Using ClearQuest ALM

Contents

Using Rational ClearQuest ALM 1
Overview 2
ClearQuest ALM record types 4
Roles in an ALM process 10
Why use an ALM schema? 15
Getting started for administrators 16
Configuring and using ALM. 17
Working with projects 20
Project security 21
Project settings 22
Data and project imports 26
Prerequisites to creating an ALMProject 26
Creating a project 27
Project process 33
Copying an existing project 36
Managing work 37
Overview of ALM work process 38
Working with roles and process 39
Tracking work and builds 44
Iterations 45
Building and maintaining baselines 45
ALM with UCM. 47
Using ALM without UCM 53
Testing 54
Accepting or rejecting a request 55
Sample data 56
OpenUp 57
Mandatory fields for ALM record types 57

Index 59

Using Rational ClearQuest ALM

The application lifecycle management (ALM) packages and ALM schema provide out-of-the-box solutions for ensuring ALM best practices in your new or existing Rational® ClearQuest® change management system.

The ClearQuest ALM packages provide you with a role-based process and security model that offers optimal performance with a collection tightly integrated record types that align with the roles most common in development organizations.

ALM is the coordination of development activities to produce software applications or components, and supports the lifecycle management of assets and their relationships. ALM helps facilitate processes for developing software that span multiple roles while managing all of the content that is produced by each role. It supports team members that may be distributed around the world who need to collaborate. The results of their work must be traceable to the originating request for change. Tasks are automated, and the work is governed to assure completion and quality.

The foundations of ALM are:

- Governance - Continuously assessing progress and results
- Traceability - Managing dependencies and impact
- Collaboration - Connecting stakeholders and contributors
- Automation - Improving performance, compliance, and quality
- Distribution - Connecting the software delivery chain

For example, a requirement or product request can impact the design, development, build, and testing of an application. A change can impact every member of a team. Each role in a work process may produce content that contributes to the design, implementation and testing of that requirement. Understanding and managing the amount of effort involved to satisfy each requirement is critical for a team to be able to deliver on time or under budget. Project managers must ensure that requirements are implemented and tested with sufficient quality before a solution can be delivered. Software development teams must create artifacts (source code, requirement, or test case) and understand the relationships between different artifacts.

The ClearQuest application lifecycle management (ALM) feature is delivered as a set of two packages, or equivalently, as an out-of-the-box schema, with sample data you can start with by importing, and scripts that support the ClearQuest UCM integration and Project cloning.

The ALM packages provide an out-of-box solution to help teams manage the work involved in delivering software projects, and capture ALM best practices which can be used as is, or extended and applied to existing ClearQuest implementations.

- The **ALMProject** package provides project context for coordinating a team's work, including role-based security. The project package includes roles, record access per project, and project planning capabilities.
- The **ALMWork** package provides the capabilities to govern and trace project health through work process management.

ALM provides support for a variety of application development processes by facilitating role-based and data-driven process customizations. The comprehensive collection of supplied record types aligns with specific workflow roles and include the appropriate actions for that role. This support provides greater performance benefits and better enables parallel development for an organization's change management system.

Work can be assigned to team members that are either co-located or distributed. That work is traceable to the original request, and traceable to the project that implemented the request. Projects define a context for completing work and can be secured by setting security policies and defining roles.

By using an ALM schema, all work can be organized by a Project and each change request represented as a Request. Tasks address Requests, and Activities are units of work that are worked on and completed to complete a Task. Parallel development is enabled by multiple Activities that can be associated with a Task. Each Activity can be assigned by a Role (such as Dev, Test, and Doc). The Request can be assigned to a team member based on Request type and a Role. Moreover, because a user may fall into one or more roles (such as submitter, developer, or tester), an ALM change management process allows for easier transition between the roles.

The ALM schema provides support for the following areas of change management and workflow process:

- Facilitates team coordination of work throughout the software lifecycle.
- Allows for project-specific configuration without modifying the underlying schema.
- Adds project-level security.
- Provides role-based access to projects that is easily managed by project managers
- Supports globally distributed development (GDD) teams.
- Provides solutions scalable from small teams up through large enterprises.
- Allows for maintaining audit trails so that teams are accountable.
- Supports but does not require Multisite and UCM-enabled systems.

You can apply one or both ALM package types to an existing schema without impacting your current teams or record types. All ALM record types in this schema are prefaced with **ALM** to help distinguish them from other records in your current schema.

Overview

An ALM schema provides the capabilities for security, governance, traceability, and workflow management.

The ALM schema relies on some primary, interrelated record types for ensuring that requests for change are assessed, assigned, tracked, worked on, and resolved with a process that ensures complete traceability and tracking.

The primary record types are ALMRequest, ALMTask, and ALMActivity. An ALMRequest record represents a request for some type of change. An ALMTask addresses a Request and helps manage the Activities that are the units of work to complete the Task.

An ALMTask can be used by a manager or team lead to manage work to be done and the resources to be allocated for all Activities.

Each of the ALM work record types has a **Type** field that can be used to describe a work type (such as Defect, Enhancement, and Release requirement). Although the ALM packages include supplied **Type** values that are common for software development best practices, you are not limited to them, and create your own types.

Relationships between the record types can help facilitate the processes for managing change across multiple roles. For example, when a Request record **CreateTask** action is executed, one or more Tasks can be created. If Tasks have already been created for this Request, you can specify that the same or a different set of one or more Tasks be created with Task types that are the same as the Request types, or different ones. When a Task record **CreateActivity** action is executed, one or more Activities of different types can be created. If Activities already exist for that Task, you can create a different set of Activities. The flexibility to customize the work process is available but optional.

Each work type can have specific user roles associated with them (for example, associate Test role with Test Activity). Each role lists the team members who are allowed to perform that type of work.

An ALM work process begins with a request:

1. A user submits a request. The change request could describe an enhancement request, a release requirement, or a defect.
2. A triage team or change control manager reviews the request and accepts or rejects it. If they accept it, they create a task, which is a high-level description of the work to be done to implement the request. The request record includes a link to the task record, and the task is assigned to a project.
3. A lead developer or other team lead reviews the task and then activates it. Activating the task creates activities to complete the task. The team lead assigns these activities to team members. Examples of activities are Development activities; Test activities; and Documentation (Doc) activities. The task record includes links to the activity records, and the activity records have links to the task.
4. Developer, Test, and Doc leads assign their activities to team members who update the activity records to reflect the status of their work. When they finish work they deliver their changes and mark their activities Complete.
5. A release engineer integrates and builds delivered changes, and creates baselines.
6. A tester tests changes in the baselines. The Test lead marks a test task Complete, after the test activities are worked on and completed.
7. The user who submitted the request reviews the task and its activities and, if satisfied, marks it Complete.

This process helps provide the traceability from an initial user request to all activities required to satisfy the request.

The common kinds of roles for an ALM schema are:

- Submitters. While not a defined ALM role, a submitter vcan be anyone, for example, any support engineer, developer, tester, technical writer, or manager. A submitter can:
  - Submit requests.

  - Check on request status.
- Development or project managers or team leads. These roles can triage requests and identify release targets. A manager can:
  - Check on request status and close as appropriate.
  - Check to see if the workload for developers is balanced appropriately.
  - Run reports (request metrics, find, close, incoming, status of release).
- Doc Assessor, Tester, and Developer These roles:
  - Find requests assigned to them.
  - Work on and resolve requests.
- Support or product manager. These roles:
  - Run reports (request metrics, find, close, incoming, status of release).
  - Check on request and release status.

While a user may fill multiple roles at any given time, an ALM schema allows for a more clearly defined transition between the roles. For example, the same user, who is a Developer, can submit a Request and then assign an associated Activity to him/herself, and resolve it. In this example, the same user is the Submitter, Developer Lead, Developer, and Tester.

ClearQuest ALM record types

The set of records in the ALMProject package can provide the project context and work process in which work occurs.

The ALMWork package provides Request, Task and Activity record type sets that you can use and configure to help govern work processes. You can use these record types as is, or use them to create your own set, or types, of Request, Task, and Activity records based on the sample data.

While the Request describes the request, or issue, and is owned by a submitter, Tasks and Activities are the records that track the work that is assigned and must be completed in order to resolve the Request.

- Request: A record that reports a request. All work on requests for change begins with a Request. A Request can have one or more Tasks associated with it.
- Task: A record that associates a Request with a target release. Multiple Tasks can be associated to a Request. Each Task can have one or more Activities associated with it.
- Activity: A record that captures activities performed in support of a Task. Multiple Activities can be associated to a Task. Each Activity is assigned based on a Role.
- Comment: A device for communicating with the record owner. Comments can be associated with Requests, Tasks, and Activities, and are used for Questions, Comments, and Responses.

