Rational. Synergy

IBM

Introduction to Synergy

Introduction to Rational Synergy

Release 7.1

Before using this information, be sure to read the general information under Appendix A, "Notices" on page 55.

This edition applies to VERSION 7.1, Rational Synergy (product number MN-SCM-IV-ICM70-08-01) and to all subsequent releases and modifications until otherwise indicated in new editions.

Table of Contents

Chapter 1: Introduction 1

Transitioning from other tools . . . 1
Conventions . . . 2
Rational Synergy graphical user interfaces . . . 2
Rational Synergy command line interface . . . 2
Typefaces and symbols . . . 2
Contacting IBM Rational Software Support . . . 3
Prerequisites . . . 3
Submitting problems . . . 3

Chapter 2: Benefits of using Rational Synergy 7

Goals of a configuration management tool . . . 7
Rational Synergy benefits . . . 8
Easy to use, right out of the box . . . 8
Fast start-up . . . 8
Rapid productivity for new users . . . 9
Flexible, automated workflow . . . 9
Secure team engineering environment . . . 10
World-wide control and transfer of information . . . 10
Seamless integrations for Windows development . . . 12

Chapter 3: Rational Synergy terminology 13

The Rational Synergy database . . . 13
Tasks and objects . . . 13
More about objects . . . 16
The check out and check in operations . . . 16
History . . . 18
Properties . . . 18
Current task . . . 18
Users, lifecycles, and states . . . 19

Projects and project groupings 20
Directories and candidates 22
The work area 22
The synchronize operation 23
Using, creating, adding, deleting, or removing objects 23
Update, baseline, tasks, and process rules 24
Grouping tasks in folders 25
The build process, products, and makefiles 26

Chapter 4: Rational Synergy methodology 27

Task-based methodology. 27
Users. 28
Projects and workflow. 28
Release. 30
Project purpose. 32
Update properties 33
The default workflow of Rational Synergy. 34
The use of tasks 34
The development process 35
The integration test cycle 35
The system test cycle 37
Releasing the system test baseline 38
Preparing for the next release 38
Summary 39
Parallel development 41
Parallel concurrent development 42
Parallel platform development. 42
Parallel release development 43
Component-based development. 43
Managing components 44
Publishing components 44
Referencing components 45
Process patterns 45

Chapter 5: Terms and concepts 47

Appendix A: Notices 55

Trademarks 58

Index 59

1 Introduction

IBM® Rational® Synergy is a comprehensive, feature-rich configuration-management system that gives software development teams control over software, document development, and maintenance activities. Rational Synergy supports medium-to-large development teams working in heterogeneous, distributed computing environments.

This manual is designed to help users gain a basic understanding of Rational Synergy's terms, concepts, and methodology. Rational Synergy products have several interfaces, but Rational Synergy is the main interface discussed in this document.

It is assumed that you understand the fundamentals of your Windows® or UNIX® operating system and its associated directory file structure.

Transitioning from other tools

Rational Synergy manages the process of maintaining multiple versions of the same file in an archive, plus much more. As a new Rational Synergy user, you may have previously used a version control tool, such as PVCS (Windows) or RCS or SCCS (UNIX). These tools maintain control of file versions, but without many of Rational Synergy's benefits, such as workflow management, product reproducibility, and rule-based configuration update, to name a few.

Although Rational Synergy differs greatly in process from other version control tools, users familiar with other tools should transition easily to one of the Rational Synergy interfaces.

Conventions

This section describes conventions used in this manual.

Rational Synergy graphical user interfaces

Rational Synergy has the following variety of graphical user interfaces. Note that when this document discusses the Rational Synergy products (Rational Synergy and Synergy Classic), it uses the general name "Rational Synergy products." When discussing a specific interface, it uses one of the following names:

- Rational Synergy

  This interface is for users who work as developers and/or build managers.

- Synergy Classic

  This interface provides CM capabilities for administrators.

Rational Synergy command line interface

The command line interface (CLI) examples shown in this guide apply to both Windows and UNIX platforms.

Typefaces 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 items that you can select, such as buttons, icons, etc., and menu paths. Also used for the names of dialog boxes, dialog box options, toolbars, folders, baselines, databases, releases, properties, and types. 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.
`Courier Italic`	Represents values in a command string that you supply. For example, (*`drive:\username\commands`*).

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.
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/.

2 Benefits of using Rational Synergy

Goals of a configuration management tool

The purpose of a configuration management system is to help development teams control the process of modifying source code, developing documentation, and managing products. The following discussion focuses on some of the major issues that confront development teams, and how a configuration management tool can help reduce or eliminate the challenges of coordinating myriad changes to multiple files by multiple developers.

A configuration management tool must be able to bring in existing projects from flat and spread-out directory structures, as well as from deeply nested directory structures. Once project data is migrated, the tool must provide an easy way to create a baseline, a project version from which all subsequent development effort is based.

Ideally, a configuration management tool should be transparent, allowing developers to work on source code files as usual, insulated from disruptions caused by changes from other developers. At a certain point, however, that insulation must relax so the developers can incorporate other developers' changes into their own projects. Therefore, a configuration management system should provide developers with insulated areas in which to work, yet also provide an efficient way in which they can work as a team, sharing modifications to their source code.

A configuration management tool should enable a team to reach build plateaus, where a greater degree of stability is attained with each higher build level. Additionally, the tool should implement a process by which development can continue as these higher-level builds take place. Such a process ensures that the team is on the correct path to product release and enables them to fix any code that breaks a higher-level build.

A configuration management tool must excel in the ability to reproduce various versions of your software application. Often, development teams work on a defect correction release and a new features release in parallel. These releases may have different build requirements. After the defect correction release is finished, the new features release team normally incorporates those changes into their work. An important goal of a configuration management tool should be to allow several types of projects to take place simultaneously and provide a way for developers to reuse the code, no matter what directory it was created in. Once parallel products are released, technical support personnel must be able to

reproduce past milestone releases for customer support. As the number of parallel releases completed by a development team increases, so increases the need for product version reproducibility.

Rational Synergy benefits

Rational Synergy provides a complete configuration management environment in which development teams can work easily, quickly, and securely. This section describes the features and benefits that Rational Synergy provides to software development organizations.

Easy to use, right out of the box

Most development teams run on tight schedules with small windows for down time. Because it is important that development teams to establish a comfort level with a new tool right away., Rational Synergy provides the following standard features:

- Intuitive, easy-to-use graphical user interfaces (GUIs) and comprehensive CLI
- Simple task-based approach to tracking changes
- Integrations with many popular tools and development environments
- Flexible process support, driven by templates
- No customization required to use the tool productively
- Ready-to-use defined privileges and object lifecycles, security, and access rules
- Help from the GUI and the CLI

Fast start-up

Development teams can start using Rational Synergy soon after it is installed—usually the same day. This is possible because of the following Rational Synergy features:

- It is a fast and automated migration tool that allows you bring existing projects in your file system under Rational Synergy control.
- Rational Synergy is compatible with existing build and make procedures, so you can build products using existing makefiles.

Rapid productivity for new users

Once Rational Synergy is up and running, it is relatively easy for new users to come onboard. If you are a new user, you can start using Rational Synergy right away. Once the CM administrator adds you as a Rational Synergy user, you can copy a project to create your personal work area, then make changes as follows:

- Select the task to work on.
- Copy the project and change files.
- Complete the task, thereby checking in all of your changes.

These steps represent the most fundamental use of the Rational Synergy tool. There are a multitude of features to learn about so you can take full advantage of Rational Synergy. You may want to attend a training session to gain a thorough understanding of the product. However, you can immediately start to work in Rational Synergy to help your team meet a critical deadline or commitment.

Flexible, automated workflow

Rational Synergy's task-based methodology provides a straightforward way to build and test your software so you can find problems as quickly as possible and attain the quality level you require. The task-based workflow enables you to:

- Easily review the reason for a change and identify all the files modified to implement the change
- Develop changes in an insulated environment and see others' changes when ready
- Carefully control the changes that are built into test areas and software releases
- Detect configuration conflicts, such as parallel versions or missing changes
- Preserve and reproduce the software that you ship
- Automate build-management operations
- Manage parallel development
- Easily set up different workflows for different teams

Secure team engineering environment

The following features ensure that your development team can develop software projects with a minimum of attention to the Rational Synergy tool:

- The data repository is a dependable, commercial, off-the-shelf relational database management system (RDBMS).
- Rational Synergy uses developers' existing directory structures and tools.
- Rational Synergy provides developers with private, insulated work areas that give them full access to their own checked-out versions of files to prototype, edit, build, and debug, before making their changes available to other developers.
- Rational Synergy provides project reproducibility by accurately creating baseline configurations.
- Rational Synergy allows you to trace files and projects through task association and check-in and check-out capabilities.
- Developers can use tested and checked-in files on demand by updating their projects with Rational Synergy's update process.
- Rational Synergy manages concurrent development changes through automated parallel development support and built-in security.
- Developers never need to stop development while integration or release areas are being tested, because those areas are insulated from developers' ongoing changes.
- The default lifecycle ensures that only authorized changes are released.
- Rational Synergy controls data and file access operations with security that you can set for each user of the database.

