or promote a specific change request or task may be handed over to another database in the DCM cluster.
- Change request and task resolvers can be chosen from a list that can include external database users listed in a Rational Change file.
- Transfer sets can define a change request scope that includes a change request query. The change request scope defines: 1) Whether change requests should be included, and, if so, 2) Whether change requests include their associated tasks, and, if so, 3) The associated objects of each associated task.
- The change request scope and change request query of a transfer set can be defined via the GUI or CLI.
- Transfer sets can define whether the change request scope is cumulative or not. When disabled, the indirect change request members of the transfer set are precisely defined by the change request query and scope; any members that no longer match the query are removed. When enabled, the scope query adds to the existing indirect change request members and all existing change request members are preserved. This option is typically enabled when change requests are replicated on a selective basis (using a specific change request query), especially when intermediate hub databases are used. This allows an object that was once replicated to a database to continue to be updated even though it is no longer found by the change request query defined for the transfer set.
- Only tasks that are assigned to a developer and modifiable in the current database can be used as the *current task* for the developer, or used to *check out* or *check in* objects. Such tasks may have been created and assigned in a different database and replicated to the current database.

- When a task is selected in a DCM-enabled database, it is displayed in the *`dbid#nnnn`* format; where *`dbid`* is the database ID, # is the DCM delimiter, and *`nnnn`* is the task number. This data appears in Rational Synergy Distributed task dialogs and certain fields in Rational Change.
- Parent/child and related change requests are supported.

Configuring DCS

DCS can be configured as explained in the following list.

- Transfer sets can exclude non-modifiable tasks. By default, tasks in any state are eligible to be included in a transfer package. If needed, the transfer set definition can exclude all incomplete tasks.
- Transfer sets can exclude imported objects. By default, transfer sets allow a database to generate a transfer package that includes objects that were received from other databases. This allows replication through a chain of such databases. If this option is disabled, the database in which the object was generated must always perform the replication to the receiving databases of the object.
- DCM model parameters allow DCS behavior to be adapted for complex DCS methodologies. These methodologies are described in "Planning a distributed change request methodology" on page 102.

Planning a distributed change request methodology

Before DCS can be used effectively, you need to develop a plan for the flow of information between databases. This plan must address the Rational Change lifecycle (transitions between states), and the corresponding DCM configuration Management flow (handover of control from one database to another).

As a starting point, consider the following questions. Your answers will help you determine your DCS methodology and topology — sample methodologies are described in "Planning a distributed change request methodology" on page 102, and sample topologies are discussed in "Types of DCM replication topologies" on page 108.

- In which database(s) will change requests be entered?
- Will change requests be verified and then transitioned in the same database in which they were entered?
- Will change requests be assigned to personnel in remote databases?
- Will the tasks for a change request be created and assigned from a single controlling database for that change request?
- Will tasks be created and assigned in more than one database against the same change request?
- In which database(s) will change requests be resolved?
- In which database(s) will change requests be concluded?
- Do reporting or query requirements require visibility of change requests from other databases? If so, do these requirements include a breakdown of associated tasks?
- Is reporting on associated objects required?
- If change request replication is required, does it need to be selective? If so, will it based on product name, or will it be based on some other criteria?
- Is replication of *registered* or *task_assigned* tasks required? Or is replication of tasks that are in the *completed* state sufficient?
- Are there issues regarding commercial sensitivity of configuration Management data in remote databases? How will these be addressed?
- What email triggers are required? How will email notification against different databases be implemented?

Overview of different distributed change request methodologies

This section presents various ways that DCS can be used to manage change requests. The list is by no means complete — your methodology may vary from the examples shown here.

Sample methodology 1 – Managing change requests only in the creating database

The simplest methodology is to transition change requests and tasks in the same database in which they are created. You can then use DCM to replicate change requests and/or tasks to other databases for use in reports. This plan uses an Rational Change workflow — a DCM workflow is not required. DCM replication allows change requests and/or task data to be used in read-only queries and/or reporting. Therefore, the default DCS model parameters are used. Replication can be organized as a unidirectional publish and subscribe methodology (for more information, see "Publish and Subscribe methodology" on page 23).

Sample methodology 2 – Entering all change requests in a single database

Another methodology is to enter all change requests in one database. This could be a main product management database from which a company controls changes to a number of applications. Following is a typical Rational Change and DCM workflow for this methodology:

1. A change request is entered in the main product management database.
2. The change request is verified, rejected, or marked as duplicate in that database.
3. The change request is assigned to a product development team leader, and then it is handed over to the corresponding development database.
4. All associated tasks are entered and assigned in one development database.
5. Upon completion of all tasks, the product development team leader resolves the change request and hands it over to the main product management database.
6. A concluder in the main product management database either concludes the change request, or sends it back for rework in the development database.

Replication of tasks from the development database(s) back to the main product management database may also be required. This allows reports to be created and keeps product managers informed about the progress of tasks. In other cases, details of *completed* tasks might be sufficient, and the transfer set(s) could exclude incomplete tasks. In still other cases, it may be unnecessary to transfer any tasks from the development database(s) to the main product management database.

If selective replication is needed, a change request query is usually based on product name and/or product subsystem (and possibly the status of the change request). In such cases, transfer sets should **not** exclude imported objects — otherwise, handover of the imported objects back to the main product management database will never take place.

Special cases

If you use Sample Methodology 2, special processing is sometimes required. The following paragraphs describe such cases.

When Rational Change is used to defer an *assigned* change request, any associated *assigned* tasks are also deferred. When the change request is reassigned, any associated deferred tasks are once again *assigned*.

In the above scenario, a change request that is *assigned* and being worked on in a development database could be deferred by a team leader and handed back to the main product management database. Later, the change request could be reassigned. However, the associated tasks might not exist in the main product management database. Even if they did, they could not be modified there.

To handle this, DCS checks change requests during the receive operation. Thus, in this example, DCS would find the change request in the *assigned* state that is being received in the development database, and check for any associated deferred tasks and reassign them. Details of such automatically-transitioned tasks are included in DCM email notifications.

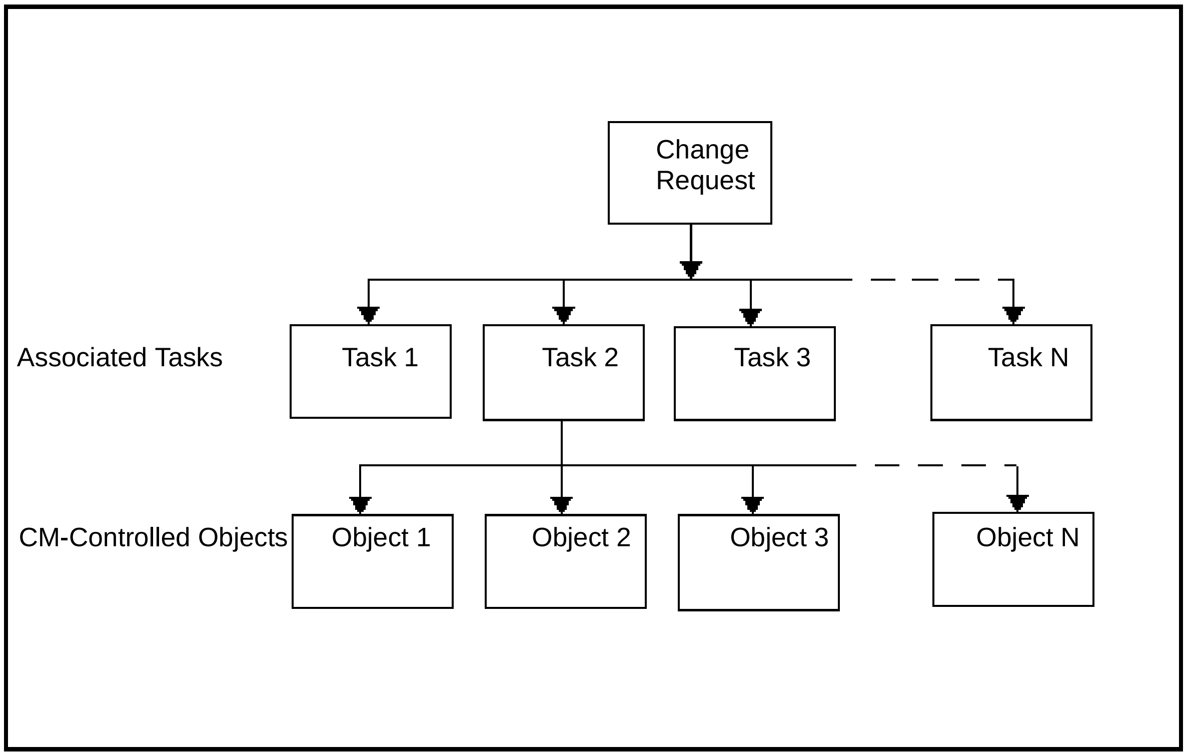
The default DCS model parameters are adequate in most cases, provided you use one or both of the following implementations:

- If you are replicating no tasks or only some tasks, then you would: 1) Replicate change requests to a development database only when they are in the *assigned* state, and 2) Replicate change requests back to the main product management database only when they are in the *resolved* state.
- Replicate all tasks either with their associated objects, or without their associated objects.

The point is this: By default, when an object is received, any relationships from that object to other objects are reproduced in the receiving database. Moreover, any other appropriate relationships that exist in the receiving database, but are not in the transferred data, are deleted.

As shown in the figure below, the main relationships in Rational Change are as follows:

- Change request to task (`associated_task`)
- Task to object (`associated_cv`)



Due to the object relationships shown in the above figure, the following problem occurs when only some tasks are replicated back to the main product management database: If a change request is replicated to the development database, the change request is associated only with some, but not all, of the appropriate tasks. By default, during a DCM receive operation, the other tasks are disassociated from the change request.

To avoid this problem, do one of the following: use the correct DCM workflow associated with the Rational Change lifecycle, make sure that all of the associated tasks are in the databases where the change request object exists, or disable the image handling when change requests are received — this removes all of the aforementioned implementation restrictions; however, task deletions or the removal of relationships will not be replicated (for more information about image handling, see "Rational Change Distributed model parameters" on page 113 and the *Help* that is referenced in that section).

Note By default, the above problem does not exist with tasks and their associated objects. The default behavior of DCS is to skip the receive of tasks that are modifiable in the receiving database. This behavior can be modified; however, read "Rational Change Distributed model parameters" on page 113 before doing so.

Sample methodology 3 – Using central CCB database/satellite development databases

With this methodology, each change request is sent to a central Change Control Board (CCB) database; where the CCB is a team that reviews and dispositions problems, issues, and change requests. To accomplish this, you can either:

- Enter each change request directly in the CCB database — such problems are created as parent change requests.