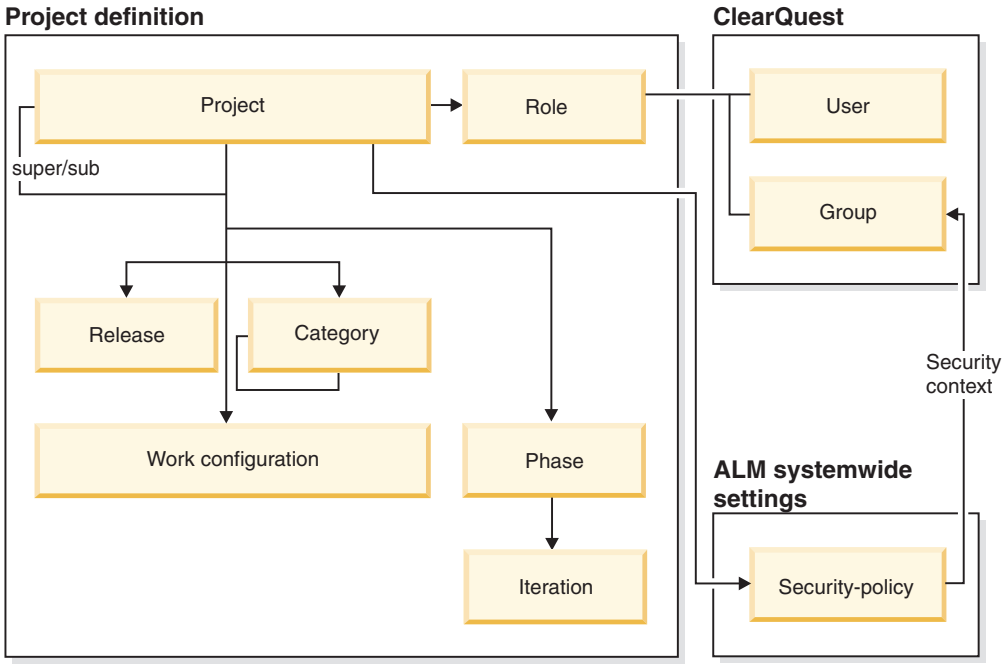
Projects

The ALMProject package provides the context for managing all work in your change management system.

All work in the ALM schema is organized by a Project. The Project provides both the context and role-based security model for your work. The other record types in a project definition are used to define the context within which work occurs. They are built into the ALM schema, and are also provided in the ALMProject. package.

Security is an important aspect of all project-based work. In the ALM schema project security is defined by who has access to the project, and what they can do.

You create a security policy to define what users have access to a project. Security is set on a project by project basis and is required for each project. The security policy is inherited by all the records related to a project.



Roles are used to define which users or groups can perform which actions for a project. You can view the project roles and define new roles for the project by creating a new **Role** record type. You must also assign Users (or Groups) to a role to have access to the project.

In order to manage several projects, you can use **Category** record type to classify a project, and the **Release** record type to identify the version of the software the project delivers. The Category helps to classify the product, feature or component that the project delivers, and the Release identifies the version. For example, for a Project named **CQALM**, you could set the Category to **ALM** and set the Release to **7.1.0**. These three identifiers define the uniqueness of the project. A later project could be Project=CQALM, Category = ALM, and Release=7.2.0, where only the Release number has changed.

Projects often have relationships to other projects. These relationships can be established as **Related Projects**. Large projects can define smaller sub-projects with links between them by using the **Super** and **Sub** project fields. You can also manage projects that track many revisions to the same product or component using the **Prior** and **Next** project fields.

You can divide a project into Phases and Iterations, by using the **Phase** and **Iteration** record types. You can view the phases and iterations for a project on the **Plans** tab.

A **Work Configuration** record allows project managers to establish a customized work management process on a project by project basis. A Work Configuration record for a project helps determine what work types are used by the project. The

ALMWork package Request, Task, and Activity record types help manage work and create more meaningful queries regarding the completion of planned work.

Note: Once you add ALM packages to an existing database, you may want to prevent users from being able to create new change requests that are in an older non-ALM Project, and are not ALMRequests. To do so, you must establish an access control or other mechanism to prevent it from occurring. By default, users cannot submit ALMRequests until an ALMProject is first created.

Requests

All work begins with a Request. All requests for change begin with a Submitter who submits a Request record.

In the ALM work process, the Request record is the starting point in the workflow process. The Submitter is responsible for submitting the request, periodically checking its status, and responding to any questions or comments. Any stakeholder can submit a request.

A Request:

- Contains information related to where and how the request was found.
- May include a Request type and category that can help identify and resolve the Request. Request type examples are Enhancement, New Feature, and Defect.
- Is mastered at the Submitter's site.
- Is owned by the submitter.
- Can have Comments (and questions) added to it.
- Contains references to all Tasks associated with it. The Tasks contain information about where the request will be fixed.

The ALM work process starts with a Request in an Opened state. When the solution is accepted, the Request record is moved to a Completed state. The Submitter creates the Request record. The record is then in the Opened state. The Submitter can withdraw the Request, or review the associated Task status. If the tasks are completed, and the Submitter agrees with the resolution, then the Accept action can be selected to move the record to the Completed state.

If the Submitter does not agree with the resolution, then selecting the Reject action moves the record into the Rejected state. A Reviewer can reconsider the state of the request if new information is available, and then accept the new information, by selecting the Accept action to move the record to the Completed state. Similarly, a Submitter can withdraw a request by selecting the Withdraw action and moving the record to the Withdrawn state. A Submitter is allowed to reopen the request from the Withdrawn, Completed, or Rejected states.

Submitters review the state of the associated (child) Tasks. If the Submitter agrees with the solution provided (for example, a related Task with a state of Completed and resolution code of Fixed), then the Submitter would select the Accept action to move the record to the Completed state. The Completed state is the final state.

Triaging a Request

The Development or Project manager role for the ALM work process is concerned with triaging the requests and identifying the release targets. This role may be represented by engineering, product and project managers, testers, or documentation. The responsibilities of this role are to assess a Request, and then to address the Request by creating Tasks to track the resolution. Once a Task is

Activated, Activities are created and assigned complete the Task. Activity records are child records of the Task record, just as the Tasks are child records of the Request. A Request can have one or more Tasks associated with it.

Tasks

Tasks resolve Requests.

The team that is associated with the project category that a Request references reviews the Request and determines if it can be addressed in the project. If the Request will be addressed, a Task is created and associated with the Request. Anyone looking at Request can see which Tasks are needed to fulfill it. Anyone looking at the Task can see which Request it is intended to complete.

A Task record:

- Is owned by a member of the project team.
- Contains all information about where the Request will be addressed.
- Represents an aggregation of work activities spread across multiple team members.
- Includes references to all associated Activities that are required to complete the Task.
- Is mastered at the owner's database replica site. (For example, a record in the Opened state might be first mastered at a Developer Lead's database replica site and then be mastered at the Tester's site when the Task is in the Activated state.)
- Refers back to the Request which the Task is intended to complete.
- Allows for comments and questions to be added to it.

In the primary flow for a Task, the Task starts in an Opened state. The Task record is moved to the Activated state when work is being done. When a resolution is accepted, the record is moved to a Completed state.

Managing a Task

Typically, some lead role (which may be the same as the developer/project manager role), processes the workflow for a Task. In the basic work process, Activities that resolve the Task are created and then assigned.

The Task owner typically creates Activities that will complete the Task. The Task owner normally uses the **CreateActivity** action which creates a set of necessary Activities based on the WorkConfiguration record for the project. For example, the Activate action could automatically create a unique Activity for Development, for Testing, and for Documentation.

Tracking and closing a Task includes:

- Reviewing the state of any associated Activities. For activities in the Completed state, the engineering work is finished.
- Setting the Complete action to close the Task (Completed state).

Activities

Activities complete a Task.

Team leads review Tasks and determine what Activities are needed to complete them. Often a single Task requires the work of multiple users. An Activity is created and assigned to a single team member to complete a discrete unit of work. An Activity:

- Is owned by an individual contributor on the project team.
- Represents a single unit of work.
- Refers back to the Task which it helps to complete.
- Can be enabled for UCM.

The Submitted state is the starting point for an Activity. Activity records are moved to the Activated state when work is being done. When the solution is complete, the record is moved to a Completed state.

Activity records are assigned to a person that is a member of one of the roles for that activity type. The owner reviews the Activity records assigned to them, and then completes work on the activities. Typical types of Activity records are:

- Assess Results
- Build
- Create
- Test Case
- Define Requirements
- Define Vision
- Design Implementation
- Design Solution
- Detail Requirement
- General Implement
- Implement Developer Test
- Implement Test Script
- Manage Iteration
- Outline Architecture
- Outline Requirements
- Plan Project
- Plan Iteration
- Refine Architecture
- Review
- Run Developer Test
- Run Tests
- Test

Developers work on Opened state activities that are assigned to them. Review or assessment activities are for reviewing or assessing information in a developer Activity for an indication of whether other work, such as documentation, is needed. If additional work is needed the reviewer or assessor creates development activities to resolve the need.

Test activities are for Testers to test and qualify Developer activities. A Test Activity contains all testing information and results.

Related concepts

"ALM with UCM" on page 47

Comments

Comments, questions, and responses can be added to any type of Activity.

The Comment record type serves as a Question and Answer device that supports communications between team members relating to an associated Request, Task, or Activity record.

The Comment records provide an alternative to the traditional ClearQuest notes package where communication is logged into one long, continually expanding note field. The advantage of the Comment record is that it allows distributed records to be updated without needing mastership and without concern of competitive update, and provides a way to request a change where you do not have the right of Ownership. It also enables a user to decline a Request.

Each Comment is mastered at the site where it is submitted and not at the site of the submitter's (or author's) mastership.

Creating duplicates

You can use a Comment record to mark a record as a duplicate.

For example, the process of indicating that one Request is a duplicate of another is by creating a Comment. The **Comment** requests the **Owner** of the Request record to consider their Request to be a Duplicate of another. The **Comment** indicates the ID of the Request that the user adding the Comment thinks is the correct Request to remain active. The Owner of the Request may decide to withdraw their Request in response. Or they may decide that they do not agree with the suggestion that their Request is a Duplicate by Responding to the Comment that suggested duplicate status.