World-wide control and transfer of information

Rational Synergy Distributed Configuration Management (DCM) product allows you to share software changes among any number of Rational Synergy databases, anywhere in the world. With DCM, you can:

- Permit developers to work in the same user interface to which they are accustomed (DCM information is shown in the appropriate dialog boxes)
- Select the appropriate methodology to define the nature and direction of data transfer between databases
- Send source objects, projects, folders, and tasks to any DCM database, with no restrictions on how you group your objects

- Transfer an entire database or a subset of a database, either automatically or manually
- Preview the transfer list before finalizing the transfer
- Continue parallel development in databases in different physical locations, then resolve conflicts using the Compare and Merge features

Seamless integrations for Windows development

The Rational Synergy configuration management tool has been integrated with some of the industry's leading development environments. These integrations enable you to have source control and configuration management from your native development environment. The integrations include an easy-to-run installation and setup program. Many integrations are available, including:

- Eclipse™
- IBM® Rational® Application Developer
- Microsoft® Visual Studio®
- Microsoft® Visual C++®
- Microsoft® Visual Basic®
- Sybase Powerbuilder®

For a complete list of available integrations, see the IBM Software Support Home page for Rational products at http://www.ibm.com/software/rational/support/.

3 Rational Synergy terminology

This chapter introduces Rational Synergy's basic concepts and terms. The concepts are introduced in a sequential order so they build upon one another, so you should read this chapter in sequence and in its entirety.

For quick reference, there is a glossary of terms at the end of this manual.

Note that this manual discusses all of the Synergy interfaces. When possible, this manual uses generic terms that apply to all interfaces; when discussing how a specific interface works, for example the Rational Synergy interface, the terms specific to that interface are used.

The concepts and methodology are the same across all interfaces. For example, all interfaces support the task-based methodology, and all interfaces recommend that developers update their projects when working on a new project to bring in the latest members.

The Rational Synergy database

The Rational Synergy database is a data repository that stores all of your controlled data, including source and data files, their properties, and their relationships to one another. You can have one or many Rational Synergy databases, depending on how much data you want to control and how you want to organize it.

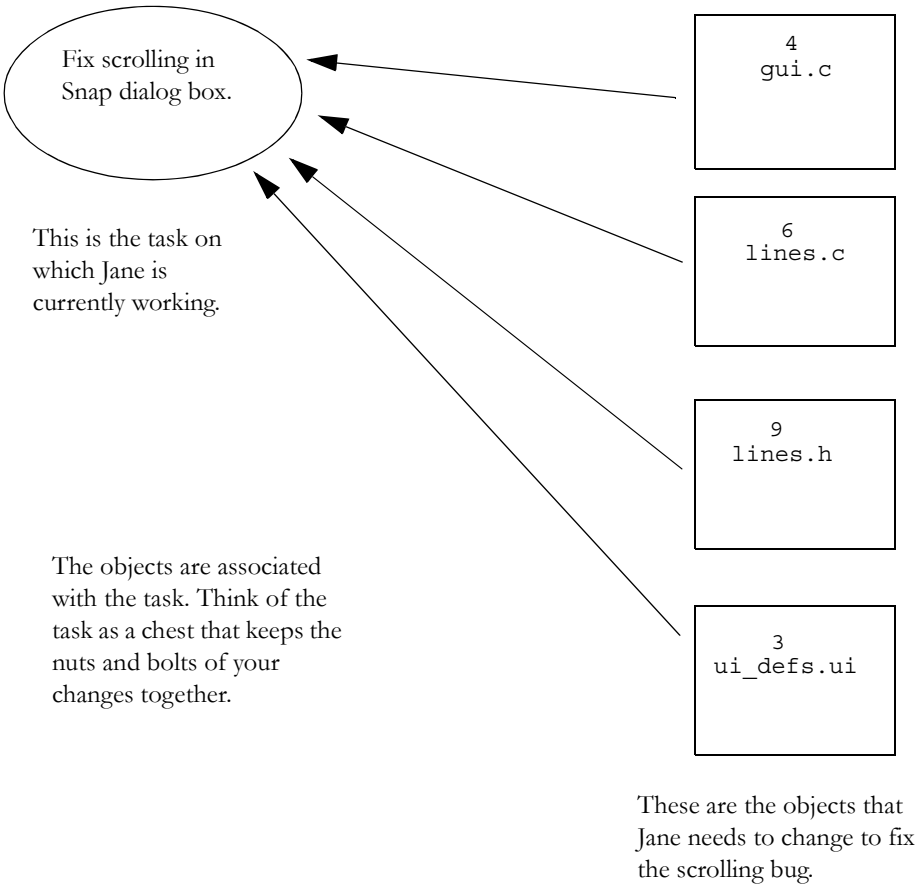
Tasks and objects

A *task* represents a logical change that needs to be made in your software application. A task groups all the software modifications that need to be made to complete the change and includes a description of the change and the name of the person responsible for completing it.

An *object* is a collection of data, such as a file or directory. Examples of objects are source files, makefiles, test results, directories, and documents. For tracking purposes, every revision of an object is referred to as an *object version*. Each object version has a set of *properties* (e.g., **name**, **owner**, **create time**) to further define it.

For example, a user reports a bug in your application's GUI. The GUI group leader assigns the bug to a developer named Jane. She is assigned a task called **Fix scrolling in Snap dialog box**. Each time she uses Rational Synergy to check out an object that fixes the scrolling problem in the Snap dialog box, the object becomes associated with the task.

For an illustration of these concepts, see the figure below.



The task and objects have a relationship. Objects that are grouped by a task are said to be *associated* with the task. All objects required to fix a specific problem stay together in a logical grouping, described by the task name. The object versions on the right side of the figure above are associated with the task **Fix scrolling in Snap dialog box**, because they contain the code changes necessary to complete this task. The number at the top of each object represents the object's version.

The task's name, in this case, **Fix scrolling in Snap dialog box**, is referred to as its *synopsis*. Additionally, when you create a new task, Rational Synergy assigns it a

number. Besides the number and synopsis, a task contains other information about the change, for example, the name of the resolver. When a task is assigned to a developer, the resolver is automatically set to that developer's name. You can also set the following properties when you create a task:

- Release

  Is a label that indicates the version of your software application. The possible values consist of releases that are significant to your software application, such as **editor/2.0** or **Rational Synergy/7.1**. The build manager sets up release values specific to your software.

- Platform

  Specifies the hardware platform applicable to the logical change. The possible values are platform names significant to your software application, such as **AIX** or **WIN2K**. The build manager sets up platform values specific to your software. You do not need to set a task's platform; this value is for your convenience only.

- Subsystem

  Specifies the software subsystem for the task. For example, if you develop a client-server software application, your subsystems might be **client**, **server**, and **communication**. If you develop an accounting software application, your subsystems might be **AR**, **AP**, and **GL**. The CM administrator sets up subsystem values specific to your software. You do not need to set a task's subsystem; this value is for your convenience only.

  The tasks and the source objects to modify reside in the Rational Synergy database. Tasks do not have versions, but they follow a lifecycle (described on page 20). Tasks do not contain other tasks.

More about objects

Any object managed in a Rational Synergy database is uniquely identified by the following properties: **name**, **version**, **type**, and **instance**.

By default, the *four-part name* (also called the *object spec* or *full name*) is written like this:

`name-version:type:instance`

The following are some examples of four-part names: `main.c-3:csrc:2` and `draw.c-beta:csrc:7`

An object name can be any combination of characters, except for restricted characters (refer to Rational Synergy CLI Help for a list of illegal characters for both). The type can be any of the default types (for example, **csrc**, **ascii**, and so on.), or any type you've created. You can designate the name, version, and type, but Rational Synergy calculates the instance.

The *instance* is used to distinguish between multiple objects with the same name and type, but that are not versions of each other. For example, a project could contain twenty different makefiles, each named `makefile`, each in different directories, and each with many versions. If you want to use `makefile-4`, a query of that object might yield six objects called `makefile-4`. In this case, the **instance** property distinguishes which makefile object you want to use. The value of the instance is normally numeric, but may be alphanumeric in some cases, such as in a database that uses DCM.

You can use a specific object version in multiple directories. You can reference an object version through its path name. However, while the file's location might change, the four-part name unique ID always remains the same.

The check out and check in operations

To modify an existing object, you must create a modifiable version of that object. You can create a modifiable object using the *check out* operation, which creates a new version of an object from an existing version. The new version includes copies of all properties from the existing version.

The check out operation is similar to the PVCS `get -l` command (Windows) or RCS `co -l` and SCCS `get -e` commands (UNIX). However, in Rational Synergy, you need to check out an object only if you plan to modify it. If you want to view or use it, you do not need to check it out. You can check out any type of object (files, directories, symbolic links, executables, etc.).

The *check in* operation normally preserves the object version by making it non-writable. Once checked in, an object is available to other users. Checking in an

object changes the **state** (or status) property, which defines who can modify and use it. The check in operation is similar to the PVCS `put -u` command (Windows) or the RCS `ci -u` and SCCS `delget` commands (UNIX).

With Rational Synergy, rather than checking in a file once, you can check in the same version multiple times. For example, you can check in a file when you're ready for testing and check it in to a different state when you're ready to release it.

Using the example of our developer Jane, we know that she is going to modify files to fix the scrolling bug. She checks out files, such as **gui.c** and **lines.c**, that she needs to modify to complete the task assigned to her. After modifying the files and fixing the bug, she completes the task so that these files can be used in the next product build.

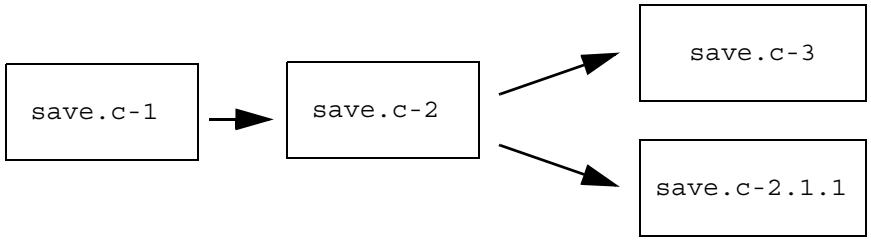
The check out and check in processes are an important part of change control during the development cycle. When a developer checks out an object (for example, a file) to modify it, the developer becomes the owner of a personal copy of that file for his use. By default, Rational Synergy allows another developer to check out his own version of the same file for modification. This is known as *parallel development*, and the different versions of the same file are called *parallel versions*. The work process is not delayed because one version of the file is already in use.