  OR

- Set the **Work in database** field to the CCB database when you enter the change request in a local database. Hence, the change request is replicated to the CCB database where it can be processed — such problems are created as parent change requests.

Once a parent change request is created in the CCB database, the CCB creates one or many *child change requests* associated with that parent, and sets the **Work in database** field for each child change request to correspond to the satellite development database in which action will be taken on that child.

All development work takes place in the satellite development databases. Thus, all tasks and associated configuration items reside in the appropriate development database(s). When a child change request is completed or rejected, email is sent to the CCB; this can be accomplished using an email trigger. The email message instructs the CCB to review the status and progress of the parent change request. The CCB cannot complete a parent change request until all of the child change requests are in a terminal state (such as *concluded*).

Typically, only change requests are replicated back to the CCB database. The CCB does not usually need to see the breakdown of each child change request into its tasks, or view the breakdown of tasks to configuration items. Users can obtain such details from the relevant development database(s).

Sample methodology 4 – Using multiple databases/single database for associated tasks

In this methodology, change requests may be entered in any database, and handed over to another database for action. This methodology has similar implementation issues as Sample Methodology 3. However, this methodology may be more complex because it uses point-to-point communication, rather than the hub (product management database) and spoke (development database) topology used in Sample Methodology 3. For more information, see "Point-to-Point topology" on page 108 and "Hub and Spoke topology" on page 108.

Types of DCM replication topologies

DCM does not rely on real-time bi-directional flow between source and destination databases. Therefore, you need to let DCS know how to replicate and hand over control of change requests across databases. Various replication topologies can be used. In order for DCS to safely hand over control of change requests, there must be one unique defined replication process through which handover takes place. We will focus on Point-to-Point and Hub and Spoke replication topologies.

Point-to-Point topology

With the Point-to-Point topology, each database generates a transfer package for either:

- A database to which information is to be sent, and/or
- A database to which control of a change request is to be handed over.

For example, imagine three databases (A, B, and C) that want to replicate change request information between each other, and possibly hand over control. Using the Point-to-Point topology, database A can generate to B and C any change requests that it wants to either send, or hand over control. Similarly, database B could generate change requests to A and C, and database C could generate change requests to A and B. When only two or three databases are involved, this is the simplest DCM replication topology.

For more details, see “Database and handover of control” on page 116.

Hub and Spoke topology

In the Hub and Spoke topology, spoke databases do not replicate directly between each other. Instead, they replicate via one or more hub databases. Thus, DCS does not know by default which replication path lets data go from one spoke to another.

For more details and examples of set-ups, see “Database and handover of control” on page 116.

Overview Rational Synergy Distributed and Rational Change Distributed (DCS)

This section describes how using DCS will:

- Activate the Rational Synergy Distributed dialogs, and
- Extend the way that Rational Synergy Distributed can be used.

Rational Synergy Distributed dialogs

DCS extends the functionality of Rational Synergy Distributed **Transfer Set** and **Task** dialogs for a given DCM database. Depending on the dialog, DCS does one or more of the following:

- Activates the **Problem Scope** and **Problem Query** fields.
- Activates the **Show Problems** button.
- Extends the dialog to activate the database in which the focus (highlighted) task may be modified.
- Shows whether or not non-completed tasks (that is, tasks that are **not** in the *completed* state) are excluded from the transfer set.

Problem scope and query

DCS extends transfer set definitions to include a problem scope and a problem query. These definitions let the transfer set automatically include change requests that are based on a query. The following problem scopes are supported:

- **None** — No change requests are included.
- **Problems Only** — Change requests found by the specified query are included, but not their associated tasks.
- **Problems and Tasks** — Change requests found by the specified query are included, along with their associated tasks. You can exclude *incomplete* tasks by using the exclusion properties window of the transfer set.
- **Problems Tasks and Objects** — Change requests found by the specified query are included, along with their associated tasks. If an associated task is included, its associated objects are also included if they are static. The non-static associated objects of a task are excluded by DCM exclusion rules.

For any problem scope other than **None**, the transfer set can define a change request query expression. The expression does not need to include a query clause for problem types — a query clause is automatically added when the query expression is used. Thus, a blank query string returns all change request objects. The query expression can include any problem attribute using standard query syntax. For example, a query that is based on product name might be: `"product_name='foobar'"`

Problem query strings must be typed. However, you can use the problem query dialog to help you determine an appropriate query string to specify for a transfer set. The following table shows some standard keywords that can be used in problem query strings in transfer set definitions.

Keyword	Resulting Action
%to_db_id	This is replaced with the `dbid` of the DCM database for which the generate is performed. This allows the same transfer set definition to apply to a number of target databases.
%from_db_id	This is replaced with the `dbid` of the current database from which a transfer package is being generated.

DCM initialization

When a database is initialized to use DCM, DCS extends this operation so that all change request and task objects that have `created_in` and `local_to` attributes also have a `modifiable_in` attribute whose initial value is the current DCM `dbid`. Moreover, when you create change requests or tasks in such a database, you also set the `modifiable_in` attribute. For more information, refer to "Initializing DCM" on page 11.

Remote resolvers and email triggers

You can choose a resolver who is not an authorized user of the local database. This is possible because DCS extends the user list to include a user-definable set of names from `extusers.dft`. This file contains a list of external user names (one name per line). Blank lines and lines starting with the comment character (#) are ignored. The external user names are the userids of the users as defined in other databases.

In Rational Change, change requests and tasks are always assigned to local resolvers. Thus, email triggers only have to handle local user names, and setup email aliases for the local user names.

In DCS, however, the change request or task resolver may be at a remote database. Moreover, this user may not exist at the local site. Therefore you must either setup email aliases for such remote users, or adapt the trigger scripts to pass the target `dbid`. You must consider this when constructing email addresses.

The email notification text may need to include the DCM `dbid` of the database in which the change request or task is modifiable. By default, the notification text includes the path of the database from which the assignment was made. However, the change request or task may only be modifiable in another database. The trigger script will not be able to automatically determine the path of the remote database. Hence, the notification may have to include only the DCM `dbid`, or the trigger script may have to test the `dbid` and map it to the appropriate remote database path.

Overview of Rational Change dialogs and usage

This section describes how DCS:

- Activate the Rational Change dialogs, and
- Extend the way that Rational Change can be used.

Buttons and fields — special behavior

Some DCS dialogs have buttons that are either active or inactive based on whether the focus (highlighted) object is modifiable in the current database. By default, all buttons that save attributes for and/or transition existing objects are only active when the focus object is modifiable.

To enhance readability, Rational Change fields are never grayed out (even when the focus object is not modifiable).

Making changes in the current database

A **Modify** button is displayed in Rational Change dialogs only when you can modify the displayed object. In Rational Synergy Distributed, when you click the **Modify** button, DCS checks `modifiable_in` attribute of the object. If the object is not modifiable in the current database, then a confirmation dialog is displayed. This dialog serves as a warning; it forces you consider whether it is safe to make changes in the current database.

If the object is modifiable in another database, it is best to make changes there and not override the security rules. However, if the object is modifiable in the current database and then is handed over to another database, it is safe to modify the object (provided it has not been included in any DCM transfers; because once the object has been included in a transfer package, the package may have been received, and, hence, the object may have been modified elsewhere).

In this example, if you continue despite the warning, then changes made in another database could overwrite those made in the current database. Such changes will be lost without further warning. Clearly, the DCS *pt_admin* role, which is required in order for you to make these changes, carries a lot of responsibility.

Rational Change Distributed model parameters

DCS model parameters control the behavior of DCS and are implemented as attributes of the model object. Following is a list of DCS model parameters:

- `dcm_noimage_import_types`
- `dcm_allow_assoc_tasks_across_dbs`
- `dcm_problem_types`
- `dcm_relation_import_types`
- `dcm_problem_receive_actions`
- `dcm_task_assigner_role`

All databases in a DCM cluster should use the same settings for these parameters. Users in the *ccm_admin* role can change these settings using the following commands:

```
ccm query /t model /n base
ccm attr /m attribute_name /v "new_value" @1
```

For a complete description of the DCS model parameters, refer to *Synergy Classic Help*, specifically Default Options.

9 Advanced DCM Topics

Introduction

This section is for administrators who need more details about DCM operations. The following topics are included:

Database and handover of control

DCM protects objects by enforcing a policy that for any specific object, it is only modifiable in one database in a DCM cluster at any point in time. In all other databases, the object cannot be modified unless you are in `ccm_admin` role. The only exception to this is that an object may be transitioned to a later state.

The controlling database for an object is determined by either its `modifiable_in` attribute or its `created_in` attribute. If the `modifiable_in` attribute exists, then the controlling database for the object is explicitly defined and the attribute value specifies the DCM database ID of the controlling database. If the `modifiable_in` attribute does not exist, then the controlling database is determined by the value of the `created_in` attribute.

The control of an object can be handed over from the current controlling database to some other database by changing the `modifiable_in` attribute value from the current DCM database ID to the ID of the new controlling database. This marks the object as pending handover of control and this protects the object against update until acknowledgment has been received that the control has been successfully passed to another database. This acknowledgement is achieved by examining the object when received from another database. Handover of control is normally only provided for change requests, tasks, release definitions, processes, process rules and folder templates. However, its use is not limited to any specific type of object.

DCM protects objects in a number of ways:

- Security rules prevent anyone except a user in *ccm_admin* role from modifying an object that is either controlled in another database or is pending handover of control.
- DCM receive will skip objects on import and not update them if the object is controlled in the receiving database or is pending handover of control. DCM includes a skipped objects report in the DCM Event Log and any DCM receive email that is sent. This prevents the controlling copy of an object being potentially overwritten by an older copy of itself from some other database.
- When checking in objects with a check in comment, the comment will not be updated on any objects that are controlled in other databases.

By default, when a DCM database definition is created, the **Handover allowed** setting is FALSE and you will not be able to hand over control to that database. If you wish to handover control of an object to another database, change this setting to TRUE.

Transitions to a controlling database

By default, DCM never allows the controlling copy of an object to be overwritten by a copy of itself from any other database. For further details, see “Database and handover of control” on page 116. This protection also applies to replicating transitions. For example, imagine a case where an object is created in database A, transitioned to the *integrate* state and replicated to database B. The object is controlled in database A by default. In database B the object is transitioned from *integrate* to *released*. By default, if the object is sent from database B to A, the object in A will not be updated and will remain in the *integrate* state. This behavior is intentional as database A "owns" the object. Without such protection, a build manager in another database might transition the object to the *rejected* state, even though in database A that object is included in *released* projects.