Baselines

A Baseline lists the Activities that have been completed and delivered.

A Project Release Engineer (or Builder) creates a baseline of source code files and also creates builds. They may also validate and promote Builds.

You create a new ALMBaseline record when one or more ALMActivities are completed and delivered.

"Builds"

"Creating baselines" on page 46

"Tracking work and builds" on page 44

Builds

A BTBuild record has a reference to the ALMBaseline used to create the build.

A Release Engineer uses the latest baseline to (or Builder) create, run, validate, and promote a build. The ClearQuest ALM solution helps translate the build process to an installer, user, or tester of a build with the ALMBaseline and BTBuild record types. These record types help form the work process by including information that identifies the actual code baselines and builds.

"Baselines"

"Creating a build record" on page 46

"Tracking work and builds" on page 44

Roles in an ALM process

The ALM packages provide support for a role-based change management (CM) system design through a collection of record types that apply to role-based development.

Your work process determines what actions are available for each record type (such as **Project**, **Request**, **Task**, **Activity**, **Baseline**, and **Build**). For example the following roles might perform these actions:

- **Submitter** is a role that submits a Request.
- The **Triage** role triages requests, and can add and plan Tasks.
- The **Developer Lead** and **Test Lead** roles activate a Task, and assign Activities. They assign different kinds of activities for the **Tester**, **Doc Assessor**, and **Developer** roles.
- **Developer** is a role that can deliver a change, and complete an Activity.
- **Builder** is a role that can create a Baseline, run a Build, and validate and promote a Build.
- **Tester** is a role that can perform tests (as a kind of Activity), and complete an Activity.
- The **Test Lead** can review the status of and complete a Task.
- When a Task is completed, the original **Submitter** can accept a Task, if the completed Task resolves the original Request, or reject the Task.

Because a Request can be Completed or Withdrawn at any time with work still proceeding with the associated Task, or a Task can be Completed at any time regardless of the state of the associated Activities, only a manager or team lead user role should have the user privileges to Complete a Task once the associated Activities are Completed.

A Request submitter can be any user of the system and the submitter can Accept or Reject Request resolution. All users can also perform Comment record type actions.

When an Activity is Submitted, the mastership of it can transfer to the site of the person responsible for doing the next Activity action. Mastership is controlled by the ratl_mastership property of the Role Primary member included on the WorkConfiguration Activity record for each given Type. Also, if the Role Primary member is supposed to do the next action on an Activity they must have the action listed in the Role ApprovedActions to allow them to perform the action (for example, to Open an Activity if they are going to work on it). The OOTB sample configurations provided you with and enable you to use a system that incorporates these best practices.

Approved actions

Each user role has a defined set of approved actions.

Each defined user role in an ALM schema has a set of approved actions that help define the work for that role. The actions that are appropriate to each role are defined in the role label as the set of ApprovedActions. This list of actions includes the actions for each record type that the role may perform.

The Project Administrator role, with the Public Folders user privilege and as a member of the ALMAdmin record, can perform all defined actions for each record type.

The ALMProject SetDefault action is included in all role ApprovedActions sets. This action allows you to create Activities without first creating a Task, after viewing a Project and clicking the SetDefault action.

The SetDefault action sets the Project as a session variable for the user session. Once the session variable is set, you can create an ALMActivity. A DefaultTask is also created that is set as the Tasks value for the Activity. Once you log off, and then log back on, you must again click the Project SetDefault action. If you click the Project SetDefault action and then click the Task CreateActivities action, then this Task is added to the newly created Activities and not the DefaultTask that is associated with the Project (set by the Project SetDefault Action) that is added to the Activity **Tasks** field.

User roles

Users of a ClearQuest ALM user database work with specific types of tasks and activities based on roles.

Throughout the duration of a project, many activities must be performed to ensure that all required work is completed. The work records contained in the ALMWork package provide a way of keeping track of the tasks and issues that your team needs to address during the development cycle. The ALM schema provides Request, Task, and Activity record types that are the primary record types for managing work.

All work begins with a Request of some form. A Request can have one or more Tasks associated with it, and each Task can have one or more Activities associated with it.

- Request: A Request allows anyone to submit requests for change, such as a request for a new feature (enhancement) or to report a defect.
- Task: A Task commits a team to act on a Request within the context of a specific project. Multiple tasks can be associated to a Request. For example, many tasks may be required to complete a single Request. Alternatively, a single Request might be addressed in more than one project.
- Activity: Each Activity represents one unit of work that is assigned to an individual. Multiple Activities that are assigned to different user roles can be associated to a Task.

Each of these work record types can have more specific ALM work types defined to further refine the nature of the work.

While the Request describes the change request and is owned by any stakeholder on the project, Tasks and Activities are the records that track the work that is assigned and must be completed in order to resolve the Request.

In addition to work assignments, there are often questions or comments about the work. Therefore, a Comment record is also provided.

The Comment record provides a mechanism for communicating with a Request, Task, or Activity record owner. Comments can be associated with Requests, Tasks, and Activities, and are used for questions, comments, and responses.

Submitter:

Users of any role can submit Requests.

The submitter is responsible for submitting a Request, periodically checking the Request status, and responding to any questions or comments.

A submitter:

- Submits Request
- Checks on Request status
- Approves or Rejects a completed Request

"Submitting requests" on page 40

"Accepting or rejecting a request" on page 55

Triage:

The Triage role can be filled by engineering, product, project, test, and documentation managers.

The Triage role is responsible for evaluating requests, and creating tasks for further action. Engineering, product and project managers identify release targets. Test managers determine testing efforts needed on release targets. Documentation managers identify documentation changes needed. Technical Support managers provide Support perspective for issues.

The responsibilities of the Triage role are to triage Requests and to address the issues by creating Task records.

The Triage role:

- Triages Requests.
- Identifies where a Request will be addressed, or closes the Request.
- Creates ALMTask records.
- Checks on Task status and closes as appropriate.

The Triage role may also:

- Check to see if load balancing for their team's Activities is needed.
- Load balance Activities as appropriate
- Runs reports (for example, for metrics, finding requests, and current release status).

Possible approved actions for the release engineer role include the following actions:

- ALMRequest actions: Accept, CreateTask, Delete, DuplicateComplete Import, MarkAsDuplicate, Modify, Open, QuestionOrComment ReOpen, Reject_Solution, Withdraw, WorksAsDesigned
- ALMTask actions: Activate, Complete, CreateActivity, Delete, Import, Modify, Open, QuestionOrComment, ReOpen, Activate, Complete
- ALMActivity actions: Import, Modify, Open, QuestionOrComment, ReOpen, ReadyToWaiting, Submit, WorkOn

"Triaging requests" on page 40

"Managing work" on page 37

Dev Lead:

The Developer (Dev) Lead role can be filled by a development, testing, or other type of team lead.

The Dev Lead role is responsible for leading the development team, reviewing tasks and assigning development activities.

In an ALM work process, the Dev Lead role:

- Reviews Tasks
- Determines whether to address the Request or close the Task
- Creates Activities for Developer (or Tester or other roles such as Doc Assessor) role users.
- Reviews status of Activities

When a Dev Lead user Activates a Task:

- The Activate action automatically creates one ALMActivity record for Dev, one for Test, and one for Doc.

The **Create ALMActivities** button creates one ALMActivity for the Dev role.

Possible approved actions for the Dev Lead role include the following actions:

- ALMRequest actions: Accept, CreateTask, Delete, DuplicateComplete, Import, MarkAsDuplicate, Modify, Open, QuestionOrComment, ReOpen, Reject_Request, Reject_Solution, Withdraw, WorksAsDesigned
- ALMTask actions: Activate, Complete, CreateActivity, Delete, Import, Modify, Open, QuestionOrComment, ReOpen
- ALMActivity actions: Activate, Complete, Import, Modify, Open, QuestionOrComment, ReOpen, ReadyToWaiting, Submit, WorkOn
- ALMBaseline actions: Delete, Import, Modify, Submit
- BTBuild actions: Complete, Delete, Failure, Import, Modify, ReSubmit, Retire, Submit

"Submitting tasks" on page 41

"Activating tasks" on page 41

"Assigning documentation and testing activities" on page 42

Developer:

The Developer role can be filled by a developer, tester, or information developer.

The Developer role is responsible for developing a part of the system, including designing it to fit into the architecture, possibly prototyping the user-interface, and then implementing, unit testing, and integrating the components that are part of the solution.

The Developer role works on Activity records assigned to them and complete work on the Activities. Developer, Test, and Doc Assess roles all work on related Activities to complete a Task. The common types of an ALM Activity are:

- Developer: Developers work on and resolves issues. A Developer Activity contains a ChangeSet that includes the files being worked on, and an explanation of the Request resolution for when the Activity is completed.
- Test: Testers work on test Activities by testing and qualifying developer activities. Each test Activity contains all testing information and results.
- Doc Assess: Information developers work on a Doc Assess Activity by assessing documentation needs associated with a Developer Activity. A Doc Assess Activity contains an indication of whether documentation is needed as part of completing a Task.

Comments, questions, and responses can be added to any type of Activity.

The Opened state is the starting point for a Developer Activity. After reviewing an Activity, the developer can Complete it if the current behavior works as designed, is not going to be fixed, or for other appropriate reasons. The developer leaves the Activity in the Opened state if it cannot be quickly resolved and will not be worked on immediately, or activates the Activity to work on it. After completing work on the Activity, the Activity can be moved to the Completed state.