At some point, parallel versions need to be merged. Rational Synergy's *merge* feature enables you to blend information from two parallel versions of a file. When you merge two object versions, a third version is created. Rational Synergy uses the most recent common ancestor to recommend which changes the new version should contain. If there are no conflicts in the files, the new version is ready to use. If conflicts exist, you must choose which lines of the conflicting code to use in the merged version.

History

The *history* of an object shows all the object's existing versions and the relationships between the versions. The term *history* refers to all of the object versions created before the current object version (called *predecessors*) and all of the object versions created after the current object version (called *successors*).

The figure below shows the history of the file `save.c`. The arrows indicate which version was checked out from which; for example, version 2 was checked out from version 1. Version 1 is a predecessor of version 2, while versions 3 and 2.1.1 are successors of version 2. Versions 3 and 2.1.1 are parallel versions.



Properties

An object's *properties* differentiate it from other objects. The basic properties of an object are those items identified by its four-part name (**name, type, instance, version**) and also include **owner**, **status**, **platform**, and **release**. You can view properties in Rational Synergy's **Properties** dialog box.

The `platform` and `release` properties are important when gathering versions of the software for building and testing. Rational Synergy gathers versions whose platform and release values match the configuration you want to build and test.

Current task

The *current task* is the task you are currently working on. When you designate a task as the current task, you tell Rational Synergy that each time you check out an object, you want the object to be associated with that task automatically. When you finish making all the software changes for a task, you can complete the task. Completing a task checks in all object versions associated with the task.

Using tasks enables you to move each logical change as a unit through the Rational Synergy lifecycle. Using a task's information, you can gather versions of software for building and testing by specifying which logical changes you want.

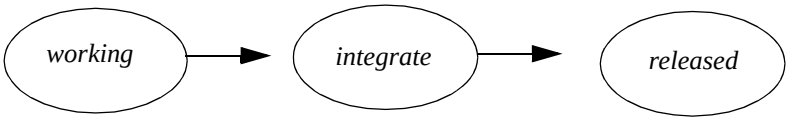
Also, when you use tasks to represent your software changes, you indicate that the object versions associated with a task should be used together, and that using some of the versions in a project without the others probably will not work. With this information, Rational Synergy can help you detect configuration problems early in the software lifecycle.

Users, lifecycles, and states

In a Rational Synergy session, each user can work as a developer or build manager. If a user has been set up to work as a developer **and** build manager, Rational Synergy allows the user to perform the appropriate operations without any interference. Typically, developers perform operations for developing and testing software. Build managers perform operations for integrating software, and configuring and building areas where developers can access the integrated software, as well as configuring and building test areas and preparing software for release.

All objects follow a lifecycle. The *lifecycle* refers to the possible states of an object, and to which states an object can move based on its current state. An object's *state* defines the object's stage in its lifecycle and the actions that can be performed, such as who can modify it.

By default, the three states that objects use in Rational Synergy's task-based methodology are *working*, *integrate*, and *released*. The figure below illustrates the lifecycle sequence of an object's default states.



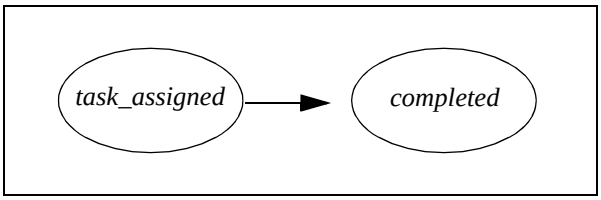
Use these states as follows:

- The *working* state is used for all new object versions, either when they are created or checked out from another version. An object in the *working* state is modifiable by the owner.
- The *integrate* state is used for build-management integration testing. An object in the *integrate* state is non-modifiable.
- The *released* state is used for objects that have been released or have reached a milestone. An object in the *released* state is non-modifiable.

Your organization may use other lifecycle schemes. Several optional states, such as *rejected*, *shared,* and *visible*, are included with Rational Synergy.

The lifecycle for an object associated with a task is closely linked to its task's state. It is the task's lifecycle that moves the object through the process. For example, you can check in an object at any time, but it is not picked up for integration testing until you complete its associated task.

By default, a task can have the following states: *registered*, *task_assigned*, and *completed*. The figure below illustrates the lifecycle sequence of a task's default states.



The *task_assigned* state is used for tasks that have been assigned to a developer. When a task is in the *task_assigned* state, it is modifiable and can be used by the resolver (the person the task is assigned to). By default, build managers can assign tasks. Developers can assign tasks to themselves only.

The *completed* state is used for tasks that are finished. When a task is in the *completed* state, it is checked in and can be modified only by the CM administrator. All object versions associated with a task must be checked in before the task can be completed. Only the task's resolver can complete a task; the resolver can be a developer or build manager.

Projects and project groupings

A *project* is a user-defined group of related files, directories, and other projects (called *subprojects*). A project normally represents a logical grouping of software, such as a library or an executable, and it contains the directory structure of the files. For example, the software that implements an editor application might be stored in a project named **editor**.

Projects are versioned like any other object. Different versions of the same project can contain different versions of the member objects, and even different members. For example, different versions of the **editor** project might represent the first release, **editor/1.0**, and follow-up releases, **editor/1.1**, **editor/1.2**, and **editor/2.0**. The **editor** project for release **editor/2.0** may contain new objects that did not exist in **version 1.0**, as well as newer versions of

many of the same objects. There may also be files that were in **version 1.0** that are not part of the **version 2.0** project.

A single object version can be a member of multiple projects. Even though the same object version appears in different projects, the object version exists only once in the Rational Synergy database.

A project can contain other projects. A project contained within another project is called a *subproject*. You can organize your software by grouping it into different projects. For example, you can set up a project for each executable, and include them all as subprojects under a project that represents the entire application.

For any given project, several different versions can exist:

- development projects are the projects developers use to develop and test their changes.
- build management projects are the projects build managers use to prepare software for testing and release.
- *released projects* are versions of the software that have been released or have reached a milestone.

A *project grouping* groups *working* and build management projects by release and purpose. It is created and maintained automatically to provide a convenient reference point for a set of projects. For example, **My 1.0 Insulated Development Projects** groups all of a developer's projects for the 1.0 release for the purpose of insulated development. A project grouping also contains the tasks and baseline used when a project is updated. This keeps the members selected to be in a project consistent for all projects in the grouping.

Think of a project grouping as a container into which you add different ingredients (properties), such as tasks, baseline, release, and project purpose, to create a group of projects with the right mix of members (files). Once you've created this project grouping, you can easily add, remove, or change the properties, specify that you don't want tasks to be updated during a project update, refresh the baseline and tasks in the project grouping to get the latest set, and move a project from one project grouping to another. This enables you to keep your projects grouped securely, yet gives you the flexibility to make changes when you need to.

Directories and candidates

Rational Synergy controls directories as well as files. Unlike a directory in the file system, a *directory* created in Rational Synergy keeps track of which files belong in it. For each file that belongs in a directory, the directory has a place holder, called a *directory entry*. The directory entry describes the name of the file that belongs there, but *not* the version. For example, the directory entry for `delete.c` knows that it needs a file named `delete.c`, but does not expect a particular version of the file. All the file versions that are eligible to be used in a directory entry are called *candidates*.

For you to add or delete an object from a directory, the directory object must be writable. If you try to modify a directory (by adding or deleting members) that is in a non-modifiable state, Rational Synergy automatically checks out a new version of that directory for you. If your current task is set, the new directory is automatically associated with the current task, and is checked in with the rest of your changes when you complete the task.

Like source objects, parallel versions of directories can occur. Rational Synergy enables you to merge parallel directories. When you merge directories, you compare the differences between directory entries, and select which directory entries should be included in the merged version. For example, if one user checked out the **sources** directory and added an object named **open.c**, and another user checked out a parallel version and added an object named **select.c**, the merge operation would show both new directory entries and you could include them both in the merged version.

The work area

A *work area* is a location in your file system into which Rational Synergy writes a project when you check it out. A work area can reside anywhere in the network file system. A project's directory tree structure in the work area is identical to the project's tree structure in the Synergy Classic database.

Rational Synergy keeps the work area synchronized with the database. On UNIX systems, the files in a work area are *linked* to the database files. On Windows systems, the files in the work area are *copies* of the database files.

When you create or change a project, Rational Synergy updates your work area automatically and transparently—when you add members to a project, Rational Synergy updates the work area with the new files; when you remove members from a project, Rational Synergy removes the corresponding files from your work area. You also can update the work area manually by working on files directly in the network file system, outside of Rational Synergy's control.

The synchronize operation

When you use Rational Synergy operations to update your project data, your work area and the Rational Synergy database are both updated with your changes and kept in sync. But if you work directly in the work area, updating your files without using Rational Synergy operations, the work area can become different than your project in Rational Synergy database. You can use Rational Synergy's *synchronize* operation to compare your work area with the database and update them selectively. Rational Synergy informs you when database and work area files are different, so you can compare the differences and choose which to update. You can also choose to merge the files; if you do so, the merge tool shows you the changes side by side and lets you select between the individual lines that differ.