When a single component or application is being developed across multiple databases, it may be desirable to allow some transitions that are made in a non-controlling database to be applied to the controlling database. However, not all such transitions might be desired. For example, you may wish to allow a transition from *integrate* to *released* to be replicated back to the controlling database but not a transition from *integrate* to *rejected* or *released* to *rejected*. The DCM setting Receive Control Transitions may be changed to implement such a policy.

On a DCM receive, if an object in the transfer package is controlled in the receiving database, DCM examines the status of the existing object and the status received from the other database. It checks the Receive Control Transitions DCM setting, and if the transition is allowed, transitions the object to its new state. However, no other property of the object will be updated and the object will still be listed in the skipped objects report. The `status_log` attribute will show the original database that made the transition and also include an entry for when that transition was applied back in the controlling database.

Replication of generic and release-specific processes

A process is a collection of generic process rules for some set of purposes that together control how the update (also called update members and reconfigure) operation includes changes in projects or project groupings and how those changes flow from development through one or more testing stages to release. A generic process rule is a pattern that is independent of release and that defines how changes are gathered when an update operation is performed.

When a release definition is created, the generic process rules are used to define how the specific process rules for that release are formed. The resultant release-specific process rules are then used for controlling the process for that release and what changes are included when a project or project grouping for that release is updated. The release definition and the set of process rules it uses defines the valid purposes that may be used for the release and how a project or project grouping with that release and one of those valid purposes is updated.

DCM in Rational Synergy Distributed 6.5 and later supports the replication of processes and generic process rules, and the replication of releases and their associated release-specific process rules. Two separate mechanisms are supported:

1. Processes and/or process rules may be explicitly added to a DCM transfer set. When a process is added, it also includes the process rules used by that process. When a process rule is added, either directly or indirectly through adding a process, the folders and folder templates of the process rule are also indirectly added.
2. Release definitions and their associated process rules may be automatically added as indirect query members through the use of a **Release Scope** and a **Release Query**. The scope and query define which releases should be automatically added, and for each such release, its process rules are also automatically added.

Built-in predefined generic process rules and folder templates are always excluded from DCM transfer packages by built-in exclusion rules. These built-in process rules and folder templates cannot be modified or deleted and they will always exist identically in each database. They therefore do not need to be replicated.

Note When a folder is added indirectly to a transfer set because of a process rule, the tasks of that folder are **not** added. If you wish the tasks of that folder to be included, you must explicitly add the folder as a direct member of the transfer set.

Local versus centralized administration

DCM supports local and centralized administration of processes, process rules, folder templates and release definitions, or a mixture of both.

With local administration, an object is locally controlled and updated. If that object is included in a DCM transfer package to that database, it will be skipped during the import phase of the DCM receive. The advantage of local administration is that it allows each database to separately control the properties of an object and this can be independent of the properties used in other databases. The disadvantage is that if it is desired that the properties be kept consistent across databases, then they must be separately maintained and updated in each database, increasing the burden of maintenance.

With centralized administration, for each object there is a single master and controlling copy managed from a defined database. Changes to that object are only made in its controlling database. Copies of that object are replicated to other databases in the DCM cluster and updated with any changes made from the controlling copy. DCM allows the control of the object to be handed over from the current controlling database to some other specified database. After the handover of control, that object is then only administered in its new controlling database. Different objects can be controlled in different databases. However, a more normal usage pattern is to control all of the objects, such as release definitions (or at least release definitions for a specified component), from a single database. The advantage of centralized administration is consistency across the DCM cluster and a decreased maintenance burden on build managers. The disadvantages are that changes can only be made in the controlling database and the properties of the object cannot be different in different databases.

DCM also supports a mixed approach. For some releases, a user might centrally control the release and its process rules, and for others locally control them. DCM also allows a release to be centrally controlled, but have some of the process rules for that release be locally controlled, so that a different process may be implemented in a specific database. The user may take local control of a release-specific process rule in order to make some local changes for that database. The process rule will remain associated with the release as a valid purpose for that release.

DCM allows users to switch between local control and centralized control models of usage. If an object has been defined in multiple databases, then it will be locally controlled in each of those databases. If you wish to centrally control the object, then perform the following steps:

- Decide which database should control that object and ensure that the object is set to be locally controlled in that database.
- In all other databases where the object exists, set the controlling database to accept control from the controlling database.
- Perform DCM replication from the controlling databases to other databases using transfer sets that include that object. It may be necessary to perform a generate since **Never** if the object was already a member of the transfer set to ensure that it is included in the generated transfer package

To break centralized control and revert to a locally controlled object, use the **Select Controlling database** dialog or command line option and specify local control. The details of the object will be unchanged. However, the object will be given a new and unique cluster ID to denote that it is a different object and is not to be updated on DCM receive.

Replication of release definitions and associated process rules

Replication of release definitions allows a user to centrally administer release definitions and their associated process rules and then automatically update other databases in the DCM cluster.

If a DCM cluster uses release values to configure shared data into projects, the release values of any transferred objects must be defined as release definitions in the destination database.

The replication of release definitions and information is controlled by the following parameters and settings:

- Each release definition has a setting named **Allow DCM Transfer.** By default, this option is selected when a release for a new component is created, or is set to the previous release when a release based on a previous release is created. Clearing this option means that details of that release will always be excluded from all DCM transfer packages.
- Each transfer set has a setting named **Release Scope** and a setting named **Release Query**. The release scope has three settings:

  Releases and Templates - Replicates the release definitions found by the release query and their associated process rules. This is the default when creating a new transfer set.

Releases - Replicates the release definitions found by the release query but not their associated process rules. If this scope is used, the release definition as replicated to another database might appear not to have any valid purposes, unless the process rules already exist for that release in the remote database.

None - No query for release definitions is performed. The only process rules that will be replicated will be those that are added explicitly to the transfer set or indirectly by adding a process definition to a transfer set.

The release query defines a query expression that will be used to automatically add matching release definitions as indirect query members of the transfer set. A blank string means query for all release definitions. This is the default when creating a new transfer set.

- In each receiving database, the DCM setting Update Release Definitions controls what actions are taken, if any, to process release information in a transfer package that is being received. The settings and their corresponding actions are described in the table below.

Update release definitions	Action
`none`	All release information in the DCM transfer package is ignored.
`active`	Release definitions in the package are used to create and/or update release definitions in the receiving database. However, a release definition is created only if it is an active release. New release definitions for inactive releases are not created. Existing release definitions that are inactive and have been received before continue to be updated. This is the default setting.
`inactive`	Release definitions in the package are used to create and/or update release definitions in the receiving database. Release definitions for both active and inactive releases are created or updated.

When a DCM generate is performed, DCM will include only release definitions, process rules or folder templates that have been modified or became members of the transfer set more recently than the last generate time for that transfer set and database. Hence, if you perform a DCM Generate since **Never**, and the transfer set release scope is set to **Releases and Templates**, and the release query is blank, all the release definitions and their associated process rules and user-defined folder templates will be included in the DCM transfer package.

To check or manually change release definitions, use the **Browse Releases** dialog or the `ccm release` command.

Replication of DCM database definitions

By default, DCM includes some details of the DCM database definitions in the source database in the DCM transfer package. This is used to create or update corresponding DCM database definitions in a receiving database on a DCM receive. This helps to automate the steps in setting up DCM database definitions and to make details such as description, location and administrative information consistently available across the cluster.

Not all of the information in a DCM database definition is replicated. Transfer modes, paths, automatic receive settings and so on may need to be different for the same database definition in different databases. For example, imagine a cluster consisting of databases `NY1` and `NY2` in New York, and databases `LO1` and `LO2` in London. The DCM database definition for `NY2` might use direct transfer mode from `NY1`, but ftp from `LO1` and `LO2`.

Note that if the DCM database ID of a database is changed, this change **is not automatically replicated**. You **must** notify the DCM administrators of all other databases of the changed identifier so that the corresponding DCM database definition in each database is updated. See “Changing a database ID” on page 62 for further details.

Email notification

Rational Synergy Distributed uses a default mail tool for DCM email notification. To use a mailer other than the Rational Synergy Distributed mailer, enter the following line in the `[Options]` section of your `ccm.ini` file:

```
mail_cmd = user-defined mail command
```

The syntax for *`user-defined mail command`* depends on the mailer that is being used. However, mailers typically require recipient, subject, and content options and arguments.

For example, following is a `mail_cmd` definition for `ccmail`:

```
mail_cmd = C:\ccmail\mailer.exe -r %recipients -s %subject -f
%content
```

The `%recipients`, `%subject`, and `%content` arguments are automatically expanded by Rational Synergy Distributed using the information that you supply in its dialogs.

Fully-expanding reconfigure properties

If **Fully Expand Reconfigure Properties** is selected in the transfer set of the source database, projects are fully-expanded during the DCM Generate operation. This means that projects include:

- All associated objects of tasks— even if these objects are not members of the project hierarchy
- All folders and tasks in the update properties of the project
- All subprojects of baselines referenced in the update properties of the project
- All of the previous bullets — even if the project is in a non-modifiable or static state (for example, *released*, *integrate*, *test*, *sqa*, and so on)

The following table shows what is included when each type of object is added to a transfer set, and the **Fully Expand Reconfigure Properties** option is selected

.

Object type	Objects added to the transfer set	History
folder	The folder itself, its member tasks, and all objects associated with the tasks.	NA
task	The task itself and all objects associated with the task.	NA
project	The project itself and all member objects of the project hierarchy. For non-static projects such as *prep* projects, the following are also added: -- Any project baseline that the project is using -- All folders and tasks in the update properties of the project' -- All tasks in the folders. For all projects (including static projects such as *released* projects), the following are also added: -- Any project baseline that the project is using, and all subprojects of that baseline -- All folders and tasks in the update properties of the project -- All tasks in the folders -- All objects associated with such tasks; even if they are not a member of the project hierarchy.	optional

Object type	Objects added to the transfer set	History
baseline	The baseline object itself, its project members and task members. Baseline project members and task members are further expanded as described above.	NA
source	The object itself and the its associated task(s).	optional

If the default is used (that is, if **Fully Expand Reconfigure Properties** is not selected):

- Static projects will **not** include expansion of folders, tasks, and associated objects of the tasks.
- Prep projects will include folders and tasks, but not objects that are associated with the tasks — only objects that are used in the project hierarchy will be included.

To learn more about the **Fully Expand Reconfigure Properties** option, see "Creating a transfer set" on page 42 and "Adding objects to a transfer set" on page 43.

Transfer of associated baselines

The **Include Associated Baselines** transfer set option controls whether baselines associated with tasks or projects are also included as indirect members of a transfer set when those tasks and projects are members of the transfer set. The default setting is for this option to be cleared (not selected). Note that baselines can be added as direct members of a transfer set, and this is unrelated to the **Include Associated Baselines** option.