Possible approved actions for the Developer role include the following actions:

- ALMActivity actions: Activate, Complete, Delete, Import, Modify, Open, QuestionOrComment, ReOpen, ReadyToWaiting, Submit, WorkOn
- ALMBaseline actions: Modify, Submit
- BTBuild record actions: Complete, Delete, Failure, Modify, ReSubmit, Retire, Submit

"Querying for work assigned to you" on page 43

"Completing a development or documentation activity" on page 44

Release engineer:

The release engineer role creates baselines and builds.

The release engineer role is responsible for integrating developer changes, creates baselines from source code, and builds the solution.

Possible approved actions for the release engineer role include the following actions:

- ALMRequest actions: Accept, CreateTask, Delete, DuplicateComplete, Import, MarkAsDuplicate, Modify, Open, QuestionOrComment, ReOpen, Reject_Solution, Withdraw, WorksAsDesigned
- ALMTask actions: Activate, Complete, CreateActivity, Delete, Import, Modify, Open, QuestionOrComment, ReOpen
- ALMActivity actions: Activate, Complete, Import, Modify, Open, QuestionOrComment, ReOpen, ReadyToWaiting, Submit, WorkOn
- ALMBaseline actions: Delete, Import, Modify, Submit
- BTBuild actions: Complete, Delete, Failure, Import, Modify, ReSubmit, Retire, Submit

Tester:

The Tester role works on and completes test Activities.

The Tester role is responsible for the core activities of the test effort. Those activities include identifying, defining, implementing, and conducting the necessary tests, as well as logging the outcomes of the testing and analyzing the results.

The Tester role works on Activity records assigned to them and completes the work. Each Test Activity:

- Tests and qualifies a related Developer Activity.
- Contains all testing information and results.

Comments, questions, and responses can be added to any type of Activity.

The Submitted state is the starting point for a Test Activity. After reviewing an Activity, the Tester can complete it if the current behavior works as designed, is not going to be fixed, or for other appropriate reasons. The Tester opens the Activity if it cannot be quickly resolved and will not be worked on immediately, or activates the Activity to work on it. Once testing of the completed Activity is completed, the Activity can be moved to the Completed state.

Possible approved actions for the Tester role include the following ALMActivity record actions:

- Activate
- Complete
- Import
- Modify
- Open
- QuestionOrComment
- ReOpen
- ReadyToWaiting
- Submit
- WorkOn

The Test Lead role is responsible for leading the test team, reviewing tasks, and assigning test activities. Possible approved actions for the Test Lead role include the following actions:

- ALMActivity actions: Modify, Open
- ALMTask actions: Modify, Complete, ReOpen

"Testing" on page 54

Why use an ALM schema?

The ALM schema and packages provide a scalable change management workflow solution that captures the best practices of software development process design.

There is a significant performance benefit to using an ALM schema that relies on a set of record types, rather than relying on a more generic (and monolithic) record type. Each record type has a smaller set of operations associated with it, and less hook code. This set of record types offers fewer constraints to the maintainers of a system than broader and more generic record types.

In replicated environments a further advantage is that mastership issues are minimized, because roles-based record types tie record ownership and mastership to the change request resolution process so the correct owner and location always have mastership.

There are several advantages to using an ALM schema. For example, Administrators can identify and create a set of all supported product releases (or projects) that a customer can install such as:

- Full releases, fix packs, and other fixes or features.
- Bundles, offerings, collections, assemblies, common components, and shared common components.

Using an ALM schema and process can:

- Provide process consistency across the organization

- Support componentization work
- Provide consistent, reliable reporting of metrics
- Minimize mastership issues and reduces replication times
- Improve application performance
- Provide artifact or activity-based management
- Identify and utilize a role-based process model
- Support integrations or data feeds to other systems

ALM in a ClearQuest MultiSite environment

In addition to helping establish a role-based change management system, a common workflow can also help address mastership and replication requests in a distributed environment. For example, if a user logs in to a replica that resides in Bangalore, when the user creates a new Task, the Task is mastered where the Developer owner is located. The Task is created on the Bangalore site and then replicated. The initial Task mastership is determined by the default Developer owner, regardless of where the Request is mastered. Note that:

- A Request has to replicate to all sites to be visible.
- A Request contains references to the associated Tasks, and Tasks contain a reference to the Request.
- Each Task is mastered at the site of the default Developer owner once the Task is Opened. The Task is mastered at the site of the QE Lead once the Task is Activated.
- The Activity for an associated Task is mastered at the site of the Owner once it is Opened or Submitted. The Assigned Developer is the Owner of the Developer Activity. The default Developer owner can be determined by a Project, Component or Subcomponent field value. The default Doc owner is the owner of the Doc Assess Activity, and the default QE owner is the owner of the Test Activity.

The ALM workflow paradigm provides clear support for parallel, distributed development. For example, Tasks created to address Requests can be being reviewed for progress by Requestors when Developer Activities are Complete but not Tested or Assessed for documentation. Some Developer Activities can be Completed and others still Opened while Testing can be done on the work that is complete to date.

Sorting records in a MultiSite clan cannot be done in way that users running the same query at different sites see the same sequence of records if the sort fields on a query have more than one record with the same sort key or concatenated sort key values. For example, if you sort by Name and two records have the same Name then users at each site may not see the two records in the same sequence at both sites. If you use the record ID as a second sort field, note that the IDs are allocated blocks of IDs that may not reflect the order of the records being submitted. If you use the History filters (`History.action_name in ('Copy_Record, 'Import')OR History.old_state = 'no_value'`), you can get the first History record for any record for sorting to find the absolute sequence in which any two records enter a clan. You can use `History.expiration_timestamp IS NULL` to get the last History.Action.

Getting started for administrators

Administrators can define roles and processes for each project, as well as security policies to allow access to projects.

The main concepts for administrators to understand when working with the ALM schema or packages in ClearQuest are:

- Projects provide the context for managing work created by the members of the team. Users are granted access to projects through security policies, and their actions are defined by their role.
- Managing work is enabled in the form of requests, tasks, and activities. The definition of activities and tasks is driven by process definitions which can change from project team to project team.
- System-wide settings can be defined for managing projects and work. The ALM schema provides a set of system-wide settings that you can customize, without impacting the underlying the schema.

Configuring and using ALM

These topics describe how to set up and configure an ALM schema for users to follow an ALM work process.

Before creating an ALM Project, several records must already exist. There are also records that cannot be created until after the Project is created.

You must first create an ALMAdmin record. To create this record, you must log in as **admin**.

The ALMAdmin record controls who is allowed to create project configuration-related records. Users who are not listed in the ALMAdmin record do not have the project configuration record types listed in the **New** record menu list. If you add your own user ID to the ALMAdmin record, you must then logout and then re-logon in order for the change to be effective in the client's menu selection list.

After you create the ALMAdmin record, you can add either the individual users or ClearQuest user groups who are allowed to create projects. Once you add other users to the ALMAdmin list, then those other users can also add other users to the ALMAdmin record. The mandatory field in this record is **Members or Groups**. The ALMAdmin record's set of members can perform all actions on all record types.

If the ALMAdmin member is not a ClearQuest SecurityAdministrator, they only see Projects and other SecurityContext protected records secured by SecurityPolicies of which they are members.

The ALM sample database can be used to create projects based on the sample projects, **OpenUP Sample Project** and **Triage Sample Project**. To use the sample data, you must first log in as **Admin** and add a set of users to the ALMAdmin record. Then, you must add ClearQuest user groups to the SecurityPolicy records **OpenUP Security** and **Triage Security**. You can run the **All SecurityPolicy** query to find these records. See "Project security" on page 21 for information on how ClearQuest groups and SecurityPolicy records interact with projects and the team members who perform roles on the project.

Related concepts

"Working with projects" on page 20

Setting up

Setting up an ALM work process for users includes:

- Creating user accounts
- Creating user groups
- Creating security policies
- Creating role labels and roles
- Creating new projects

Before creating an ALM Project, you must first create several ALM records, including ALMAdmin, ALMSecurityPolicy, CategoryTypeLabel, ALMCategory, and ALMReleaseLabel record types. There are also related records for customizing your work process that cannot be created until after you create the Project record.

Related concepts

"Working with projects" on page 20

Applying packages to existing schema:

Before you can set up an ALM project or ALM work process to an existing schema you must apply the ALMProject and the ALMWork packages to it.

Use the Package Wizard to apply the ALMProject and ALMWork packages. For detailed instructions on applying packages to a Rational ClearQuest schema, see "How to Apply a Package" in Rational ClearQuest Designer Help.

When choosing what package to install, you can select ALMWork and get both packages installed, or you can select ALMProject, if you do not want the ALMWork package. You can also apply the packages separately by selecting and applying the ALMProject package, and then selecting and applying the ALMWork.

When you modify or customize an ALM schema, follow these rules so you can apply future ClearQuest version and ALM package upgrades:

- Do not delete or change any existing package fields and always name any new fields with names not likely to conflict with any field name added later by package upgrades.
- Do not modify or delete form tabs and always place any new fields on a new tab. Name new tabs with names not likely to conflict with tabs added by later package upgrades.
- If you add hooks to an ALM schema (for example, in order to record additional auditing information), use base action hooks rather than the placing the hooks directly into state transition actions. By using base action hooks, you do not need to add hooks to each individual state-transition action, since base action hooks are called for every action on the record type. However, your hooks may need to check the action type to see if the current action is a state-transition action (for example, before deciding whether or not the hook is necessary to record the audit information).
- When you perform a package upgrade, go to the schema version before the upgrade and open the schema and locate any ALM records that you have customized. Locate the new tab and select all form controls and copy them. Open the upgraded version of the schema, add the tab to the record form and paste the form controls you copied earlier.
- If you modify any current package attributes, keep a literal record of what you change and be prepared to reapply them or modify new functionality provided

by a package upgrade with any modified functionality. The cqload exportintegration command can be helpful for keeping track of the changes you make.