If you need to work outside the Rational Synergy database, you can modify a file in your work area, regardless of whether the database object version is checked out. When you reconnect to the database and synchronize your project, Rational Synergy automatically checks out a new version of the object from the database and adds to it the changes you made to the object in the work area.

The synchronize operation is important for data integrity. If you do not save the work area file changes to the Rational Synergy database, the work area files are subject to the reliability of the system. When you synchronize, your files become part of the Rational Synergy database and are backed up whenever the database is backed up.

Using, creating, adding, deleting, or removing objects

Earlier in this chapter we discussed the check out operation, which creates a new version of an object from an existing version. Remember that an object can be a file, directory, document, or other collection of data. There are several other ways to modify the contents of your project:

- **Use a different version of an object** – If you want to select a different version of an object in your project, for example, go back to an earlier version, you can *use* the version of the object you want. The use operation is especially helpful during the debugging process. If your testing fails, you can replace an object with an earlier version to troubleshoot the problem. You can use any version of an object except those checked out to other users (for example., objects in the *working* state).

- **Create an object**

  When you want to create a completely new object, instead of checking out a new version of an existing object, you *create* the object. This operation is similar to the PVCS **vcs -I** command (Windows) or SCCS `create` command (UNIX). An object version must be created explicitly (i.e., it must be under Rational Synergy control) before you can check it in. When you create an object, you create it in a directory within a project, and it appears in the project's work area.

  You can also create projects and directories. When you create a project, its work area is created immediately. When you create a directory, it is empty until you create to or paste in it.

- **Add an object**

  You can add an existing file, directory, or project to a directory in your project by copying and pasting or dragging and dropping it. You don't need to check out an object to add it to your project.

- **Remove an object**

  You can remove objects by performing a *delete* or *cut* operation. When you **delete** an object, it is removed permanently from the database; you cannot delete an object if it is used by another project in the database. When you **cut** an object, you remove it from your project, but it remains in the database. If you need it later, you can add it back to your project.

Update, baseline, tasks, and process rules

Update is the process of updating the object versions in a project or directory. Each object version in the project or directory is evaluated, and the appropriate version is selected from the available candidates in the Rational Synergy database. Developers normally update their projects whenever they start working on a new task. They do this to bring in the latest members of a project.

A set of projects for a particular release and purpose bases its members on a *baseline*. A *baseline* is a grouping of static projects and tasks. Think of it as a snapshot in time of one or more projects and the *tasks* they contain. It might represent a particular build, a milestone, or a release.

Note that if a project uses process rules, the process rules identify which baseline to use. The projects that reference the process rules use the baseline to identify which baseline project to use when updated. (A baseline project is a starting point for the project; each project looks at the baseline to find its starting point—called a baseline project.) For example, if the **Insulated Development** process rule for

the current release specifies that the **Integration Build 20020913** baseline should be used, and it contains static projects **toolkit-int_20020913** and **calculator-int_20020913**, a developer's **calculator-bob** project would select **calculator-int_20020913** as its baseline project.

Therefore, *process rules* are patterns that define how projects are updated. They specify rules for selecting a baseline plus a set of tasks to be used when you update a project. Your team uses process rules to tailor and coordinate their software development and testing process.

The update operation also uses the tasks in the baseline. This streamlines the tasks that are evaluated, which improves the performance of the update operation. An update that uses baselines only analyzes the tasks that were added since the last baseline, rather than all tasks for the entire release.

The build manager typically creates the baseline and sets up the process rules, then makes them available to developers for a particular milestone or release.

Grouping tasks in folders

A *folder* is a named grouping of tasks set up by the build manager. A folder is used to gather all tasks that should be grouped logically, for example, according to the task's state, release, owner, or any combination of these properties. Examples of folders are **All Tasks Assigned to Jane** or **All Completed Tasks for Release editor/2.0**.

You can add tasks to a folder in two ways:

- Manually select each task you want to add. Once you use this method, the folder contains only that set of tasks until you manually modify the folder again.
- Specify a database query. For example, you can set up a query that selects all of your tasks for release **editor/2.0**. Whenever you access that folder, Rational Synergy queries the database and gathers the tasks that match the query criteria. The advantage of using a query is that you do not need to manually update the folder every time you create a new task; if a task matches your query's criteria, Rational Synergy automatically adds the new task to the folder.

Another useful aspect of folders is they allow multiple users to share a set of tasks. For example, after a software configuration passes integration testing, the tasks that passed testing can be made available to developers in a folder.

The build process, products, and makefiles

Build is the process of generating a file from existing source files, using a tool or compiler or code generator. The documentation uses the terms *build* and *make* interchangeably.

Files that are built by processing other files are called *products*. Any object type is a potential product; the most common products are executables, libraries, and relocatable object (`.obj`) files.

Projects can contain any number of *makefiles*, which are files that contain the instructions for building products. You can use a third-party make tool of your choice

Products can be uncontrolled or controlled. *Uncontrolled products* exist in your work area, but not in the Rational Synergy database. *Controlled products* are product files that are controlled as object versions within Rational Synergy. Because controlled products exist in the Rational Synergy database, users can share them.

An object becomes a product by manually marking the object version as product.

Note All types of controlled objects can be products except for projects, symbolic links, and directories (because they are not used as targets in makefiles).

4 Rational Synergy methodology

Because the information presented in this chapter is sequential, so you should read it in the order presented.

Task-based methodology

Methodology is the process and strategy used to manage software. Within Rational Synergy, the methodology controls the flow of software throughout the development cycle, from original development, through testing, release, and maintenance.

Rational Synergy supports the *task-based methodology*, which enables you to track changes to your software application using tasks as the basic unit of work. A task represents a single logical change. The following discusses some of the benefits of task-based CM.

- Task-based CM is intuitive.

  Developers naturally think in terms of logical changes, and mentally map each change to the specific files that need to change. With most configuration management systems, developers must remember to check in each file they change. The task-based CM methodology helps developers work the way they think, by automatically keeping track of all related changes and checking them in together as one step.

- Task-based CM takes the guess work out of creating a release.

  With task-based CM, you can configure your application as a baseline plus a set of tasks. It makes sense to create a release of your application by starting from the last milestone or release and adding a specific list of fixes or enhancements.

- Task-based CM warns you of potential conflicts between files.

  Because it knows more about the relationships between files than non-task-based systems, task-based CM can detect conflicts in your software configuration before testing occurs. Rational Synergy can detect missing or partially missing tasks when you update your configuration.

- Task-based CM provides more information about your release.

  You can list the contents of a release in a meaningful way using the task descriptions, rather than just a list of source files.

- Task-based CM integrates change requests with the tasks that fix them.

Tasks can be related to the defect and enhancement reports your customers submit, providing a tight integration between your change request system and actual software changes.

Users

A Rational Synergy user can perform those operations applicable to a developer or build manager. The CM administrator sets this for each user at the database level. These settings determine what operations you are allowed to perform in a database, and may be different for each database in which you work. For example, user Jane may be able to perform developer and build manager operations in the **main_product** database, but only perform developer operations in the **integrations** database. During a Rational Synergy session, if the CM administrator has set you up to be able to perform developer and build manager operations, you can perform the appropriate operations without ever having to change settings or restart a session.

Since this chapter discusses the development methodology, the discussions concentrate mainly on operations that developers and build managers can perform.

Projects and workflow

A project contains a specific set of member objects and provides an insulated working environment (for basic information on projects, see "Projects and project groupings" on page 20). Different versions of the same project can be used for different purposes. For example:

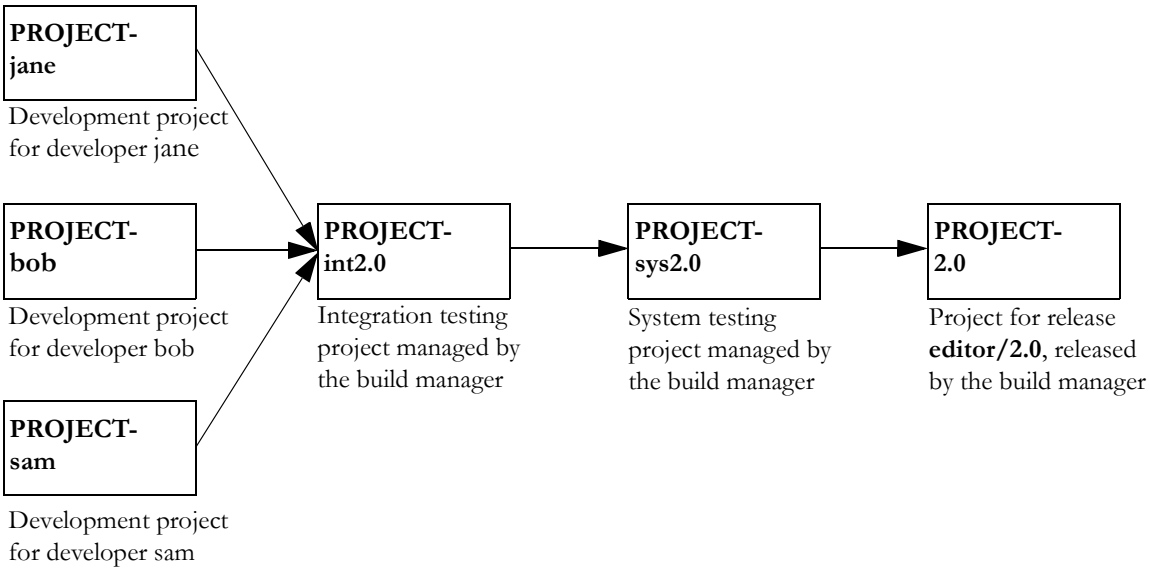
- Developers each have a working version for developing and testing their ongoing changes.
- A separate version of the project can be used to collect the latest completed tasks for integration testing.
- A version of the project can be used to build a specific set of changes for system testing.
- Another version of the project can be used to save a specific configuration as a release or milestone.