The advantage of including associated baselines in a transfer set is that when a project or task is replicated to another database, its associated baselines accompany it to the destination database. This allows users to perform a Find Use and see in which baselines the project or task was included.

In the case of an application that is built by assembling different components that are developed in multiple databases, including the associated baselines can cause problems for update (reconfigure). This is because in any one database, a baseline will have a subset of projects relating to the components developed in that database and it will not represent all the projects across all the databases. If that baseline is sent to another database, update may choose it as the latest baseline. However, it will not have all of the project members required to select a baseline project, and update may be unable to select a suitable candidate version of some projects.

If you are using the predefined **Entire database** transfer set, the all baselines will be replicated to each database and the **Include Associated Baselines** option setting will not matter.

If you are developing an application in a database using a release value that is specific to that application, and you replicate all of the projects for that application, then selecting **Include Associated Baselines** could be of benefit to developers. However, if you only replicate some of the projects to each database, then selecting on this option could be harmful and you should use the default setting.

Structure of a transfer package

The transfer package contains three objects that DCM reads and writes as binary: the information file, the preview file, and the contents file or directory. DCM creates these objects in the source database generate_directory when the source database generates a transfer package. The files remain in the *`generate_directory`* until the source database receives confirmation that DCM has successfully transferred the files to the destination directory. When the transfer is finished, the transfer files reside in the receive_directory of the destination database— An exception to this is when the Direct mode transfer mode is used (that is, the source database generates the objects directly in the *`receive_directory`* of the destination database). The files remain in the *`receive_directory`* directory until DCM successfully loads them into the destination database.

The package contents file or directory may contain up to 5 parts:

- User data such as files, directories, tasks as defined by the transfer set membership. This is held under a `data` directory.
- Optional release part under a `releases` directory that holds information on release definitions.
- Optional templates part under a `templates` directory that holds information on folder templates.
- Optional process part under a `process` directory that holds information on processes and process rules.
- Optional types part under a `types` directory that holds information on type definitions.

The transfer file names have the following syntax:

`<FROM_DB_ID><TSET_NUMBER><TO_DB_ID><TIME><name>`

where *`name`* is: `dcm_info.txt` for the DCM information file, `dcm_preview.txt` for the DCM preview file, `.tar.gz` for zipped package contents, and `.dcm` for unzipped contents. The portion of the name that precedes *`name`* is identical for all files that are part of a given transfer package. For descriptions of the information that precedes *`name`*, see the table in the section below.

DCM information file

The `dcm_info.txt` file contains DCM database and transfer set parameters that are used by the destination database. You can see most of this information in the **Generate** dialog.

DCM preview file

The `dcm_preview.txt` file contains a list of the objects in the user data part of the transfer package. The **Preview Transfer** dialog uses this list at the destination database. The file contains one header line that labels the fields contained in the subsequent lines, followed by one record for each object in the transfer set. Note that the preview file does not contain any details of the information included in the administrative part of a DCM transfer package. For example, type definitions or release definitions included with the package are not listed.

Following is an example of a DCM preview file:

```
displayname@@@name@@@version@@@cvtype@@@subsystem@@@owner@@@sta
tus@@@release@@@created_in@@@local_to@@@create_time@@@modify_ti
me@@@task_synopsis@@@problem_synopsis@@@description
example-
1@@@example@@@1@@@project@@@A#1@@@ccm_root@@@prep@@@1.0@@@A@@@A
@@@1161698530@@@1161698548@@@@@@@@@
example-
1@@@example@@@1@@@dir@@@A#1@@@ccm_root@@@integrate@@@1.0@@@A@@@
A@@@1161698533@@@1161698555@@@@@@@@@
All 1.0 Integration Testing Projects from database
A@@@1.0%003aintegrate@@@1@@@project_grouping@@@A#1@@@ccm_root@@
@prep@@@1.0@@@A@@@A@@@1161698535@@@1161698537@@@@@@@@@
A#1@@@1@@@1@@@folder@@@A@@@ccm_root@@@prep_folder@@@1.0@@@A@@@A
@@@1161698537@@@1161698537@@@@@@@@@All Completed Tasks for
Release 1.0
example.txt-
1@@@example.txt@@@1@@@ascii@@@A#1@@@ccm_root@@@integrate@@@1.0@
@@A@@@A@@@1161698548@@@1161698555@@@@@@@@@
```

Package contents

The `.tar.gz` file or `.dcm` directory contain the objects that are being transferred. During the generate operation, DCM does the following to create this file or directory:

1. If type definitions are not excluded, DCM creates a temporary directory in *`generate_directory`* named *`<FROM_DB_ID><TSET_NUMBER><TO_DB_ID><TIME>`*`\types` and exports each user-defined type definition. The format is equivalent to using the `ccm typedef /export` command.
2. If release definitions are not excluded, DCM creates a temporary directory in *`generate_directory`* named *`<FROM_DB_ID><TSET_NUMBER><TO_DB_ID><TIME>`*`\releases` and exports the release definitions that are to be included. It also constructs a

pseudo release table for these release definitions for compatibility with prior releases of Rational Synergy Distributed and creates a `<FROM_DB_ID><TSET_NUMBER><TO_DB_ID><TIME>\rel_tbl.txt` file.

3. If templates are not excluded, DCM creates a temporary directory in *`generate_directory`* named `<FROM_DB_ID><TSET_NUMBER><TO_DB_ID><TIME>\templates` and exports the templates that are to be included.
4. Determines which objects to include in the data part of the transfer package.
5. Creates a temporary subdirectory in *`generate_directory`*. The directory is: *`<FROM_DB_ID><TSET_NUMBER><TO_DB_ID><TIME>\data`*
6. Exports each object in the transfer list to the temporary data subdirectory.
7. If the package is to be zipped, it collects all of the files and directories under the *`<FROM_DB_ID><TSET_NUMBER><TO_DB_ID><TIME>`* temporary directory into one file using `ccm_tar` piped through `ccm_gzip`, and removes the temporary subdirectory. If the package is not to be zipped, the package contents directory is renamed to `<FROM_DB_ID><TSET_NUMBER><TO_DB_ID><TIME>.dcm`

Transfer modes

A transfer mode is a method by which DCM sends transfer packages to a destination database.

DCM provides the following transfer modes:

- Manual copy
- Direct
- Local copy
- Remote copy
- File transfer protocol
- User defined

These methods are discussed in detail in the sections that follow. Cross-platform information is included for transfers between UNIX and Windows databases.

To see how to choose a transfer mode, refer to “Creating a DCM database definition” on page 41 or “Generating the transfer package” on page 53.

Manual copy mode

User-Supplied Options: Transfer Path (optional)

When you generate a transfer package using the Manual Copy mode, DCM places the transfer package in the *`generate_directory`*, but does not send them. The Manual Copy mode is used when you do not want to automatically send a transfer package, or when automatic methods of transfer are not possible.

This mode requires that you manually move the transfer package from the *`generate_directory`* to the *`receive_directory`* of the destination database. Typically, this is done using magnetic tape or writable CD-ROM.

For more details on transfer files, see “Transfer of associated baselines” on page 127.

Direct mode

User-Supplied Options: OS, Path, and Transfer Path

When you generate a transfer package using the Direct mode, the transfer package is created directly in the transfer path of the destination database. Thus, the Direct mode can only be used when the source and destination databases are on the same LAN and on file systems that are visible to each other.

Local copy mode

User-Supplied Options: OS and Path

When you generate a transfer package using the Local Copy mode, DCM uses a copy command to send the transfer package from the `generate_directory` to the `receive_directory` of the destination database. The Rational Synergy Distributed engine that generates the transfer package performs the copy.

The Local Copy mode is appropriate when the source database engine can write to the `receive_directory` of the destination database as user *ccm_root*. Such transfers are common between two DCM databases that can access the same file system.

DCM uses the **OS** field to determine which copy command to use. If the source database is UNIX-based, DCM uses the UNIX `cp -r` command. If the source database is Windows-based, DCM uses the Windows `copy` command for files and `xcopy` command for directories.

DCM uses the **Path** field to complete the path to the `receive_directory` of the destination database. In the Local Copy mode, DCM automatically appends the `dcm\receive` directory to the Path value.

Note Transfers that use Windows-to-UNIX Local Copy mode require that the UNIX file system that contains the destination database be visible to the Windows host. This is accomplished using a third-party NFS client package for Windows, or a gateway server solution such as Samba.

The following table shows examples of Local Copy mode path values for UNIX and Windows transfers.

Source	Destination	Destination database path
UNIX	UNIX	`/ccm71/ccmdbs/ESX`
UNIX	Windows	`/ccm71/ccmdbs/ESX1`
Windows	Windows	`\\`*`server_machine`*`\ccm71\ccmdbs\ESX2`
Windows	UNIX	`x:\ccm71\ccmdbs\ESX`

Remote copy mode

User-Supplied Options: Host Name, OS, and Path

When you generate a transfer package using the Remote Copy mode, DCM sends the transfer package from the `generate_directory` to the `receive_directory` of the remote destination database using an `rcp` command. The Rational Synergy Distributed engine that generates the transfer package performs the copy.

The Remote Copy mode is appropriate when the engine of the source database can log into the destination database machine and write to the `receive_directory` of the destination database as user *ccm_root*, but a common file system is not available.

DCM uses the **Host Name** field as the name of the destination database machine.

DCM uses the **OS** field internally. If the source database is UNIX-based, DCM uses the following `rcp` command:

```
rcp -r info_file preview_file path_to_data
host_name:receive_directory
```

If the source database is Windows-based, DCM uses the following `rcp` command:

```
rcp -b -r info_file preview_file path_to_data
host_name:receive_directory
```

Note An `rcp` service must be running on the Windows host for UNIX-to-Windows and Windows-to-Windows Remote Copy transfers. However, an `rcp` service is not shipped with Windows. Third-party commercial or shareware packages can be used to obtain `rcp` services. These packages usually support both `rcp` and `rsh` capabilities for Windows machines. Some packages support drive letter mappings in the path name, such as the `d:` in `d:\ccm71\ccmdbs\ESX`.

For more information, refer to the documentation for your `rcp` software package.

Specific third party `rcp` service packages are not warrantied or supported. You must make sure that such packages can be used to perform remote copies of files and directories between the source and destination hosts in your environment.

DCM uses the **Host Name** and **Path** fields to complete the path to the `receive_directory` of the destination database. In the Remote Copy mode, DCM automatically appends the `dcm\receive` directory to the Path value.

Note You should verify your settings and security by testing your `rcp` commands. Test the commands outside of DCM using your values for the **Host Name** and **Path** fields.