For example, to prevent naming conflicts, use a naming convention that includes a prefix added to new records, fields, actions, or tabs you create (such as ″C_″, ″UX_″, and ″XP_″).

See Naming restrictions in the Developing Schemas user assistance for more information.

After you apply a package to a Rational ClearQuest schema, check in the new version of that schema. For detailed instructions on checking in schemas, locate the topic ″Checking in a schema″ in Rational ClearQuest Designer Help. And, after checking in the new version of the schema, upgrade any user databases that used an earlier version of the schema with the new version that includes the ALMProject and ALMWork packages, or create new user databases. For detailed instructions on upgrading user databases, locate the topic ″Upgrading a user database″ in Rational ClearQuest Designer Help.

Related concepts

“Data and project imports” on page 26

Creating user accounts:

You need to create user accounts for the various roles that the ClearQuest ALM process uses.

If you are using the OOTB ALM schema you have the option of selecting sample data. You can create a user database and populate it with sample data. The sample data includes an ALMAdmin record and a SecurityPolicy record both of which reference users in your system. There should be one ALMAdmin record that initially has one member (admin). You can log in as admin and add any Project administrators to that ALMAdmin Members list. You can also clone a Project and some of the related records using the cloning utility.

In addition to being able to add other Members and Groups, ALMAdmin members and users in ALMAdmin groups are able to Add and update all Label records, Category, Project and Role records. They can also perform all Actions on all records regardless of their inclusion in Project Roles. Users listed as ALMAdmin record members or groups have the ClearQuest Security Administrator privilege and can add SecurityPolicy ratl_context_groups.

1. Click **Start → All Programs → IBM Rational → IBM Rational ClearQuest → ClearQuest User Administration** to open the User Administration tool.
2. Click **User Action → Add User**. Create user accounts for all users who need to access ALM records.

“Data and project imports” on page 26

“Copying an existing project” on page 36

Creating user groups and project security:

You need to create user groups and assign users to them as follows.

1. Click **Start → All Programs → IBM Rational → IBM Rational ClearQuest → ClearQuest User Administration** to open the User Administration tool.

2. Click **Group Action → Add Group**. Enter `Admin` in the **Name** field. In the Users field select **admin** and user IDs of other users who are permitted to import and delete records. Click **Add** to add them to the **Member Users** field.
3. Click **OK**.
4. Repeat the previous step to create a group named `ChangeControlBoard`. Add the user IDs of users who are permitted to create projects and create and modify records of the following record types that span projects:
   - Category
   - Project
   - ResolutionCode
   - SecurityPolicy
   - Record types whose name includes the suffix **Label**.
5. Create groups for SecurityContext as follows:
   - All records that reference a project must reference a SecurityPolicy record.
   - A SecurityPolicy record has a name and a reference to a user group.
   - The SecurityPolicy name is set on the History tab for all records except records supplied by ClearQuest packages other than the ALM package.
   - The user ID of everyone who needs to see and work on records associated with that record's SecurityPolicy must be included in the user group associated with the SecurityPolicy referenced by that record type (such as ALMCategory, ALMProject, ALMPhase, ALMIteration, ALMRole, ALMComment, ALMRequest, ALMTask, ALMActivity, BTBuild and ALMWorkConfiguration record types).

Related concepts

"Project security" on page 21

Related tasks

"Creating security policy" on page 27

Working with projects

All work in the ALM schema is organized by a Project. The Project can help provide the context, access control, and security model for your work.

These topics describe the ALM Project and the tasks involved in setting up a new project for defining categories, system-wide label types, and roles.

An ALM Project includes a collection of objects that comprise a Project definition, system wide settings, and existing ClearQuest User and Group administration.

- A Project includes Role, Category, Release, Work Configuration, Phase, and Iteration records to help provide context for work.
- Role records rely on existing User and Group administration.
- A security policy is based on a system-wide settings and existing security context controls.

An ALM Project is a unique combination of a Category and Release of that Category, where Category can be a Product, application, service, feature, or other type of component. The project must also have a unique name.

Projects may arranged into a project hierarchy using the **SubProjects** and **SuperProjects** fields. Projects can be chained together into sequential projects using the **PriorProject** and **NextProject** fields.

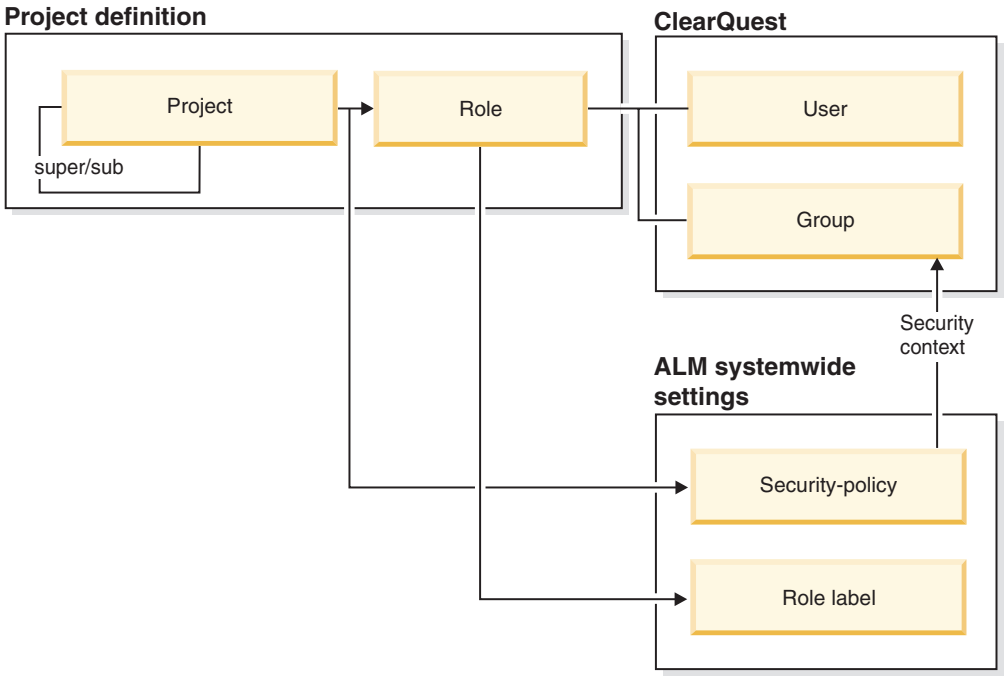
You can use project planning records to define iterative projects by using ALMPhase and ALMIteration records.

Project security

Security is an important aspect of all project-based work.

In an ALM schema project security is defined by who has access to the project, and what they can do. A security policy determines access, and roles determine what are the allowed actions for a user. The record types for defining the security-policy and roles for a project are:

- Project
- Role
- User and Group
- Security-policy
- Role Label



You define project security by following these steps:

1. Create users and groups. Existing ClearQuest deployments already have users and groups established. For new ClearQuest deployments see the ClearQuest Administrator user assistance for creating users and groups.
2. Create security policies. A security policy defines which users have access to the project. If you are a ClearQuest administrator, you can add one or more ClearQuest groups to a security policy record.
3. Choose a security policy. When creating a project, if you have more than one security policy defined, you can select a security policy from a drop-down list. Administrators can define security policies and allow project managers to choose a policy that best applies to their project. A security policy can be used by one or more projects. Security is set on a project by project basis and is inherited by all other records related to that project.
4. Create role labels. Roles are used to define which users or groups can perform which actions for a project. Many times an organization has a set list of role

names, such as Analyst, Developer, Architect, Tester. You define your roles by creating an ALMRoleLabel record for each role.

5. Create roles. While role names may be shared across an enterprise, the role definitions may change from project to project. Each project determines which roles are included, and which users perform each role, as well as the allowed actions for each role. You define a new role for the project by creating a ALMRole record.

Project settings

The ALM solution comes with a set of system-wide settings that provide the power to modify the solution to fit your enterprise without impacting the underlying schema.

Administrators can define categories of projects and label types that can help set policies for standardization, organization, and governance of projects. The Label records allow you to define a set of names which appear in the user interface, most often in the form of drop-down lists on records. Some examples of Label types are Work Type, Role, Resolution Code, and Category Type. These settings allow for reuse and consistent classification across multiple projects, and can adapt to the enterprise.

Over time the number of projects produced by an organization can be large. Project uniqueness and classifying features may be needed to identify projects. Additionally, large projects may be subdivided into smaller projects but share the same release version. Category and Release settings can be used to define project uniqueness.

You can define a set of Categories and Release Labels to help classify projects. The values you create can be used for any new project. Managers or team leads can use a Category to classify a project, and can use a Release to identify the version of the software the project will deliver. For example:

1. An ALMAdmin user:
   a. Creates ALM Categories. You can create a hierarchy of categories to classify projects. Projects are identified by a Category, which helps to classify the product, feature or component that the project delivers. In some organizations a need arises to create more than one category tree. For example, a single organization may identify projects using some combination of product and service. Category Types are used to identify the classification scheme. For example, using the example above, two CategoryType records are created, one for product and one for service. Once your category types are defined, you create categories for that type. Categories can be hierarchical. First create CategoryType. Then create the categories for that type.
   b. Creates Release Labels. A Release which identifies the version of the software. Many organizations standardize on a nomenclature for release labels. The ALM solution supports this need by providing the Release Label record.
2. A manager or team lead:
   a. Chooses a Category. When creating a project record, the available categories appear on a drop-down list. Choose a category that classifies this project.
   b. Chooses a Release. When creating the project, the available Release labels appear on a drop-down list.