All these projects for different purposes enable a team to work together on the same application. The projects and the way they are set up to select changes define your team's workflow.

Rational Synergy provides a default workflow consisting of the following stages:

- Developers develop and test their changes in their development projects. When they complete a task, it is available for inclusion in the integration testing projects. When developers update their projects, they keep their own checked-out versions, and they get the latest versions that have passed integration testing.
- The integration testing projects gather all the tasks that have been completed to date. These projects are often used to implement a "daily build and smoke test," a best practice for software development. The goal of integration testing is to find problems as soon as possible after they are introduced. The build manager manages the integration testing projects.
- The build manager creates a baseline when the build passes integration testing.
- The system testing projects are used to build a specific set of changes for in-depth testing. The build manager defines and updates the list of changes to ensure that projects are insulated from developers' ongoing changes. Individual fixes can be added, built, and retested until the project meets the team's quality standard. System testing projects are often used to prepare for a release or milestone.
- After the software is released or reaches a milestone, the build manager can baseline or release projects to preserve the configuration. Released projects can be used as baselines for new releases.

The figure below shows an example of how projects are used to implement Rational Synergy's default workflow. The arrows indicate the flow of tasks through the projects.



Rational Synergy is set up so that you can use this methodology out of the box. However, Rational Synergy's process model is flexible, enabling you to customize the default methodology to suit your team's process.

Release

In Rational Synergy, you always work on a particular *release*. The release is a label that indicates the version of your software application. For example, the first release of your software might be **editor/1.0**, and the second release could be **editor/2.0** or **editor/1.1**.

When you check out a project, you specify the release to use. Likewise, when you create a task, you specify the release in which the task is included. The release is important because Rational Synergy uses it to organize your tasks and projects and to ensure that tasks are used in the projects with matching release values.

Rational Synergy stores releases for your software application. A release enables you to mark projects, tasks, and folders for particular releases. It also helps you to keep track of which object versions were developed for each release.

Only a build manager can create or alter releases. Build managers can view them in the **Releases** explorer or with the `ccm release` command. Each Rational

Synergy database has its own set of releases, although you can transfer them between databases by using distributed change management (DCM).

A typical release can be any of the following. The examples in the following table show the release, which is created by the build manager. It's made up of the component name and the component release. The release is what you see.

Release	Component name	Component release
1.0		1.0
2.0		2.0
2.0_patch		2.0_patch
Rational Synergy/7.1	Rational Synergy	7.1
editor/2.0	editor	2.0
editor/2.1	editor	2.1

A release consists of an optional component name and release delimiter, and a component release. The component name might represent the name of an application or component, such as **Rational Synergy** or **editor**. The component release identifies the specific release of that application or component.

Note that the component name is not a mandatory part of the release. In the first row in the table above, the **1.0** component name doesn't have a component, and Rational Synergy leaves it blank

Whenever you check out an object, Rational Synergy automatically copies the release from the current task to the new object.

Project purpose

A project *purpose* is a setting that specifies a project's use and ties it to a set of rules for an update process. Rational Synergy provides the following predefined project purposes:

Purpose	State
Insulated Development	*working*
Collaborative Development	*working*
Custom	*working*
Integration Testing	*prep*
System Testing	*prep*
Shared	*shared*
Visible	*visible*

The Insulated Development, Integration Testing, and System Testing purposes are used for the default methodology as described in the preceding section. The Shared, Visible, and Collaborative Development purposes are for teams that use variations of the standard methodology that enable them to work together more closely. The Custom purpose enables developers to specify the baseline and tasks for their custom projects.

The purposes used most by developers are Insulated Development, Collaborative Development, and Custom. The Insulated Development purpose is the default purpose used when a developer creates a new project. When the developer updates his project, he'll get all of his own assigned and completed tasks for the current release, in addition to the latest Integration Testing baseline for the current release.

The Collaborative Development purpose is available as an alternative to Insulated Development. Typically, smaller teams use this purpose when the chance of other developers' changes breaking the build is less likely. When a developer uses the Collaborative Development purpose and updates his project, he'll get all of his own assigned and completed tasks for the current release, and all completed tasks from other developers for the current release, in addition to the latest Integration Testing baseline for the current release.

The purposes used most by build managers are Integration Testing and System Testing. The Shared and Visible purposes are used by teams requiring an alternative to the default, task-based methodology.

When you create or copy a project, you specify its purpose, and Rational Synergy automatically sets up your new project for that purpose by setting the project's update properties.

Additionally, the State column shows the state in which the project is created by default for the purpose. For example, a project is created in the *visible* state only if its purpose is Visible. A project is created in the *working* state if any of the following purposes are chosen: Insulated Development, Collaborative Development, or Custom. The state also ensures that the project selects the correct members when you update.

Update properties

When you update a project, the project uses a set of properties, called *update properties,* to automatically determine which object versions to select and bring into your work area. (Recall that in previous releases, "update" was called "reconfigure.") Update properties are stored with a project grouping, and consist of a baseline plus a list of tasks and/or folders.

All project groupings have update properties. When you check out a project, the purpose you select (i.e., Insulated Development, Integration Testing, etc.) and the project's release value determine how the project's update properties are set up. Alternatively, you can manually set your update properties.

When you perform an update, Rational Synergy updates your project as follows:

1. Rational Synergy determines which tasks to use. It evaluates each folder listed in the update properties and adds its list of tasks, plus any tasks you specified directly in the update properties.
2. Rational Synergy calculates a list of object versions by looking at each task in the list that is not already in the baseline. This list of objects, plus the members from the baseline project, becomes the candidate list for your update. Only the object versions in this list are considered as candidates.
3. Rational Synergy evaluates each candidate using a simple set of rules that compare the candidate's properties with the project's properties to select the best match.

The default workflow of Rational Synergy

This section describes Rational Synergy's default workflow.

The use of tasks

A *task* represents a problem in or enhancement to your software application. Because a task groups all the objects you are modifying for a specific problem or enhancement, you only need to complete the task that groups those objects, rather than check in the objects individually. In this way, tasks do a lot of the work for users.

By default, any Rational Synergy user (developers, build managers, and so on) can create a task. Tasks can also be generated and assigned based on change requests submitted by customers or technical support engineers.

When someone creates a task, he can *assign* it immediately, if he knows who should resolve the problem. A build manager can assign a task to himself or another user; the user who creates a task can assign it to himself. When a user assigns a task, he should set its release value to indicate the version of the software application in which it is included.

After tasks are assigned, developers use the following process:

1. Select a task to be the current task.

   You can select any of your assigned tasks to be the current task.

2. Make all changes necessary to complete the task.

   Because Rational Synergy automatically associates all object versions you change with the current task, any object on which you perform an action (for example, check out or add objects), is associated with the current task.

   Perform unit testing, so you know if additional modifications are required.

3. Complete the current task.

   When you complete the task, Rational Synergy first checks in the objects associated with it, then completes the task. Completed tasks are available to the build manager for integration testing and to the build manager for further integration and system testing. After completed tasks pass integration testing, the build manager makes them available to other developers.

The development process

Each developer working on a project has a *working version* of the project. Developers copy their development projects from the build manager's integration testing projects.

Normally, a developer does not check in development projects. If a developer makes and tests a change in his development project, Rational Synergy automatically checks in the individual object versions that implement the change when he checks in the task; the developer does not check in the project itself. Development projects are like containers—they can be reused from release to release. The projects' contents change each time the developer updates.

When a developer starts work on a new task or is ready to update his project to bring in the most recent changes, he updates it. He keeps a project current by updating it when needed, and he uses the same version of a development project for every change to its member objects.

Each developer is responsible for using his development project to unit test his changes. The developer should update his project, then re-test his changes before checking in the task, to verify that his changes are compatible with other developers' latest changes. After unit testing is complete, the developer should complete the task, thereby checking in all interdependent objects simultaneously. This ensures that all necessary objects are available to the build manager for integration testing.

By default, when a developer updates his development projects, Rational Synergy collects all of his own assigned and completed tasks for the current release, plus the latest tasks that have passed integration testing for that release.

The integration test cycle

The goal of the integration test cycle is to find problems as early in the development cycle as possible. Recall that build management projects are the projects build managers use to prepare software for testing and release. The *integration testing projects* collect the most recently checked-in changes and build them for integration testing. Because the integration testing projects contain many recent changes and users are continually checking in new objects, the integration test area is typically unstable. Such instability is to be expected, because the goal is to find problems.

By default, when a build manager updates the build management projects to build them for integration testing, Rational Synergy collects all the completed tasks for the current release. The build manager does not want to include developers' assigned tasks because the developers are still working on the fixes and the objects are not ready for integration.

The integration test cycle is often iterative—the team may build, test, fix, and add tasks many times before the software reaches the desired quality standard. Normally, when the build manager updates a set of integration testing projects, the task list is refreshed automatically to get the most recently completed tasks. If a build manager needs to fix a broken build, he'll want to stop the completed tasks from being automatically brought into the integration testing project, and then include only the tasks that fixes the build to the integration testing project grouping.

Consider a build manager who is updating the integration testing projects for release editor/2.0. The following default behavior occurs:

- Each project's baseline project is set to the appropriate project in the latest Integration Testing baseline.
- Each project includes the folder, **All Completed Tasks for Release editor/2.0**. This folder uses a query to select all tasks for release **editor/2.0** that are in the *completed* state.