The following table gives examples of Remote Copy mode path values for UNIX and Windows transfers.

Source	Destination	Destination database path
UNIX	UNIX	`/ccm71/ccmdbs/ESX`
UNIX	Windows	`\\`*`server_machine`*`\ccm71\ccmdbs\ESX1`
Windows	Windows	`\\`*`server_machine`*`\ccm71\ccmdbs\ESX1`
Windows	UNIX	`/ccm71/ccmdbs/ESX`

File transfer protocol mode

User-Supplied Options: Host Name, OS, and Path

When you generate a transfer package using the File Transfer Protocol (`ftp`) mode, DCM sends the transfer package from the *`generate_directory`* to the `ftp` directory of the remote destination database using an `ftp` command. The Rational Synergy Distributed engine that generates the transfer package performs the copy.

The File Transfer Protocol mode is appropriate when the engine of the source database cannot access the file system of the destination database. The File Transfer Protocol mode permits non-authenticated users to transfer data to a specific directory at a secure facility using an anonymous login. This mode also lets authenticated users transfer data using their own login IDs.

DCM uses the **Host Name** field for the name of the `ftp` server. An `ftp` service must be running on the destination machine.

Note For `ftp` transfers from an Windows engine, Microsoft® provides the BackOffice internet suite for Windows 2000 Server. Third-party servers are also available. For more information, consult the documentation for your service.

DCM uses the **OS** field for internal purposes only.

DCM uses the transfer path to specify the path to the `ftp` directory. If a blank value is used for the transfer path, then the default path to the destination database *`receive_directory`* is assumed.

Note Do not initiate an `ftp` transfer from a Windows client connected to a UNIX server; this operation causes the transfer to fail. This is because the client cannot see the DCM transfer package located on the server file system.

Some sites have security restrictions. In these cases, you may only be able to send data to a staging area. You may specify this staging area as the *`receive_directory`*.

Note Manually try the `ftp` commands before using them with DCM. The following sections will help you do this.

FTP from a UNIX Engine

When you start an `ftp` transfer from a UNIX machine, the engine or the client process must be able to log into the destination machine without your intervention. This means that you must supply the host name, user name, and password login arguments to the `ftp` command. You accomplish this by entering the login arguments in the `.netrc` file on the source machine.

For example, the following `.netrc` file entries supply the login arguments for an anonymous transfer to the ESX `ftp` server:

```
machine ftp.ESX.com
login anonymous
password dcm_transfer
```

To enhance security, you can enter other login and password combinations in the `.netrc` file.

DCM first tries to perform the transfer from the engine using the `.netrc` file in the home directory of *ccm_root*. If the transfer from the engine fails, DCM attempts a transfer from the client using the `.netrc` file that is in the home directory of the person who is performing the transfer.

For a UNIX engine `ftp` transfer, the following `ftp` commands are executed using the Korn shell (`ksh -c`):

```
ftp @host < @infile
```

where `@infile` is replaced by a temporary constructed command file and `@host` is replaced by the **Host Name** field value.

The command file is constructed by first taking the contents of either:

- `<`*`database_path`*`>/etc/UNIXftp.in` file. By default, the file contains the single command:

  ```
  binary
  ```

- If the above file is not found: `<database_path>/dcm/UNIXftp.<dbid>` file, where `<dbid>` is the destination DCM database ID.

To the contents of the above file are added the following commands:

```
put <data_file> <data_file>
put <preview_file> <preview_file>
put <info_file> <info_file>
get <info_file> <temp_file>
quit
```

The last `get` command is performed so that DCM can compare the DCM information file with that retrieved from the remote machine. This is done because many ftp implementations return a zero success exit status code even if the transfer failed or the connection wasn't successfully established.

Note Some UNIX `ftp` servers do not support the "#" character used in transfer package file names. Therefore, DCM automatically converts these characters to "@" for UNIX-to-UNIX `ftp` transfers. The receive operation can handle transfer package file names of either format. Also, the `.netrc` file should be `rwx------` (that is, read/write/execute only by the owner).

FTP from a Windows engine

When you start an `ftp` transfer from a Windows machine, the engine or the client process must be able to log into the destination machine without your intervention. A built-in Windows `ftp` command supports the unattended `ftp` transfer.

The built-in command is:

```
ftp -n -s:@infile @host
```

where `@infile` is replaced by a temporary constructed command file and `@host` is replaced by the **Host Name** field value.

You can use a different `ftp` command by entering a line like the following in the `[Options]` section of your `ccm.ini` file:

```
DCM_NT_FTP_CMD=your_command
```

The command file is constructed by first taking the contents of either:

- `<database_path>/etc/UNIXftp.in` file. By default, the file contains the commands:

```
user anonymous dcm_transfer
binary
```

You can modify the user name and password in this file to change the login commands for all Windows `ftp` transfers from the database.

- If the above file is not found: `<database_path>/dcm/UNIXftp.<dbid>` file, where `<dbid>` is the destination DCM database ID. You can include login commands that specific to that destination database.

The following commands are added to the contents of the file:

```
put <data_file> <data_file>
put <preview_file> <preview_file>
put <info_file> <info_file>
get <info_file> <temp_file>
quit
```

The last `get` command is performed so that DCM can compare the DCM information file with that retrieved from the remote machine. This is done because many ftp implementations return a zero success exit status code even if the transfer failed or the connection wasn't successfully established.

Note Some UNIX `ftp` servers do not support the "#" character used in DCM transfer package file names. Therefore, DCM automatically converts these characters to "@" for Windows-to-UNIX `ftp` transfers. The receive operation supports transfer package file names in either format.

User defined mode

User-Supplied Options: Host Name, OS, and Path

The User Defined mode lets you use a script or batch file to perform transfers. This might be necessary, for example, to encrypt data before transferring it.

When you use this mode, DCM executes an external command script instead of an internal command. The script must be located in the `database_path\bin` directory, and it must be named `dcm_transfer` for UNIX engines and `dcm_transfer.bat` for Windows engines. The script reports an error by returning a non-zero value; a zero value means that all commands were successful.

The script accepts ten arguments, in the following order:

1. Destination database ID
2. Destination server OS (UNIX or Windows)
3. Destination server Host Name
4. Destination server CCM_HOME installation area
5. Source database ID

6. Destination database path
7. Transfer path
8. Path to DCM data file or directory
9. Path to DCM preview file
10. Path to DCM info file

The preceding arguments let you perform the automatic receive after the transfer is complete. To perform an automatic receive in a user-defined transfer script, turn ON the Automatic Receive toggle in the destination database definition, as explained in "Creating a DCM database definition" on page 41.

DCM performance

There are many factors that govern the performance of DCM. This section describes these factors and provides tips that can help you improve the performance of DCM.

Transfer sets

When a transfer set includes a direct member, sometimes the transfer set is marked as "needing recompute". Examples include:

- When a transfer set contains a project and its membership changes. For example, project membership can change during an Update Members, Use, or Unuse operation.
- When a transfer set includes a folder and its membership changes. For example, the task membership of a folder can change during an Update Folder, Add Task, or Remove Task operation.

When a generate operation is performed, if the transfer set needs a recompute, all of the direct members are expanded to determine all the indirect members. For a complex transfer set with many projects and folders, this can take a lot of processing time and database queries.

You can often achieve greater performance and efficiency using several transfer sets rather than one large one. Instead of having to recompute a large transfer set, you may only need to recompute a small one.

Transfer set definitions include the Fully expand reconfigure properties option. The default for this option is OFF. Typically, you will want to use the default setting because it results in a smaller transfer set indirect membership.

If the **Fully Expand Reconfigure Properties** option is set to ON, the transfer set expands the update properties of all projects that are members of that transfer set. Moreover, all associated objects of included tasks also become part of the transfer set. This often results in a large transfer set membership, and a recompute of that membership can take a long time.

Local databases

When the destination database is local, you can achieve better performance using the appropriate database definition options.

For a destination database that is visible over the local file system:

- Use the Direct mode transfer mode. This causes the transfer package to be generated directly into the transfer path of the destination database without zipping the package.

For a destination database that is **not** visible over the local file system, but has a high-speed network connection and is on a trusted host (that is, you can use the `rcp` and `rsh` commands without entering a password):

- Use unzipped packages and the Remote Copy transfer mode. You may also improve performance if you specify a generate_directory for the transfer set that is a local file system rather than an NFS file system.

Remote databases

If the destination database is **not** on a trusted host (that is, you cannot use the `rcp` and `rsh` commands without entering a password), or if there is a slow network connection to the destination host:

- Use the zipped packages, and use the file transfer protocol, user defined, or manual copy transfer mode.

Location of transfer packages

DCM allows two paths to be specified for a given destination database definition:

- The **Path** is the path to the destination database.
- The **Transfer Path** is the location where DCM will place transfer packages that are generated for the destination database.

If the **Transfer Path** is blank, which is the default when adding a new destination database, then the default transfer path is the `dcm\receive` directory of the destination database.

For more information on how DCM writes transfer package files to the file system of the destination database, refer to "Destination database" on page 25.

How to use automatic receive

An *automatic receive* is an operation you start at the source database as part of the generate operation. Automatic receive lets you perform a generate, transfer, and receive using a single operation from either the GUI or command line.

When you perform a generate operation with an automatic receive, DCM uses the `ccm_receive` command to start the receive on the destination machine. This command starts a Rational Synergy Distributed session, passes the arguments to the DCM receive operation, and then stops the session.

DCM performs automatic receives as user *ccm_root*. This occurs because the Rational Synergy Distributed engine process performs the receive, and the engine process runs as *ccm_root*. Moreover, the *ccm_root* user has the *ccm_admin* role; this role is needed to change any non-modifiable objects when DCM loads the data.

The following sections explain how to set up source and destination databases to enable automatic receives. Before selecting an Automatic Receive option in a destination database definition, be sure to read the section(s) that apply to the platform(s) you are using.

UNIX-to-UNIX automatic receive

For UNIX-to-UNIX transfers, DCM uses `ccm_remexec` on the source machine to launch `ccm_receive` on the destination machine.

For receives on the current machine

If the Host Name in the destination database definition matches the name of the source (current) machine, then the generate, transfer, and receive operations are done on the current machine. In such cases, DCM executes `ccm_receive` on the current machine without using `ccm_remexec`.

If the source and destination database host names are the same, but DCM attempts to execute `ccm_remexec` anyway, use the long host name in the **Host Name** field. For example, instead of setting the Host Name to `mymachine`, set it to `mymachine.mydomain.com`.

For receives on a different machine

To perform a UNIX-to-UNIX automatic receive on different source and destination machines, the following conditions must be met:

- DCM must be able to log into the destination machine as user *ccm_root*, because `ccm_receive` must be executed as *ccm_root*.