For example, if you start with the out-of-the-box ALM schema that is included and you create a Project named *myALM*, set the Category to *ALM* and set the Release to *7.1.0.0*, these three identifiers define the uniqueness of the project. A later project could use the same category name except changing the Release value to 7.1.1.0. Project names cannot be reused.

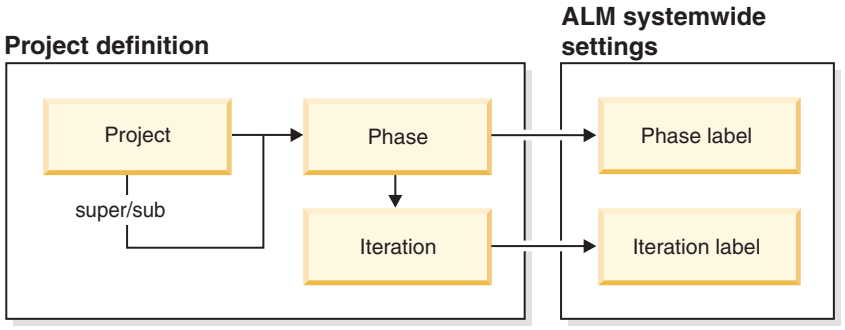
These topics describe system-wide settings, categories, security policies, and role labels.

Project planning

You can define iterative projects by defining phases and iterations to help ensure that work is completed

Projects can be divided into Phases and Iterations which are a planned, measured interval of time (for example, in weeks). The Rational Unified Process (RUP) defines four phases of a project: Inception, Elaboration, Construction, and Transition. Another example is to use Define, Design, Develop, and Deliver phases. Another example is to use Milestone1, Milestone2 , and so on, as phase names.

A Phase is divided into Iterations. Iterations can help focus a team on delivering incremental value to stakeholders in a predictable manner.



You must create at least one phase record for your project, even if your project does not require using phases. In this situation, you can name your one Phase record **Iteration**, and then add as many ALMIteration records as desired, and giving each iteration name a simple number as the name. An iteration name will then display as **Iteration 1**, **Iteration 2**, and so on.

You can start by defining the Phase and Iteration labels used in your organization. For example, the first construction iteration may be labeled C1 (representing Construction Iteration One).

When creating Phase records, choose a Phase label and choose the project (for example, PhaseLabel=Construction, StatusLabel=Pending). Create one Phase record for each Phase of your project.

You can create Iteration records for each phase.

The Phases and Iterations for a project appear on the **Plans** tab.

Record settings

You can use system-wide settings for record types to help manage, categorize, and apply security policies to work processes.

The following types of system-wide settings are for managing both projects and work:

- Category

  The project categories you define are available system-wide. This allows for reuse and consistent classification across multiple projects. CategoryType labels are also available system-wide. Categories can be secured by a Security Policy and so they are available or hidden from a particular user. Categories and category types allow you to model your classification system for projects. You could also define a set of hierarchical categories to decompose large systems into smaller more manageable units.
- Security Policy

  Security policies are defined by adding one or more ClearQuest groups to an ALM Security Policy record. Once set, project managers can create new projects and choose the existing security policy needed for that project. You only need to define a security policy if a new policy is needed.
- ALMAdmin

  The Admin record type determines who can create projects, categories, and labels.
- ALMType

  Types are used to identify the nature of the work. Types apply to Request, Task, and Activity records. You set the types system-wide. Project teams then configure which types to use by creating a Work Configuration. Some examples of a Type include, but are not limited to, **Enhancement**, **Defect**, and **New Feature**.

ALMSecurityPolicy records are associated with a Category, and are also associated with Projects, as Projects are created that reference the Category. For teams doing component development, there can be several Components, each with its own Categories and Releases, as part of one or more Offerings. In this case, a one to one relationship between a Category and a SecurityPolicy may cause some records to be not visible to people that need to see them. To prevent visibility issues, a SecurityPolicy should include either one large ClearQuest user group as its ratl_context_groups reference or it should have a user group for each component with all of the user groups referenced by the SecurityPolicy that would be shared by all of the development teams working on components. There are also performance benefits by maintaining a set of smaller groups rather than using one large group (or setting a SecurityPolicy to the Everyone group), and organizing groups and SecurityPolicy records by the structure of the components.

Component development example

Each versioned piece of new development work can be a Project with a Category that specifies the component and a Release that specifies the version of that Category.

A customer finds a problem in a major product produced by your development team. This Product (referred to as an Offering) includes several Components each of which is developed by separate teams. When the customer sees the problem, they think of the Offering as having the problem rather than the Offering's Components. When the team lead follows the triage process for a Request for that Offering and reviews the Request, they will note that:

- The problem is in a Component that is included in the Offering and needs to be fixed there not in the Offering which may be nothing more than a collection of Components and not have any of its own code apart from what comes in the Components it includes

- The Offering needs to include the new version of the Component once its been fixed and a new version of that Offering needs to be provided to at least the customer that discovered the problem and probably all other customers.

The triage team creates two ALMTask records that are associated with the ALMRequest entered against the Project for a given Category and Release (for example, Category='OfferingA' and Release = '1.0'):

- ALMTask with Project Category='OfferingA' and Release = '1.1'
- ALMTask with Project Category = 'ComponentZ' and Release= '3.4'

The triage team first reviewed the ALMBaseline record for Project Category='OfferingA' and Release = '1.0' because these values are what the ALMRequest identified as the FoundInProject, and they can see that the Release of 'ComponentZ' that is listed in the ALMBaseline ComposedOfBaselines field is Release = '3.3'.

Activities are created for the ALMTask for 'ComponentZ' and the solution is developed, documented and tested. An ALMBaseline record is created when the actual baseline is created for Project Category = 'ComponentZ' and Release = '3.4' and a second ALMBaseline is created for Project Category = 'OfferingA' and Release = '1.1' and that ALMBaseline record has a ComposedOfBaselines value (another Baseline record) that has a Project Category = 'ComponentZ' and Release = '3.4'.

A BTBuild is created for the ALMBaseline whose Project Category = 'OfferingA' and Release = '1.1'. Testers can see that a BTBuild is displayed in the Build column and the Composite.Build column of the 'Dev' Activity displayed in the Task's Activity Form control whose Project Category = 'OfferingA' and Release = '1.1'. They can see that there is at least the ID of a Build produced from the composite baseline and in the result set of the query they can see the name of that Build. Component testers and Offering testers can both see that there is a Build based on the Composite Baseline.

In the Composite Baseline record the Component is listed in the ComposedOfBaselines field.

"Projects" on page 4

"Project security" on page 21

Labels

Label records can be used to help organize and govern projects.

There are many instances where standardization across the enterprise is needed to help organize and govern projects. Label records enable you to define a set of names which appear in the user interface (for example, as drop-down lists on records).

Most Labels consist of a Name and a Description.

The following labels can be shared across the database instance:

- ALMCategoryTypeLabel: Categories provide the means to classify projects, and create a hierarchy of types of categories. By defining more than one category type, you can create hierarchies of categories belonging to one category type or another. For example, you can define category types such as Solution, Product, SOA Service, Re-Usable Component, and Business Unit, and create an appropriate hierarchy for these types.

- ALMPhaseLabel and ALMIterationLabel: Many processes, including the Rational Unified Process recommend dividing projects into Phases, where each phase can have one or more iterations. Doing so helps to divide a project into more manageable units. For example, the Rational Unified Process suggests for Phase labels: Inception, Elaboration, Construction, and Transition. An iteration is a planned, time-boxed interval typically measured in weeks. Iterations focus the team on delivering incremental value to stakeholders in a predictable manner. A Phase can also be used to define milestones for an agile development process. By using Phase and Iteration labels you can ensure consistent adoption of terminology across your organization.
- ALMReleaseLabel: Releases are used to identify the version of the software being developed. Some organizations standardize by using release names or numbers. You can use a Release Label record to identify a release label that will be used by others in the organization. For example, IBM uses a four digit release numbering scheme for all products, such as ClearQuest 7.1.0.0.
- ALMResolutionCodeLabel: When units of work are completed, a resolution code is set to provide a history and context for the type of resolution. For example, not all work is completed in a project. Sometimes there are duplicate requests, or a reported problem can not be reproduced or works as designed. You have the opportunity to define a set of resolution codes to be used by your organization.
- ALMRoleLabel: Role labels help to ensure a consistent use of roles across the organization. Analyst, Architect, Project Manager, Developer, Tester are examples of role names.
- ALMStatusLabel: A status label can be used to report the status or health of a Project, Phase, or Iteration. Some examples of a status are Healthy, Suspect, and Critical. Some organizations use color coding such as Green (for healthy), Yellow (for caution), and Red (for critical).
- ALMTypeLabel: Requests, Tasks and Activities vary from organization by organization, therefore each of these record types has a set of work Type labels that can be viewed in a **Type** drop-down list. The names that appear on that list come from the work Type record. You start by defining the work Type Label. The value is the name of the type (for example, Enhancement).

Data and project imports

If you add the ALMProject and ALMWork packages to your existing schema, you must perform the data imports which are provided as part of ClearQuest ALM. You may also want to perform the project imports which are provided.