The build manager updates his integration testing project regularly to build the software for integration testing. When he updates the project, the **All Completed Tasks for Release editor/2.0** folder uses its query to select from the database all tasks that were completed by developers working on release editor/2.0, and the project is updated with the object versions associated with those tasks. Then the build manager can build the software application.

If the build is not successful, the build manager can do two things: he can create tasks and assign them to the developers whose objects failed to build, or he can tell the developers whose objects failed to build that they need to fix them, then each developer would create his own task.

Once the build manager successfully builds the product, he creates a new baseline. When a developer updates his projects, this ensures that the most recently tested changes are brought in.

Normally, the build manager does not check in integration testing projects. These projects are like containers—they can be reused from release to release. The projects' contents change each time the build manager updates and brings in developers latest completed tasks.

The system test cycle

Once the development team has made significant progress towards a stable build or a milestone, the build manager typically builds a software installation for the Quality Assurance (QA) team to use for system testing. The goal of system testing is to prepare the software application for a milestone, such as a release. A *system testing project* represents all developers' work that is ready for system testing; it contains the versions of the files, directories, and products used for system testing and release preparation.

Because developers continue to complete tasks as they develop and test their changes, the integration testing projects continue to pick up the developers' latest changes. The build manager needs to prepare the more stable system testing projects as an area insulated from developer's newly checked in changes.

When a build manager updates the build management projects to build them for system testing, he specifies an exact list of tasks to be tested. By managing the exact list of tasks that are included in the software, the team can fix, build, and retest the software until it meets their quality standard. The system test cycle is often iterative—the team may build, test, fix, and add tasks many times before the software reaches the desired quality standard.

Consider a build manager who is updating the system testing projects for release editor/2.0. The build management projects are set up as follows:

- Each project's baseline project is initially set to the appropriate project in the latest Integration Testing baseline.
- On subsequent iterations, the build manager stops the latest Integration Testing baseline from being automatically brought into the system testing project. He'll add tasks as needed to get the system testing build to the desired quality level. (He'll be testing an exact list of tasks and only wants those tasks that he's specified.)

The build manager initially sets the System Testing project grouping to pick up the latest Integration Testing baseline, in preparation for a new system test cycle. The system testing then iterates through the following steps:

1. The build manager updates and builds the system testing projects.
2. The QA team tests the resulting software.
3. The team reviews any problems that were found and decides which of them should be fixed for this cycle.
4. The team creates tasks for the problems that were approved in step 3 and assigns them to developers.

5. The build manager stops the System Testing project grouping from picking up the latest Integration Testing baseline.
6. After the developers have completed the approved tasks, the build manager adds them to the system testing project grouping. The process starts again at step 1.
7. Repeat step 1 - step 6 until there are no additional problems to fix.
8. Create the baseline.
9. Release the baseline.

Releasing the system test baseline

Once the software ships, the build manager should immediately transition the system test baseline used to build the software to the *released* state, to create a starting point for future releases. All objects in the *released* state are non-modifiable, which guarantees that the software is preserved and can be re-created later, if necessary.

Preparing for the next release

Once the build manager releases a project, it can be used as a baseline for the next release. The following steps are necessary in preparing for the next release:

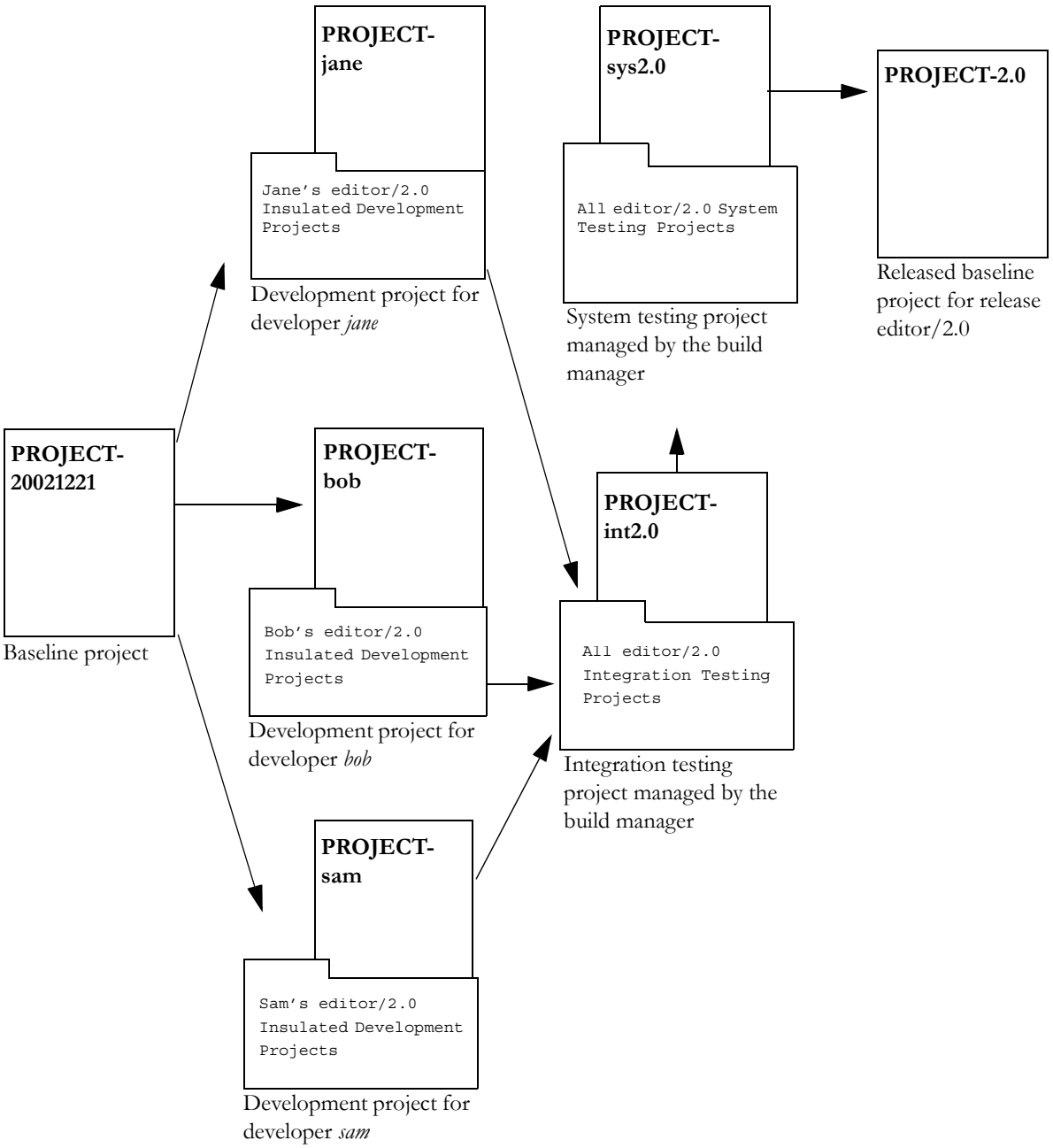
- The build manager adds the new release to the list of releases and sets up process rules for the new release.
- The developers update their development projects to use the new release so they can reuse the projects while working on the new release.
- The build manager changes the integration testing projects to use the new release. Because integration testing projects are not checked in, the build manager can reuse the projects for integration testing of the new release, by updating the projects.
- The build manager changes the system testing projects to use the new release.

  If your team started the next release before finishing with the current release, the build manager needs to perform a copy project to create a new integration testing project and a new system testing project. (He copies the old projects to create the new.)

Summary

We have now covered the entire development cycle, from developers completing tasks in their development projects, through the build managers gathering and testing of tasks, to the final project release and the establishment of a baseline for the next release. The figure below summarizes how projects are used to implement the workflow. The arrows show how tasks flow through the various projects. The key points in the figure are:

- The project shown as the baseline project is the one that developers normally copy to a development project. Alternatively, a developer could copy the Integration Testing or System Testing project. Developers usually copy from a project that is a member of the baseline. The project shown at the end, **PROJECT-2.0**, is part of the new released baseline. In the example, only one project is shown for the sake of simplicity. Typically, a baseline contains many projects.
- Each developer's project includes the latest baseline. The build manager creates a new baseline after the completed tasks pass integration testing. Each developer also has a personal folder (for example, **Jane's Assigned or Completed Tasks for Release editor/2.0**) to gather his or her tasks for the specified release. Each developer can add tasks to or remove tasks from his project grouping.
- The build manager uses the integration and system testing projects for testing. The integration testing projects use a folder named **All Completed Tasks for Release editor/2.0**, that gathers tasks using a query. The build manager adds approved changes to or removes approved changes from his project grouping.
- The build manager creates a baseline from each integration testing build and system testing build that passes the appropriate level of testing. At the end of the release, the build manager releases the final system testing baseline.


PROJECT-jane
Jane's editor/2.0 Insulated Development Projects
Development project for developer *jane*
PROJECT-sys2.0
All editor/2.0 System Testing Projects
System testing project managed by the build manager
PROJECT-2.0
Released baseline project for release editor/2.0
PROJECT-20021221
Baseline project
PROJECT-bob
Bob's editor/2.0 Insulated Development Projects
Development project for developer *bob*
PROJECT-int2.0
All editor/2.0 Integration Testing Projects
Integration testing project managed by the build manager
PROJECT-sam
Sam's editor/2.0 Insulated Development Projects
Development project for developer *sam*

Parallel development

Parallel development is the simultaneous development of more than one version of an object. By default, Rational Synergy allows you to develop any object type (for example, **csrc**, **library**, etc.) in parallel. It is important to understand that parallel object versions must have some characteristic (i.e., property) that distinguishes one from the other, so that Rational Synergy can select the correct version in a project. Such characteristics determine how the object version is evaluated during an update. The properties that distinguish parallel object versions from each other are called *parallel development properties*.