- You must set up the `.rhosts` file on the source machine to allow the *ccm_root* user to log into the destination machine without a password. Otherwise, the receive will not be automatic because the system will request a password.

Windows-to-Windows automatic receive

For Windows-to-Windows transfers, DCM uses `ccm_rem` on the source machine to access `ccm_remd` on the destination machine. The `ccm_remd` application then launches `ccm_receive` on the destination machine.

For receives on the current machine

If the Host Name in the destination database definition matches the name of the source (current) machine, then the generate, transfer, and receive operations are done on the current machine. In such cases, DCM executes `ccm_receive` on the current machine without using `ccm_rem`. The session initiated by `ccm_receive` is started by user *ccm_root*, but you do not need to log into the current machine as *ccm_root*.

If the source and destination database host names are the same, but DCM attempts to execute `ccm_rem` anyway, use the long host name in the **Host Name** field. For example, rather than setting the Host Name to `mymachine`, set it to `mymachine.mydomain.com`.

For receives on a different machine

To perform a Windows-to-Windows automatic receive, you must set up the destination machine as follows:

1. You must be logged into Windows as user *ccm_root*. Otherwise, the receive fails, and the following error message is displayed: `User is not admin user (ccm_root)`.
2. The `ccm_remd` application must be running on the destination machine. Otherwise, the receive fails, and the following error message is displayed: `Cannot establish connection`.

UNIX-to-Windows receive

The UNIX-to-Windows automatic receive is not supported at this time. However, an alternative method is to set up a scheduled job using the Windows `at` command at the destination database. Using the `at` command, the automatic receive can be performed as often as needed.

To schedule a receive at the destination database, use the `at` command with the following `ccm_receive` command:

```
ccm_receive -h engine_host -d destination_database_path /
dbid source_database_ID -ts transfer_set_name /ccm_home
Windows_CCM_install_area -dir receive_directory
```

You can omit the `/ts` option if you want to receive all transfer sets from the specified database. You can omit the `-dir` option if you want to use the default *`receive_directory`*.

Windows-to-UNIX receive

The Windows-to-UNIX automatic receive is not supported at this time. However, an alternative method is to set up a `cron` job, at the destination database, that executes the `ccm_receive` command.

Missing transfer sets

When receiving a transfer package, the last generate time in the package information file is compared with the generate time of the last received package. If they do not match, it is probably because a transfer package is missing. In such cases, read the DCM warnings, and do one of the following:

- Ignore the missing package — this may result in subsequent errors, as explained in "Missing transfer packages" on page 147.

  OR

- Discard the one that you are trying to receive and regenerate the transfer package.

Note The generate times of received packages are kept in the `*database_path*\dcm\receive\history` directory. Do not delete this directory or its contents. Otherwise, you may get DCM warnings when you receive transfer packages.

Receive failures

Reasons for failure

The following checks are made prior to extracting transfer packages:

- Database parameters must be in DCM databases (see “Establishing common database parameters” on page 34). DCM verifies that critical parameters are consistent between the source and destination databases. If DCM discovers an inconsistency, the check fails.
- The source and destination databases must use the same DCM delimiter. Otherwise, transfer packages generated from the source database will not be recognized by the destination database. The DCM manager must ensure that the same delimiter is used throughout the DCM cluster.
- The source and destination databases must use the same release delimiter.
- Compatible versions of Rational Synergy Distributed must be used by the source and destination databases. If this check fails, DCM displays an error message that lists the acceptable version(s).

Caution If this check fails, and you are receiving a package generated from a newer release than the one you have, you may need to install a patch that lets the package be received. In any event, contact Rational Synergy Distributed Support for assistance. Otherwise, crucial data conversions may not be performed.

- Case settings of the source and destination databases must be compatible. A transfer package from a database set to use Lower Case can be received into a database set to Preserve Case. However, a transfer package from a database set to Preserve Case will be incompatible with a database set to use Lower Case if any of the object names contain upper-case characters.

  Note An incompatibility message can be ignored if the transfer package contains **only** lower case filenames.

- The version delimiters of the source and destination databases must be identical. If they are not the same, the transfer package may contain objects whose names have the version delimiter of the destination database. Such objects will not be accessible to users.

Caution If this error message is received, proceed only if you are sure that users will be able to access the objects.

Missing transfer packages

When receiving transfer packages, DCM checks for missing packages. If a missing package is reported, contact the DCM manager at the source database to determine what happened. Typically the safest course of action is to delete the transfer package, and have the DCM manager at the source database generate a new one that has an earlier last generated time.

Caution If this error is ignored, it may result in empty directory entries or missing relationships.

Missing type definitions

By default, DCM includes all user-defined type definitions in transfer packages. During a receive operation, DCM automatically creates, in the destination database, any user-defined type definitions that are not already defined in the destination database.

If a transfer set excludes type definitions, then the transfer package may contain objects whose types are not defined in the destination database. If DCM finds missing type definitions, it does not allow the receive operation to be performed. In such cases, do one of the following: have a user with the *type_developer* role create the missing type definitions, or delete the transfer package, and have the DCM manager in the source database regenerate it without excluding type definitions.

Object name conflicts

Each object in a DCM cluster has a unique cluster ID attribute - for more information, see “Cluster IDs” on page 149. If an object in a DCM transfer package has the same four-part objectname as an existing object in the receiving database, but has a different cluster ID value, this is reported as an object name conflict. In effect, there are two objects with the same objectname that could be totally unrelated to each other.

An object name conflict can arise under a number of conditions:

1. An object is created in a source database and sent to a destination database. It is deleted from the source database. Later, another object is created in the source database that has the same four-part objectname and then sent to the destination database. The most common case for this occurs with projects in *prep* state.
2. An object is created in a source database and sent to a destination database. The source database is restored from a backup that was taken before that

object existed. Later, another object is created in the source database that has the same four-part objectname and then sent to the destination database.

3. An object is created in a source database and sent to a destination database. The object is renamed in the source database and a new object is created having the former four-part objectname of the object. The new object is replicated without the renamed former object to the database. DCM will automatically rename objects but only if the renamed object is replicated to the destination database.

The first step on receiving an object name conflict is to check the object in both the source and destination databases. Especially compare the create time, owner and for files, the source contents.

In cases (1) and (2), if both objects are required, then change the version attribute of the newer object, remove the transfer package, regenerate it and rerun the receive. In case (3), either rename the object in the destination database, or remove the package, ensure that the renamed object is in the transfer set, regenerate the transfer package and rerun the receive.

If both objects should be the same, you should delete the old object from the database and then rerun the receive. Note that the import process used by DCM receive never attempts to update the source of a static object that has previously been received. If you just delete the `cluster_id` attribute and rerun the receive and the source attribute between the two objects is different, **the receive will not update it to the new contents**.

Cluster IDs

All objects in a DCM database are automatically assigned a cluster ID. This attribute uniquely identifies an object across all Rational Synergy Distributed repositories, and is directly related to the database in which the object was created. The purpose of this attribute is to:

- Track name changes across databases.
- Check for object name conflicts - different objects having the same four-part objectname.

When an object is imported, DCM checks its name and cluster ID. This check identifies objects that have been renamed. For example, suppose a static object is sent to another database, renamed, and sent back to that other database. In such cases, the cluster ID is used to detect the name change. Thus, the destination database ends up with one occurrence of the renamed object.

Type definitions

In order for DCM to receive an object of a specific type, a definition for that type must exist in the destination database. By default, DCM includes all user-defined type definitions in transfer packages. During a receive operation, DCM automatically creates, in the destination database, any user-defined type definitions that are not already defined in that database.

Some methodologies require a local administrator to define all type definitions. In such cases, automatic type definition replication can be suppressed by excluding type definitions from the transfer set (see “An optional types part, containing information about type definitions.” on page 49). If a type definition in the transfer package is not defined in the destination database, the receive operation is abandoned, and an error message is displayed.

By default, after a DCM receive operation successfully completes, the type definitions contained in the transfer package are removed from the file system of the receiving database. However, it is sometimes desirable to keep the type definitions. For instance, if type definitions are kept, the `ccm typedef -import -force` command can be used to maintain all user-defined type definitions in a central database.

The DCM setting Keep Type Definition after Receive controls whether the type definition data is kept after a successful DCM receive. The default value is `FALSE`, which means that the data is removed from the file system of the receiving database. A value of `TRUE` means that the data is kept under the *types* receive directory, which is described as follows:

The transfer package is extracted under the *`receive_directory`\`package_ident`* path, where *`package_ident`* is made up of the database, transfer set number, and time value (for example, `M#22#UM#962984701`). The `types` directory is located under the *`package_ident`* directory, as is the `data` directory for the exported object.

Note When the type definition data is no longer needed, remove all files from the `types` and `data` receive directories to free disk space.

Parallel checking

In parallel development, it is important to merge parallel changes made to the same object that are to be included in a release. In a single database environment, users may receive warnings about parallel versions at the time an object is checked out or checked in. In a distributed environment, the parallel version may be in a different database and, therefore, may not be visible at the time it is checked out or checked in. It is therefore not possible to disable parallel development in a distributed database environment.

By default, during a receive operation, DCM checks for parallel versions on each newly-created object received in the transfer package (except projects and products) if the package was generated using the predefined **Entire database** transfer set or if that object was a history member. Note that objects that were not history members and where a user-defined transfer set was used will not be checked for parallels. This is because there is no guarantee that the history of the object is complete, and a disjointed history would give rise to many false reports of parallels.

This behavior can be changed to include all received objects (except project and product objects) using the DCM setting Parallel Checking. Details of these parallel versions are included in any DCM receive email that is sent to the recipient(s) defined in the transfer set.

DCM can also send parallel email notifications directly to developers who own or created such parallel versions. If the **Local Parallel Notification** option in the transfer set of the source database is enabled, an email is sent (during parallel checking) to each user who is the owner of a parallel version. The email lists each received object and its parallel version(s) that relate to the user. Again, this feature only applies when using the **Entire database** transfer set or for objects that were history members of the transfer set.

The introductory text of the email is defined in the `*database_path*/etc/dcm_local_parallel_intro.txt` file. This text contains the keyword `%database`. When an email is sent, this keyword is replaced with the full path to the destination database.

If you are not using the **Entire database** transfer set and not using history members, then parallel checking on DCM receive will not be performed. This might be the case, for example, if you are using a user-defined transfer set and adding folders to the transfer set as a means of replicating tasks for a release. In such cases, you should perform separate checks for parallels on a regular basis using conflict detection (the **ccm conflicts** command, or the **Detect Membership Conflicts** operation in Rational Synergy Distributed (not Synergy Classic).