If you are importing data from a nonALM database, you must create import files for that data. See “Mandatory fields for ALM record types” on page 57 for the mandatory fields for each record type.

Prerequisites to creating an ALMProject

You must first create one, and only one, ALMAdmin record and assign members to it who are allowed to create ALM administrative records.

After you have created an ALMAdmin record, you. or a user listed as a member in the ALMAdmin record, must create the following records before creating an ALMProject.

- ALMSecurityPolicy
- CategoryTypeLabel
- ALMCategory

- ALMReleaseLabel

ALMSecurityPolicy

The ALMSecurityPolicy record controls the visibility of records by using the Security Context feature in ClearQuest. This is also known as *record hiding*. A user listed in the ALMAdmin record can create this record, but only a Security Administrator can add to the Context Groups. The Security Administrator must add ClearQuest user groups to the Ratl_Security tabs Context Groups list. **Name** is a mandatory field, and **Context Groups** must be populated for the SecurityPolicy to be complete.

CategoryTypeLabel

CategoryTypeLabels classify what the purpose of the category might be. Examples are Product, Offering, Solution, Feature, and Component. **Name** is a mandatory field.

ALMCategory

An ALMCategory provides an association for a project that describes something about what is the goal or output of the project. Examples of categories are Product, Offering, Solution, Feature, or Component. The mandatory fields for ALMCategory are **Name**, **Owner**, **Security Policy**, and **CategoryTypeLabel**.

ALMReleaseLabel

The ALMReleaseLabel applies a name or version number to the output of the project. Examples are 1.0, Beta, and 1.1. **Name** is a mandatory field.

Creating a project

You can create a new ALMProject after the prerequisite record types have been created.

These topics describe the tasks you need to complete to set up an ALM project for your team.

Projects are governed by having various work processes defined and used by the teams working on the project. The process defines what are the types of work that must or could be done to satisfy the requests on the project.

"Project process" on page 33

"Work Configuration" on page 34

Creating security policy

The security policy is used to hide records from certain groups of users.

Only users who are members of groups listed in the SecurityPolicy record have visibility to records which are protected by the SecurityPolicy. The ALM schema uses the SecurityPolicy to protect ALMCategory, ALMProject, ALMPhase, ALMIteration, ALMRole, ALMComment, ALMRequest, ALMTask, ALMActivity, BTBuild and ALMWorkConfiguration record types.

Only users listed as a member in the ALMAdmin record can create a SecurityPolicy record. Only a user who is also has the SecurityAdministrator

privilege can set the Context Groups field of the SecurityPolicy, which is necessary for indicating which groups are allowed to have visibility to records which are protected by the SecurityPolicy.

Note: By default, the **Everyone** SecurityPolicy is added to BTBuild records that are created, so users that are not using ClearQuest ALM can still see BTBuild records they may be working with.

1. In the Rational ClearQuest client, click **File → New → ALMSecurityPolicy**.
2. Enter a name for the security policy in the **Name** field. Enter a description of the security policy in the **Description** field. Specify a user group. Select a security policy (for example, **OpenUp Security**).
3. Click **OK**.

Creating category type labels

A category type label is the name for a particular type of category.

Categories are used to classify and present an hierarchical structure for aspects of interest to your projects. What types of categories exist for your projects is completely up to your organization, processes, and resulting outputs from your projects. Typically, categories identify the architectural components of your project, but may be used to classify features of your product, or organizational boundaries. Example category types are Offering, Product. Solution, Feature, and Component.

1. In the Rational ClearQuest client, click **File → New → ALMCategoryTypeLabel**.
2. Enter a name for the category label in the **Name** field. Enter a description of the category label in the **Description** field.
3. Click **OK**.

Creating category

Categories are used to classify and present a hierarchical structure for your projects. In order to create a category, you must first create ALMCategoryTypeLabels. What types of categories exist for your projects is completely up to your organization, processes, and resulting outputs from your projects. Typically, categories identify the architectural components of your project, but may be used to classify features of your product, or organizational boundaries. Example category types are Offering, Product. Solution, Feature, and Component.

You create an ALMCategory record to specify a particular instantiation of a type of Category. For example, if you have an ALMCategoryTypeLabel named Product, then you might have an ALMCategory record named ClearQuest.

Categories can be arranged in hierarchies; a category may have a SuperCategory and multiple SubCategories. You create your category hierarchy to reflect the natural partitioning of classification scheme for your development deliverables, component or feature structures.

1. In the Rational ClearQuest client, click **File → New → ALMCategory**.
2. Enter a name for the category in the **Name** field. Select a category type label in the **Category Type Label** field. Select a category in the **Super Category** field.
3. Click **OK**.

Creating release labels

Release labels are the identifiers that you use to specify the deliverable results from your project. Most software projects use some form of numerical code to identify

specific releases of the software, but ALMReleaseLabels can be any name that you choose. Examples of release labels include 7.1.0.0, Beta, Release 1, and Hot Fix 2.

1. In the Rational ClearQuest client, click **File → New → ALMReleaseLabel**.
2. Enter a name for the release label in the **Name** field. Enter a description of the release label in the **Description** field.
3. Click **OK**.

Creating the project

Once the prerequisite record types have been created, you can create an ALMProject record.

An ALMProject record coordinates and controls the work efforts of a team to produce some work product of the project. It defines the parameters within which your team operates in the ALM process, including iterations, phases, team members, category, and release.

The mandatory fields for an ALMProject are **SecurityPolicy**, **Category**, and **Owner**. **Name** and **Release** are optional fields that can help identify project uniqueness. These fields are mandatory if a project already exists with blank or non-unique Name and Release values. You can also specify values for **SuperProject** and **PriorProject** if they exist and are relevant to your new project.

1. In the Rational ClearQuest client, click **File → New → ALMProject**.
2. Select the owner in the **Owner** field. Select the category associated with the project in the **Category** field. Click the **History** tab. Select a security policy (for example, **OpenUp Security**) in the **SecurityPolicy** field. Specify a Name and Release for your project.
3. Click **OK**.

You cannot set the default Task and Activity for the new project until after you submit the project so they can reference it. Once the project exists, you can configure it in more detail.

Creating role labels

Role labels identify the roles that users perform in the ALM process. You need to create the role labels before you create roles. The following list includes some of the possible roles in a work process:

- Analyst
- Developer
- Tester
- Project Manager
- Triage
- Dev Lead
- Test Lead
- Release Engineer
- Stakeholder

The OpenUp process in the sample database provide an example of a full set of roles.

The ALMRoleLabel record gives a name to a set of role capabilities, and lists a default set of ApprovedActions that should be allowed for any person who is fulfilling this role. The mandatory fields in an ALMRoleLabel record are **Name** and **ApprovedActions**.

1. In the Rational ClearQuest client, click **File → New → ALMRoleLabel**.
2. Enter the name for the role label in the **Name** field. Enter a brief description of the role label in the **Description** field. Click the **Approved Actions** tab. Select the approved actions for the role label you are creating. This table lists a set of possible ApprovedActions for some of the roles.

Table 1.

Role label	Approved actions
Project Manager	Iteration::Modify, Iteration::Submit, Phase::Modify, Phase::Submit, Project::Complete, Project::Modify, Project::Launch, Project::Postpone, Project::ResumeInProgress, Project::ResumePending
Triage	Request::CreateTask
Dev Lead	Activity::Modify, Activity::Open, Activity::Submit, Task::Activate, Task::CreateActivity,Task::Modify, Task::Reject_Task, Task::Unreproducible, Task::WorksAsDesigned
Test Lead	Task::Complete, Activity::Open
Developer	Activity::Activate, Activity::Complete, Activity:: ReOpen
Release Engineer	BTBuild::Submit, BTBuild::Complete, BTBuild::Retire, BTBuild::Failure, BTBuild::ReSubmit, Baseline::Submit

3. Click **OK**.

Creating roles

After you create role labels, use them to create roles.

The ALMRole record lists a set of people who can perform the role for the associated project, and describes the set of actions that these people are authorized to perform. The mandatory fields are **Project**, **SecurityPolicy**, **RoleLabel**, **Members or Groups**, **Primary**, and **Approved Actions**.

1. In the Rational ClearQuest client, click **File → New → ALMRole**.
2. Select the project associated with the role you are creating. Select the role label for the role.
3. Click the **Members** tab. Select the primary user for the role. In the **Members** field click **Add** to select additional users who operate in this role.
4. Click **OK**.

Creating phase labels

Phases identify the major stages of the application development lifecycle and may be divided into even smaller intervals called, iterations.

You can define iterative projects by defining phases and iterations to help ensure that work is completed.

Projects can be divided into Phases and Iterations which are a planned, measured interval of time (for example, in weeks). The Rational Unified Process (RUP) defines four phases of a project: Inception, Elaboration, Construction, and

Transition. Another example is to use Define, Design, Develop, and Deliver phases. Another example is to use Milestone1, Milestone2 , and so on, as phase names.

A Phase is divided into Iterations. Iterations can help focus a team on delivering incremental value to stakeholders in a predictable manner.

You must create at least one phase record for your project, even if your project does not require using phases. In this situation, it is suggested to call your one Phase record Iteration, and then add as many ALMIteration records as desired, typically just giving each iteration name a simple number as the name. An iteration name then displays as Iteration 1, Iteration 2, and so on.

The ALMPhaseLabel record gives a name to a phase. Some project planning processes have well established names, such as Inception, Elaboration, Construction, and Transition. Other possible names are Milestone and Iteration. The **Name** field is mandatory for an ALMPhaseLabel record.