Rational Synergy supports the following types of parallel development:

- **Parallel concurrent development** occurs when multiple developers check out their own *working* versions from the same object. Each developer most likely works on different sections of code. Once they complete their work, the two versions of code should be merged.
- **Parallel variant development** (also known as *parallel platform development*) occurs when multiple developers are working on different versions of the same object for different hardware platforms (often called **variants**). The different versions of the objects (such as one version for Windows and another for UNIX) are usually not merged.
- **Parallel release development** occurs when an organization needs to produce multiple releases of its software product simultaneously. An example of this situation would be different developers working on the next release, a patch to the current release, and a maintenance release, all baselined on the current release. Parallel releases are typically merged after one of the releases is finished. For example, when a patch for the current release is finished, it is merged into the maintenance release and when the maintenance release is finished, it is merged into the next release.

The following paragraphs describe how Rational Synergy manages these types of parallel development.

Parallel concurrent development

Rational Synergy manages parallel concurrent versions using the values of the `status` and `owner` properties. When an object version is in the *working* state, only the owner can include it in his project. The update process compares the owner of a *working* object version with the owner of the project being updated, to ensure that the correct version is selected. As soon as two parallel *working* versions (and their associated tasks) are checked in, they must be merged, since there is no single version that contains all the changes. If the versions are checked in but not merged, a project update that includes both tasks selects the version with the latest creation time, and the project shows a parallel conflict.

Parallel platform development

The **platform** property identifies a project or object that is designed for a particular platform. If you build your software for multiple platforms, you need a platform-specific project for each platform, and each project version needs its `platform` property set. For example, if you build the **snap** project for UNIX and Windows, you need two versions: one with its **platform** property set to **unix**, and one with the property set to **win**.

When you update your projects using a baseline, folders, and tasks, selection is limited to the candidates specified by your update properties. If your candidates contain parallel versions for different platforms, you must set the candidates' **platform** property so that the update operation can select the version that matches the project's **platform** property.

For example, consider the **snap** project, which contains **line.c**. This version of the project has its platform value set to **win32**, indicating that it is built for the 32-bit Windows platform. The **line.c** object has two versions: one marked for **win32**, and one marked for **unix**. If both are candidates because they were included in the project's folders or tasks, the selection rules picks the one whose platform value matches.

Parallel release development

Recall that the **release** property identifies a project or task that is specific to a particular release. If you are developing software for multiple releases, you need one version of each project for each release, and each project version's **release** property should be set accordingly. This ensures that each project selects the matching subprojects and tasks during an update.

Your tasks must be marked with the correct **release** property so they are selected by the projects with a matching release.

Note You must create separate tasks to make a change that applies to multiple releases.

Component-based development

A component is one or more library or executable files, along with supporting files that indicate how it is used, such as header files, help, information about compatibility and dependencies, hardware or software requirements, design information, test cases, and so on. A component can also contain source code.

In Rational Synergy, a component can be represented by an individual file or a project.

Because file versions are reusable in Rational Synergy, they can be created or built in one project and used in another. For example, the `ccmscci.dll` library file might be built in the **ccmscci** project where its source code resides, but the same file can also be included as a member in both the **visual_studio_integration** and **va_java_integration** projects. (Each project can contain a different version of the file, depending on its needs.)

In addition, you can choose to create a new project to contain several files that are published together as a component. For example, you might create a new project called **ccmserver_ext** (similar to the **ccmserver** project, but for external use), containing the files `ccmserver.jar, ccmserver.properties`, and `ccmserver.html`. You may have many versions of the **ccmserver_ext** project, each containing a published version of the component with compatible files.

You might also consider an entire source project a component, if the component users need to be able to view or modify the code.

Because components can be composed from other components, you might even have an entire project hierarchy that represents a component.

Managing components

In Rational Synergy, a *release* specifies the release label of your software application, for example, Rational Synergy/7.1 or Rational Change/5.2. Releases are similar to versions, but they apply to an entire software product. A release can represent a product you have already delivered (or released), or it can represent a release you are currently developing. Each project is usually marked for a specific release, as is each task.

Each component should have its own release stream. For example, consider a team who develops a GUI library and two applications: a calculator and an editor. They might set up releases similar to the following example.

Component	Release Streams
GUI library	gui_lib/1.1, gui_lib/1.2, gui_lib/1.3
Editor application	editor/1.0, editor/2.0, editor/2.1, editor/3.0, editor/4.0
Calculator application	calc/1.0, calc/2.0

The purpose of setting up a different release stream for each component is to decouple the different components so they are independent of one another. This enables the components to be on different release schedules, and ensures that the development teams can use different processes if they choose.

Publishing components

A component can be published by releasing (or checking in to a non-modifiable state) the files or projects that represent the component. When publishing components, the component developer develops software and publishes components for use by others, while the build manager gathers and builds software for integration testing, and the build manager gathers and builds software for system testing.

If a component is developed by a structured team who uses rigorous testing procedures, the component typically is published by the build manager. Conversely, if a component is developed by a small, informal team or an individual developer, the component developer publishes components.

At the time a component is published, the publisher (build manager or component developer) may choose to associate it with a task so that others can reference it. If a team wants to be notified automatically when components are published, they can define triggers.

Referencing components

A component can be associated with a task. A task enables the consumers of the component to specify which component version to use. Typically, each version of a component is associated with a separate task.

The person who publishes the component might create a task and associate it with the appropriate files and/or projects. It is also possible for the component consumer to create a task and associate it with the component files he or she needs. A given component can be associated with several tasks.

The benefit of having the consumer create the task is that he or she can unit test the new version of the component, make any additional changes needed, and associate them with the same task. This keeps together all of the logical changes of upgrading to the new component version, and the rest of the team is not affected by the new component version. (Note that a given component version may be associated with multiple tasks: one for the team who developed it, and potentially one for each team that uses it.)

Rational Synergy also enables you to group tasks into folders. This means you can build folders that group sets of compatible component versions that have been certified for use together. Such a folder can be shared by different consumer applications that want to reuse those exact sets of components.

Process patterns

Component-based development promises to accelerate the delivery of software solutions with lower cost, improved quality, and increased customer satisfaction. The practice is sweeping the software development industry; however, most of today's tools and techniques overlook the critical issue of how teams work together to manage, publish, and share components.

Rational Synergy provides a strong framework and process for sharing components between teams or individuals. It enables you to manage, publish, reuse, and distribute components, while its built-in workflow enables and encourages best practices for software development.

Rational Synergy supports component-based development in a flexible way, providing a variety of process patterns from which to choose. The process patterns are described in the white paper, *A Framework for Managing Component Based Development*, available on the IBM Software Support Home page for Rational products can be found at http://www.ibm.com/software/rational/support/.

5 Terms and concepts

.

Term	Definition
assign	The process of allocating a particular task to a developer to work on it.
associated	Objects grouped by task are said to be associated with the task.
attributes	See properties.
baseline	A snapshot of a set of projects and tasks at a point in time. May be used as the starting point for further development; may be compared to other baselines for reference.
baseline project	The project version on which you base your project is called its baseline project. For example, the baseline project for the **editor-2.0** project would be **editor-1.0**. When you check out a new version of a project, its baseline project is set automatically.
build	The process of generating a file from existing source files, using a tool, compiler, or code generator.
build manager	A person in a development organization who is responsible for building projects and products from source files. Build managers can create, copy, modify, and delete processes, releases, folders, and folder templates. Build managers can create, modify, and delete process rules, purposes, and releases.
build management project	Projects build managers use to prepare software for testing and release.
candidates	The file versions that are eligible to be used in a directory entry within a project.
check in	An operation used to make a developer's object version available to other users.

Term	Definition
check out	A process that creates a new version of an object from an existing version stored in the Rational Synergy database. Developers check out objects so they can work on them.
CLI	Acronym for "command line interface." You can perform most Rational Synergy operations from the UNIX or Windows CLI.
collaborative development	The ability of reusing file versions in Rational Synergy. File versions can be created or built in one project and used in another.
completed state	A state assigned to a task that is finished.
component	One or more library or executable files, along with supporting files that indicate how it is used, such as header files, help, information about compatibility and dependencies, hardware or software requirements, design information, test cases, and so on.
component name	*See* "release" on page 52.
component release	*See* "release" on page 52.
controlled product	A product file that is controlled as an object version within Rational Synergy.
copy project	Copies a project for use by a developer. Developers must copy a project if they need to modify it.
current task	The task a developer is currently working on. Once a developer designates a current task, any object he checks out is automatically associated with that task.
DCM	Acronym for "Distributed Configuration Management," which is a Rational Synergy module that allows you to share software changes among any number of Rational Synergy databases that are located anywhere in the world.
development project	A project a developer uses to develop and test his changes.

Term	Definition
directory entry	A placeholder in a Rational Synergy directory that keeps track of a file that belongs in that directory.
external project	A special project that contains products. Developers can access the products without having to copy the projects containing the source code used to build them.
folder	A named grouping of tasks.
folder template	A pattern used to create folders. A folder template has a set of properties that apply to each folder that is based on the template: the name, who can write to the folder, who can use it, whether it is updated manually or using a query, and the query used to select tasks. A process rule can use folder templates as part of the basis for how a project is updated.
four-part name	A unique identifier for an object in the Rational Synergy database. A four-part name is written like this: **name-version:type:instance** (Also called the *object spec* or *full name*.)
full name	*See* four-part name.
GUI	Acronym for "graphical user interface." You can perform many Rational Synergy operations from the GUI.
history	All of an object's existing versions and the relationships between the versions.
History dialog box	A Rational Synergy dialog box that displays an object's history. *See* "history" above.
instance	An property value used to distinguish between multiple objects with the same name and type, but that are not versions of each other.