How DCM stores generate times

By default, DCM saves 50 generate times, of which 20 are "old" times. The number of generate times that DCM stores is defined by the DCM setting No. of Generate Times. The number of "old" generate times is defined by the DCM setting No. of Old Generate Times.

Based on the aforementioned default values, DCM saves the 30 most recent times at which a transfer package is generated. This value is calculated as follows: 50 - 20 = 30. That is, the No. of Generate Times minus No. of Old Generate Times equals 30.

DCM keeps track of these 30 times in the order in which the generate operations are performed. If the user does not regenerate a transfer package from an earlier time, the list of generate times stored by DCM is chronological, with the most recent time being first.

DCM keeps track of the 20 "old" generate times based on the value of the DCM setting Old Generate Times Resolution. This parameter defines the interval between the "old" generate times. The default value is one (1.0) day.

For example, suppose a generate operation is performed every hour on the hour. In addition, the default values for the following parameters are used: No. of Generate Times, No. of Old Generate Times, and Old Generate Times Resolution.

Given the above scenario, DCM would eventually store the 30 most recent generate times. DCM would also store 20 "old" generate times that are at least one (1.0) day apart. Thus, you could regenerate a lost transfer package from 20 days ago.

Work area handling on DCM receive

When DCM receives a project that does not currently exist in the receiving database, by default it creates the project with the same work area properties as in the source database. An exception to this is that the work area path is not replicated. This is because that work area path might not exist or be valid at the receiving database, or if the database is on the same LAN, it might attempt to use the same work area path as used by same project in the source database, which would always fail. Instead, if the project being received has work area maintenance enabled, it is given a new default work area path based on the current user's default work area path template.

If a work area cannot be created for a new received project, DCM will automatically disable work area maintenance for the project and attempt to create the project without a maintained work area. DCM will also include a warning that it has done so in the DCM Event Log and any DCM receive email that is sent. The reason for such action is that it allows the DCM receive to proceed and to populate the project with its members, after which a user can correct the work area problem without having to re-run a failed DCM receive.

The default behavior may not always give desired results:

- If the default project path is on a file system with little free space, that space can be consumed by newly received projects. This is especially true with copy-based work areas.
- The project might be for a different client than the one where it is received. For example, a project for a Windows application received on a UNIX client has a default work area located on a UNIX file system. Later, a Windows client user will have to change the work area path to a Windows file system location.
- For *prep* projects, the default work area path may not be visible or modifiable by the build manager who needs to build from it.

Depending on your needs, you may choose to do one of the following:

- Change the work area template defined in the `ccm.ini` file used for the receive, as follows: Under the `[Options]` section, change the `wa_path_template` option to a more suitable work area path.

  OR

- Disable work area maintenance for all received projects (this action requires the *ccm_admin* role). Later, a user can define an appropriate work area path. See "Ignore Work Area Maintenance" on page 160 for further details.

When DCM receives a project that already exists in the receiving database, none of the work area properties of the project are changed. After creation, the work area properties of the project are considered to be local settings. This allows a user to change work area properties, such as relative or absolute work areas, without fear of that change being overwritten on a subsequent DCM receive.

Work area subdirectory templates and paths

Projects created in different databases might use the same project name and version. Each project must use a different work area path if it is to be successfully maintained by Rational Synergy Distributed. For this reason, by default, Rational Synergy Distributed uses a project subdirectory template as follows:

`%project_name%optional_project_instance%delimiter%project_version`

The `%optional_project_instance` keyword is expanded to an empty string for local projects using an instance value of `dbid#1`. For non-local projects or the second or further instances of local projects, it expands to the DCM delimiter and the instance value.

In the following table, in database A, projects might have the following expanded subdirectory templates that are used in the final work area path for the projects.

Project objectname	Expanded subdirectory template
`local-1:project:A#1`	`local-1`
`local-1:project:A#2`	`local#A#2-1`
`nonlocal-1:project:B#1`	`nonlocal#B#1-1`
`nonlocal-1:project:C#1`	`nonlocal#C#1-1`

Note that changing a DCM database ID or changing the DCM delimiter may therefore have an impact on existing work area paths.

Object histories in transfer sets

By default, when an object is added to a transfer set, it is added without history. The default can be changed in one of two ways:

- Define a preference in your personal `ccm.ini` file under the `[Options]` section:

  ```
  dcm_dflt_add_history = TRUE
  ```

  OR

- Define a database-wide default value for users (this action requires the *ccm_admin* role). This is performed using the **DCM Settings** dialog or `dcm` command. See “Default Add History” on page 159 for more details.

If the `dcm_dflt_add_history` is not set in a user’s `ccm.ini` file, then the default value is determined by the DCM Setting Default Add History.

Supported transfers from Rational Synergy Distributed releases prior to 7.1

In Rational Synergy Distributed 7.1, DCM supports transfers with databases of 6.4 SP1, 6.5 SP2, 6.5a, 6.6a, 7.0 and 7.1. Databases prior to 7.1 must be patched with the DCM compatibility patch. Rational Synergy Distributed 7.1 introduces a component task feature that is not supported in prior releases. Please see the 7.1 Readme for details.

If you have earlier databases (such as 6.1, 6.2, 6.3, or 6.3a) with which you wish to exchange data, you should either:

- Upgrade those databases to Rational Synergy Distributed 6.4 or later
- Replicate data via a hub database that is at release 6.4 SP1 or 6.5 SP2

Note that the CCM45SP2 export format option and the map project instances option were retired in Rational Synergy Distributed 6.5.

DCM in Rational Synergy Distributed 7.1 supports replication of processes, process rules, folder templates and release definitions with databases that are at 6.5 or later. The full functionality of the process definitions and process rules will only be available when replicating with other 6.5 or later databases. It is therefore advisable to update master and satellite databases used for the development of a component within a short time. Spoke databases that replicate through a hub database at an earlier release will have reduced functionality until the hub database has been upgraded.

When replicating from a Rational Synergy Distributed 7.1 database to a pre-6.5 Rational Synergy Distributed database:

- Process definitions and process rules will not be replicated. Process definitions were a new feature in Synergy Distributed 6.5 and there is no equivalent functionality in prior releases. Process rules were significantly redesigned for 6.5 for use with processes and are a new type of object that supersedes the old reconfigure/update templates.
- Folder templates can be replicated.
- Queries on query-based folders and folder templates will always appear as **Custom** queries in Synergy Classic in earlier releases.
- A preview of a DCM transfer package will show date/time values as integer time values rather than date strings, and names will no longer be truncated. This is because the preview file format has been changed to be locale-neutral and more client-neutral.

When replicating from a Rational Synergy Distributed pre-6.5 database to a Rational Synergy Distributed 7.1 database:

- Reconfigure/update templates will be ignored. The *recon_temp* type no longer exists in Rational Synergy Distributed 7.0 and later and has been replaced by a more powerful and flexible process and process rule design.
- Folder templates can be replicated.
- Queries on query-based folders and folder templates will always appear as **Custom** queries in Synergy Classic.
- When a release definition is received, if no process rule currently exists for one of the valid purposes of the release, then a release-specific process rule will be automatically created for that release and purpose. This could have different settings from any reconfigure template used in the older Rational Synergy Distributed database.

DCM settings

The **DCM Settings** dialog and the `ccm dcm -settings` command allows you to change the most common DCM settings that affect the operation of DCM in the current database. The table below summarizes the DCM settings available.

DCM Setting	**Default Value**	**Description**
Description	Blank string	Description of the current database. This can be specified on DCM Initialize.
Location	Blank string	Free-format text description of the geographic location for the current database. This can be specified on DCM Initialize.
Admin Info	Blank string	Free-format text details of the names and contact details of the DCM administrators of the current database. This can be specified on DCM Initialize.
Event Log Size	100	Maximum number of entries allowed in the DCM Event Log - see "DCM event log" on page 161.
Default Add History	FALSE	Determines the default setting for history when an object is added to a transfer set.
Default Include Associated Baselines	FALSE	Determines the default setting for the Include Associated Baselines setting when a new transfer set is created.
No. of Generate Times	50	Specifies the maximum number of last generated times for any transfer set and destination database pair - see "How DCM stores generate times" on page 152.
No. of Old Generate Times	20	Specifies the maximum number of last generated times for any transfer set and destination database pair that are at least Old Generate Time Resolution days apart - see "How DCM stores generate times" on page 152.

DCM Setting	Default Value	Description
Old Generate Time Resolution	1	Specifies the time resolution for which old generate times will be kept - see "How DCM stores generate times" on page 152.
Ignore Work Area Maintenance	FALSE	Controls whether projects that had a maintained work area in the source database should be created on receive with work area maintenance enabled - see "Work area handling on DCM receive" on page 153.
Update Release Definitions	active	Controls what actions, if any, are taken for release definitions on DCM receive - see "Replication of generic and release-specific processes" on page 118.
Update DCM Database Definitions	TRUE	Controls whether DCM database definitions are automatically created or updated on DCM receive - see "Replication of DCM database definitions" on page 123.
Update Reconfigure and Folder Templates	TRUE	Controls whether processes, process rules and folder templates in a transfer package should be processed on a DCM receive - see "Replication of generic and release-specific processes" on page 118.
Parallel Checking	created	Controls what parallel checking, if any, is performed on a DCM receive - see "Parallel checking" on page 151.
Keep Type Definitions after Receive	FALSE	Controls whether type definitions in a DCM transfer package should be left in the receive_directory after the DCM receive operation has completed.
Receive Control Transitions	empty list	Controls which transitions, if any, may be applied from a non-controlling database to the controlling database for an object - see "Transitions to a controlling database" on page 117.

DCM event log

The DCM Event Log captures details of the following operations:

- DCM initialize and reinitialize
- DCM database ID changes
- DCM delimiter changes
- DCM generate
- DCM transfer
- DCM receive
- Creation of a DCM package from Save Offline and Delete feature

By default, the event log can record up to 100 events. When the event log reaches the maximum size, the oldest event is deleted to make way for the next new event. You can increase the maximum DCM Event Log size through the **DCM Settings** dialog or `dcm` command.

For each event, DCM captures two types of information:

- Event Information - this is a summary of the event details. For DCM generate, transfer, and receive operations, this is the same information that would be included in any email that would be sent.
- Messages - this captures all messages that were written to the Message View or log file for that operation. In the event of a failed operation, you should review such messages for further information about the failure.

Broadcast database and packages

By default, DCM creates a predefined database definition for the **Any** broadcast database. The broadcast database definition does not represent any single real database. Its purpose is to allow you to create a DCM transfer package that may be received in any database. This may be useful when publishing a released application or component. You can generate a DCM transfer package and place the package at a well known location such as on an FTP server. Other users can download the package and receive it into their database.