Term	Definition
insulated development	Rational Synergy enables developers to develop and test their changes in their development projects without getting changes from other developers until they want them. When developers complete a task, it is available for inclusion in the integration testing projects. When developers update their projects, they keep their own checked-out versions, and they get the latest versions that have passed integration testing.
integrate state	A state given to an object that was checked in by a developer. Other developers can check out and use an object that is in the *integrate* state.
integration testing project	A project used to gather all of the tasks that have been completed to date for integration testing.
integration testing	The process of building and testing discrete software changes together to be certain they work correctly.
lifecycle	A set of states through which an object version may be transitioned. An object version can be in only one state at any given time.
methodology	The process and strategy used to manage software development.
object	A collection of data, such as a file or directory. Examples of objects are source files, makefiles, test results, directories, and documents.
object spec	*See* "four-part name" on page 49.
object version	A specific revision of an object. Each object version has a set of properties (e.g., **name**, **owner**, **create time**) to further define it.
parallel development property	The properties that distinguish parallel object versions from each other.
parallel concurrent development	Occurs when multiple developers check out their own *working* versions from the same object, usually to work on different sections of code.

Term	Definition
parallel release development	Occurs when an organization needs to produce multiple releases of its software product simultaneously (e.g., different developers working on the next release, a patch to the current release, and a maintenance release, all using the same baseline).
parallel variant development	Occurs when multiple developers are working on different versions of the same object for different hardware platforms (often called *variants*).
parallel versions	Two or more objects checked out from a single object.
platform property	An identifier that designates a project or object for a particular hardware platform.
prep projects	See build management projects. *See* "build management project" on page 47.
privileges	Privileges determine what operations you are allowed to perform in a database, and may be different for each database in which you work. Users never need to change privileges manually; when a user issues an operation, Rational Synergy determines whether the appropriate privilege for the operation is available.
process	Groups process rules into a named set designed to work together. A process is used to specify the process rules you can use for a release.
process rules	Patterns that define how projects are updated. They specify rules for determining the baseline plus a set of tasks to be used when someone updates a project. In previous releases, "process rules" was referred to as "update template" and "reconfigure template."
product	Files that are built by processing other files.
product task	A task automatically created by Rational Synergy to manage a product.

Term	Definition
project	A user-defined group of related files, directories, and other projects (called *subprojects*). A project normally represents a logical grouping of software, such as a library or an executable, and it contains the directory structure of the files.
project grouping	Rational Synergy groups projects by purpose and release, for example, **My 1.0 Insulated Development Projects**. This is called a project grouping. Additionally, project groupings contain the tasks and baseline used when a project is updated.
properties	The properties given to an object. The basic properties of an object are those items identified by its four-part name (**name**, **type**, **instance**, **version**) and also include **owner**, **status**, **platform**, and **release.**
purpose	A setting that specifies a project's state and maps it to a process rule for the project's release to ensure that it selects the right members when you update a project.
reconfigure properties	*See* "update properties" on page 53.
release property	An property that identifies a project or task that is specific to a particular release.
release	A release consists of an optional component name and release delimiter, and a component release. The component name might represent the name of an application or component, such as **Rational Synergy** or **editor**. The component release identifies the specific release of that application or component. **Rational Synergy/7.1** is an example of a release.
released project	A version of the software that has been released or has reached a milestone.
released state	A state given to objects that have been released or have reached a milestone.
roles	*See* "privileges" on page 51.

Term	Definition
shared state	A state given to projects that can be modified by multiple users.
state	Defines an object's characteristics, such as its stage in its lifecycle, and the actions that can be performed, such as who can modify it.
subproject	A project contained within another project.
synchronize	An operation a developer uses to compare his work area with the database and update it selectively.
synopsis	A task's name.
system testing projects	Projects that contain the versions of the files, directories, and products used for system testing and release preparation.
task	A grouping of all the software modifications needed to complete a logical change in a software application.
task_assigned state	A state designating a task that has been assigned to a developer.
task-based methodology	A methodology that enables a development organization to track changes to a software application using tasks, rather than individual files, as the basic unit of work.
Rational Synergy database	A data repository that stores all of your controlled data, including source and data files, their properties, and their relationships to one another.
uncontrolled product	A product that exists in your work area, but not in the Rational Synergy database.
update	A process that automatically updates a project's contents with the most recent versions of its member objects. Each object version in the project or directory is evaluated, and the appropriate version is selected from the available candidates in the Rational Synergy database.
update properties	Properties that a project uses to decide which object versions to select when someone updates a project. (In previous releases, "update" was called "reconfigure.")

Term	Definition
variant project	A platform-specific variation of a product.
visible state	A state given to source objects so they can be used, but not changed, by other users.
work area	A location in the file system into which Rational Synergy writes a project when a user copies a project for personal use.
working project	*See* "development project" on page 48.
working state	The state of a developer's personal copy of an object. This is the state in which a developer makes changes.

Appendix: Notices

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send written license inquiries to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send written inquiries to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106-0032, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:
INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Some states do not allow disclaimer of express or implied warranties in certain transactions. Therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

Intellectual Property Dept. for Rational Software
IBM Corporation
1 Rogers Street
Cambridge, Massachusetts 02142
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at Copyright and trademark information.at `www.ibm.com/legal/copytrade.html`.

Microsoft, Windows, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product or service names may be trademarks or service marks of others.

Index

A

Application Developer, 12
assign tasks, discussed, 34
associated objects, defined, 14

B

baseline
    and process rules, 24
    and project groupings, 21
    and update, 25
    defined, 47
    discussed, 24
    project, defined, 47
    project, discussed, 24
benefits of configuration management, 8
build
    defined, 26
    see also makefile
build management project, 21

C

candidate, defined, 22
check in, 16–17
check out
    defined, 16
    specify release, 30
completed state, defined, 20
concepts, discussed, 13
configuration management
    benefits and features, 8
    goals of, 7
copy an object, discussed, 24
copy project and specify release, 30
create object, 24
current task, discussed, 18
cut an object, 24

D

data repository, discussed, 13
database
    defined, 13
    how sync affects backup, 23
    sync with work area, 22, 23
    transferring information to, 10
definitions
    discussion, 13
deleting an object, 24
development
    project, and tasks, 35
    project, defined, 21
development process for projects, 35
directory entry, 22
directory, defined, 22
Distributed Configuration Management (DCM), 10

E

Eclipse, 12

F

features of configuration management, 8
folder
    add tasks to, 25
    defined, 25

G

GUI (graphical user interface), 2

H

history object, described, 18

I

IBM Customer Support, 3
integrate state, 19
integration test cycle, 35
integration testing project, 29, 35

integrations
Eclipse, 12
Microsoft Visual Basic, 12
Microsoft Visual C++, 12
Microsoft Visual Studio, 12
Sybase Powerbuilder, 12
integrations for Windows development, 12

L

lifecycle
object, 19
task, 20

M

merge parallel object versions, 17
methodology, task-based, 27
Microsoft Visual Basic, 12
Microsoft Visual C++, 12
Microsoft Visual Studio, 12

O

object
check in, 16–17
check out, 16
create, 24
create new version, 16
cut, 24
defined, 13
delete, 24
four-part name, 16
full name, 16
history, 18
lifecycle, 19
make available to other users, 16–17
modify, 16
multiple locations of, 16
paste into project, 24
properties, 18
spec, 16
state, 19–20
update, 24
use, 23
version history, 18
version, and four-part name, 16
version, defined, 13

P

parallel development
concurrent, 41
for different platforms, 41, 42
for different releases, 41, 43
property, 41
types, 41
personal project version, 35
platform property, 15, 18
privileges, defined, 51
process rules, defined, 25
product, defined, 26

project
build management, 21
defined, 20
development, 35
development process, 35
development, defined, 21
how versioned, 20
integration testing project, 29, 35
modify objects in, 23
personal version, 35
purpose, 32
released, 21, 29, 38
subproject, 21
system testing project, 29, 37
update, 24
workflow, 28–33
project grouping
defined, 52
discussed, 21
properties of an object, 18
purpose, project, 32

Q

QA test cycle, 37

R

release
name, defined, 30
prepare for next, 38
property, 15, 18
specify when copying a project, 30
release, example, 31
released
project, and versions, 21
project, workflow, 29
state, and baseline, 38
state, and lifecycle, 19

S

states of an object, 19–20
subproject, 21
subsystem property, 15
Sybase Powerbuilder, 12
sync
database and work area, 23
work area and database, 22
Synergy Classic interface, described, 2
synopsis, 14
system test cycle, 37
system testing project, 29, 37

T

task
and development project, 35
assign, 34
associated objects, 14
current, discussed, 18
defined, 13
developers' process for modifying, 34
lifecycle, defined, 20
relationship with object, 14
use of, 34
task_assigned state, defined, 20
task-based methodology, described, 27
terms
discussion, 13
test cycle
integration, 35
QA, 37
system, 37

U

update, 24
object, 24
project, 24
projects and objects
see update properties
update properties
defined, 33
how objects are evaluated, 33
use an object, 23

V

version
of object, and four-part name, 16
of object, defined, 13

W

Windows development, integrations for, 12
work area
defined, 22
sync with database, 22
workflow
and projects, 28
working
state, discussed, 19