Glossary

associated_cv	*associated_cv* is a relationship that indicates the specific object version(s) that are associated with a task.
associated_task	*associated_task* is a relationship that indicates the task(s) that are associated with a change request.
case sensitivity	*case sensitivity* differentiates between lowercase and uppercase characters.
ccm_admin	The *ccm_admin* role is needed for any operation that might change the properties of non-modifiable objects. These operations include initializing a database to use DCM, receiving a transfer set, and so on.
change request	A *change request* is a problem or issue that is entered using either: Rational Change — the standard change tracking tool, or Distributed Rational Change (DCS) — the distributed change request tracking package.
change request object	A *change request object* is a database object that describes a change request. Change request attributes include why and when the change request was made, and who submitted the request.
check in	An operation used to make a developer's object version available to other users.
check out	An operation that creates a new version of an object from an existing version that is stored in a Rational Synergy database. Developers check out objects so they can work on them.
child change request	A *child change request* is a change request that has a parent (see parent change request).
cluster ID	A *cluster ID* is an attribute assigned to each object in a DCM database. This attribute uniquely identifies an object across all Rational Synergy repositories, and is directly

	related to the database in which the object was created. The purpose of this attribute is to track name changes across databases.
controlling database	The *controlling database* for an object is the database that controls the object and is the database in which modifications may be performed. In all other databases, the object will not be modifiable.
created_in	The *created_in* attribute indicates the database in which a specific object version is created.
current task	The *current task* is the task on which you are currently working; this term is used in the Rational Synergy interface. See also, *default task*.
database ID	A *database ID* (`dbid`) is a character string that uniquely identifies each database in a DCM cluster. It is manually assigned to a database during the DCM initialization set-up process. The database ID is used in the `version` and `instance` attributes of DCM objects. This guarantees that object names are unique within a DCM cluster.
dbid	See *database ID*.
DCM	DCM refers to the Rational Synergy Distributed configuration management tool feature. DCM lets multiple Rational Synergy databases share objects that originate, and are being changed in parallel, in any Rational Synergy database in the world that has been initialized to use DCM.
DCM administrator	A *DCM administrator* is any person who performs DCM operations. The role required for most DCM operations is *dcm_mgr;* the *ccm_admin* role is only required to perform a DCM initialization or a receive operation.
DCM cluster	A *DCM cluster* is a collection of databases that make up a shared development environment and communicate with one another in accordance with a DCM methodology.
DCM database	A *DCM database* is any Rational Synergy database that has been initialized to use DCM.

DCM delimiter

A *DCM delimiter* is the character that is used: 1) Between the database ID and object instance in object references, 2) Between the database ID and object version in imported objects that have been checked out in a destination database, and 3) Between the database ID and task number in some dialogs where task numbers are displayed.

The default DCM delimiter is "#". The following characters can also be used: "!", "~", or "=".

The DCM delimiter must be the same in all databases in a given DCM cluster.

DCM methodology

A *DCM methodology* is the plan that defines what information is to replicated between which database(s), for what purpose, and how the software component or application is built from such shared changes. Some typical DCM methodologies include: Master and Satellite, Publish and Subscribe, and Peer-to-Peer.

dcm_mgr

The *dcm_mgr* role is required for the following operations: defining destination databases, defining transfer sets, adding objects to transfer sets, and generating a transfer package.

DCS

An abbreviation for Distributed Rational Change.

default task

The *default task* is the task on which you are currently working; this term is used in the Synergy Classic interface. See also, *current task*.

destination database

A *destination database* is any database that receives a transfer package.

destination database definition

A *destination database definition* is the set of parameters that name, describe, and provide transfer information for a database to which a set of changes can be sent. DCM uses this definition combined with a transfer set to generate a transfer package.

direct history

When a direct member is added with history, its member type is *direct history.*

direct member	A *direct member* of a transfer set is an object that is explicitly added using a DCM add operation. Only direct members can be explicitly removed from a transfer set. When a direct member is deleted, all of its indirect members are also removed (unless they are being used elsewhere in the transfer set). If an object is added with history, all of its predecessors and successors are added as direct members. Therefore, if an object that was added with history is deleted, all versions of that object are also removed.
dynamic role switching	*dynamic role switching* automatically changes roles if a user attempts to perform a DCM operation while in the wrong role for that operation, but has access to the correct role in the database. When the operation is completed, DCM switches back to the original role.
entire database transfer set	The *entire database transfer set* is a built-in transfer set that includes all eligible objects in the database.
from	The **From** field in the **Properties** dialog shows the `dbid` of the database in which an object version was created.
generate	The *generate* operation creates a transfer package and sends it to a destination database. The transfer package is automatically sent unless the Manual copy transfer mode is being used. If Automatic Receive has been selected, DCM automatically receives the transfer package at the destination database.
generate_directory	The *`generate_directory`* is the where the DCM generate operation writes temporary and transfer package files. By default, DCM puts these files in the *`database_path`*`\dcm\generate` directory of the source database. However, you may specify an alternate directory when you create a transfer set.
handover of control	*Handover of control* occurs when controlling database of an object is changed from the current database to some other

	database.
home database	The *home database* is the DCM database in which an object version was either initialized or created. The home database ID of an object is the same as its Local To property, and remains with the object wherever it is sent.
hub database	A *hub* is a database that is responsible for the transfer of shared data to all of the databases in a DCM cluster. The hub also receives changes to the shared data from other databases.
indirect history	When a direct member is added to a transfer set with history, its indirect members are added as indirect history members.
indirect member	An *indirect member* of a transfer set is an object that DCM includes because it is associated with an object that has been explicitly added using the DCM Add operation. Indirect members cannot be explicitly removed from a transfer set. When a folder is added to a transfer set, DCM includes folder member tasks and all objects associated with the tasks as indirect members. When a task is added to a transfer set, DCM includes as indirect members all objects associated with the task. When a project is added to a transfer set, DCM includes as indirect members: -- All member objects of a project -- Any baseline that the project is using -- All folders in the update properties of a project -- All tasks in the update properties. When a source object is added to a transfer set, DCM includes as indirect members any tasks associated with the object.
local to	The **Local To** field in the **Properties** dialog shows the ID of the database in which the initial version of an object was created. The Local To database ID is associated with the

	object throughout its history and does not change.
master database	The *master database* is a database that is responsible for updating all of the databases in a DCM cluster. No database is inherently a master database. A database is designated as a master when a master/node DCM methodology is defined (such as Master and Satellite).
master build	A *master build* is a build that combines the changes that have been made in all databases in a DCM cluster.
master integration project	A *master integration project* is a project that is used to build and test the combined changes that have been made in all databases in a DCM cluster.
master folder	A *master folder* is a folder used to collect changes from all databases in a DCM cluster. The master folder is included in the update properties of a master *prep* project.
modifiable_in	*modifiable_in* is an object attribute that indicates the database that has control over that object. This attribute protects objects from being overwritten in their controlling database by data from other databases. For example, if an object has a `modifiable_in` attribute that is set to a `dbid` other than the current `dbid`, then it is not DCM-modifiable. Moreover, if an object does not have a `modifiable_in` attribute, it is only DCM-modifiable if its `created_in` attribute is set to the current `dbid`.
node database	A *node database* is a database that accepts updates from a master database in a DCM cluster. No database is inherently a node database. A database is designated as a node when a master/node DCM methodology is defined (such as Master and Satellite).
object name conflict	An *object name conflict* is when two different objects have the same four-part objectname.
parent change request	A *parent change request* is a logical grouping of other change requests. For example, a change request might require fixes in several databases. One way to do this is to break up the change request into one or more change requests (child

	change requests) and distribute the child change requests to the appropriate databases where the fixes will be made.
peer	A *peer* is a database that controls its own flow of information to each database in a DCM cluster, and maintains its own *prep* projects. A peer database can transfer or receive data from any database in a DCM cluster.
point-to-point	With the *Point-to-Point* replication topology, each database generates a transfer package for either a database to which information is to be sent, and/or a database to which control of a change request is to be handed over.
publisher	A *publisher* is a database that makes shared data available to all databases in a DCM cluster. The publisher sends data to destination databases called *subscribers*.
receive	The *receive* operation incorporates into a database new object versions that were sent in a transfer package. A transfer package can be received manually at its destination database, or automatically as part of the generate operation.
receive database list	A *receive database list* is the list of databases from which the current database has received a transfer package.
receive_directory	The *receive_directory* is the where the DCM transfer and receive operations write transfer package files. By default, DCM puts these files in the `database_path\dcm\receive` directory of the destination database. However, you may specify a path other than the default. The path you specify must be visible to and writable by the engine process.
regular transfer	A *regular transfer* is any transfer of a specific transfer package that is sent after its initial transfer. It typically contains fewer objects than the first transfer because only changes are sent in a regular transfer.
replication topology	A *replication topology* defines whether databases replicate data directly with each other, or through intermediate hub

	databases. Two typical replication topologies include Point-to-Point and Hub and Spoke.
satellite database	A *satellite* is a database that receives updates to shared data from the central database, or *master*, and sends its updates to the master.
send	See **transfer**.
source database	A *source database* is any database that sends a transfer package to another database.
source object	A *source object* is an ascii or binary object version, such as a C source file or executable.
spoke database	A *spoke* is a database that receives updates to shared data from the central database, or *hub*, and sends its updates to the hub.
subscriber	A *subscriber* is a database that requests and receives shared data from a central database called the *publisher*.
transfer	The *transfer* operation moves a transfer package from its source database to a destination database. The transfer occurs automatically when the transfer package is generated (unless Manual copy of the transfer package is being performed).
transfer list	A *transfer list* is a list of objects that will be sent to a destination database in a transfer package, as computed by the generate operation.
transfer mode	The *transfer mode* is the method that is used to send a transfer package to a destination database. The transfer mode is selected for a database when the database is added to a destination database list. The transfer mode can be changed using the **Generate** dialog. There are five possible transfer modes: Manual Copy, Local Copy, Remote Copy, File Transfer Protocol, and User Defined. Note that when using the Manual Copy mode, you may

	find it more convenient to put transfer packages in a directory other than the default directory (that is, `*database_path*\dcm\generate)`. If so, you can specify different a *`generate_directory`*.
transfer package	A *transfer package* is a set of files that is generated from a transfer set and destination database definition pair at the source database. The transfer package is sent to a destination database and includes: a DCM information file, a preview file, and a data file that contains the objects being transferred.
transfer set	A *transfer set* is a related group of objects that is eligible to be sent to a destination database. DCM pairs the transfer set with a destination database definition to generate a transfer package.
with history	*with history* means transferring objects and their predecessors.

Appendix: Notices

Trademarks

Index