1. In the Rational ClearQuest client, click **File → New → ALMPhaseLabel**.
2. Enter a name for the phase label in the **Name** field. Enter a description of the phase label in the **Description** field.
3. Click **OK**.

Creating phase

Create the phase based on the phase label that you previously created and associate it with the project.

The ALMPhase record is associated with a project and specifies for the project how the project timeline will be grouped into larger tracking units. The phase can also be used to track work for Milestones. If the project is only being tracked by iterations, then you can configure only one phase label, and name it **Iteration**. The iteration labels could then be iteration numbers. The mandatory fields are **Project**, **SecurityPolicy**, and **PhaseLabel**. The **Prior Phase** field is optional.

1. In the Rational ClearQuest client, click **File → New → ALMPhase**.
2. Select the phase label that you previously created in the **PhaseLabel** field. Click **Add** next to the **Project** field. Either enter a keyword and click **Search** or click **Browse** and navigate to a query to run to return a list of projects. Select the project to be associated with this phase and click **OK**.
3. Click **OK**.

Creating iteration labels

The ClearQuest ALM process assumes that your team uses an iterative development process where you include multiple iterations in the schedule. The team designs, constructs, and tests a working version of the project for each iteration. You need to create labels to identify the iterations in your development schedule.

The ALMIterationLabel record gives a name for an iteration of development work. The labels can be a set of numbers. You can specify a numbering convention to improve sorting, such as 01, 02, 03, ... , 10, 11, or I01, I02, ..., I10. The **Name** field is mandatory.

1. In the Rational ClearQuest client, click **File → New → ALMIterationLabel**.
2. Enter a name for the iteration label in the **Name** field. Enter a description of the iteration label in the **Description** field.

3. Click **OK**.

Creating iteration

Create an iteration based on the iteration label you previously created and associate it with the phase and project.

You can partition development work into iterative development cycles with an ALMIteration record being associated with a project, to describe how the project workload is being planned. You must create the phases before creating the iterations. If you want iterations without phases, you can create only one phase named Iteration, and then make the iterations be numbers. The mandatory fields are **Project**, **SecurityPolicy**, **PhaseLabel**, and **IterationLabel**. The **Prior Iteration** field is optional.

1. In the Rational ClearQuest client, click **File → New → ALMIteration**.
2. Select the phase label that you previously created in the **Phase Label** field. Click **Add** next to the **Project** field. Either enter a keyword and click **Search** or click **Browse** and navigate to a query to run to return a list of projects. Select the project to be associated with this iteration and click **OK**. Select the iteration label that you previously created in the **Iteration Label** field.
3. Click **OK**.

Modifying the category to specify current project

After you have created a project and associated it with a category, you may wish to specify that this project is the current project that is performing work for this category. The category has an optional field, **Current Project**, which is used to specify which project to assign an ALMRequest to as a default value.

You can use either of the following strategies to take advantage of this feature:

- Create one project which is used to collect all Requests that relate to a particular Category. This one project always exists for the lifetime of the Category, and you set the **Current Project** for the Category to this one project (and never change it).
- Change the Category's **Current Project** to select the project that is working on the current release of your Category. With this scheme, you must remember to update the Category's **Current Project** whenever your development work starts for the next release.

You can choose to leave the categories **Current Project** blank, in which case submitters of Requests must explicitly select a Project for their Request.

1. In the ClearQuest client Navigator view, navigate to the **Public Queries → ALM → Generic** folder and double-click the **All Category** query to run it.
2. Select the category that you previously created from the Query Results grid.
3. Click the **Modify** toolbar icon.
4. Click **Add** next to the **Current Project** field. Either enter a keyword and click **Search** or click **Browse** and navigate to a query to run to return a list of projects. Select the project to be associated with this category and click **OK**.
5. Click **Apply**.

Modifying phase to specify current iteration

As your project proceeds through its timeline, one iteration will complete and the next iteration of work will commence. You can keep track of which iteration is the

current iteration for a phase by specifying **Current Iteration** on the ALMPhase record. You can specify which Phase is the **Current Phase** on the ALMProject record.

1. In the ClearQuest client Navigator view, navigate to the **Public Queries** → **ALM** → **Generic** folder and double-click the **All Phase** query to run it.
2. Select the phase that you previously created from the Query Results grid.
3. Click the **Modify** toolbar icon.
4. Select the label for the iteration you previously created in the **Current Iteration Label** field.
5. Click **OK**.

Project process

You define and customize the work process for a Project by creating records for ALM types and work configuration.

These topics describe the tasks you need to complete to set up the work process for an ALM project.

Creating ALMType labels

The ALMTypeLabel records defines the names for types of work Activities, Tasks, and Requests. Examples of ALMTypeLabels are Defect, Analyze, Design, and Test. The OpenUp sample database provides several examples of ALMTypeLabels.

The **Name** field is mandatory.

To create an ALMTypeLabel:

1. In the Rational ClearQuest client, click **File** → **New** → **ALMTypeLabel**.
2. Enter the name for the type label in the **Name** field. Enter a brief description of the role label in the **Description** field.
3. Click **OK**.

Creating an ALMType

The ALMWork package provides the ALMRequest, ALMTask, ALMActivity, and BTBuild record types to record information about various kinds of work for a project. An ALMType record associates an ALMTypeLabel with one of these ALM work record types.

ALMTypes are used in conjunction with ALMWorkConfiguration records to define the work process for different types of project work. They are also used to customize FIELD_CHOICE_LIST values for certain fields of the ALMRequest, ALMTask, ALMActivity, and BTBuild record types without modifying the schema.

The mandatory fields are **ALMRecordType**, **TypeIndicator**, and **TypeLabel**.

To create an ALMType:

1. In the Rational ClearQuest client, click **File** → **New** → **ALMType**.
2. Select the **ALMRecordType** associated with the type you are creating. Select one of the possible TypeIndicators for the chosen ALMRecordType, and select the ALMTypeLabel for the ALMType.

The TypeIndicator is a qualifier on the ALMType record that helps to identify each kind of record type in the project. There are two types of values for TypeIndicator:

- The string, **Type**, which means that this is an ALMRequest, ALMTask or ALMActivity type (with a **Type** value such as Defect or Dev).
- The name of a field whose values you are creating. **Severity** and **Priority** are TypeIndicator field values that you can set for a record.

For example, **Severity** is a TypeIndicator for field values 1, 2, and 3 for a ALMRequest record type, and **Priority** is a TypeIndicator for field values 1, 2, and 3 for an ALMTask record type. A given ALMTask can have both a Type and a Priority. For example, a Task can have Type=Defect and Priority=1. In this example, the record has two TypeIndicator values.

3. Click **OK**.

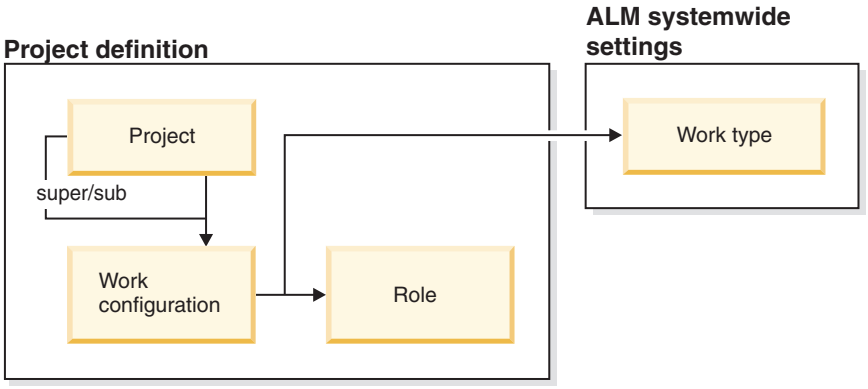
Work Configuration

A work configuration allows project managers to define a recommended process on a project by project basis.

A Work Configuration record defines the work process policies for a Project by specifying the set of activities for work to be completed. For example:

- A type of Request may need one or more special types of Task.
- A type of Task may have its own set of Activities.

The ALMWorkConfiguration specifies for a given project the type of work that is performed and the roles that can perform that work. The ALMWork package types of ALMRequest, ALMTask, and ALMActivities use the Project's work configuration records in order to control what type of work records should be generated by the CreateTask and CreateActivity actions, and various choice list selections on fields on these work records. The mandatory fields are **Project**, **SecurityPolicy**, **Record Type**, **ALMType**, **Roles**. The **Primary Children Configs** and **Secondary Children Configs** are optional fields.



Work Configuration records define what work types (ALMType) are used by the project. In this manner, process guidance can be introduced into a project by controlling the types of work that is performed to resolve a request, or to complete a task.

Each WorkConfiguration record can list a set of Primary Children configs and Secondary Children configs. These lists are used by the **CreateTask** action (on the ALMRequest record), and the **CreateActivity** action (on the ALMTask record). The

CreateTask or CreateActivity action creates a set of records that are listed in the Primary Children Configs the first time the CreateTask/Activity action is performed. Subsequent CreateTask/Activity actions use the Secondary children config list to create more records.

Using these Primary and Secondary children configs, you specify a set of Tasks to be completed for each Request type, and you can also specify a set of Activities to be completed for each Task type. For example, you can create a Task to initiate a Project. This Task might have Activities such as **Define Roles**, **Find Team Members**, and **Define Iterations**.

The ALM sample database for **OpenUP** illustrates how the work configurations can be used to implement the OpenUP process.

Example

Project A has a Request of type **Defect**. The Work Configuration is used to establish a rule that says, when a Request of type Defect is created, by default create a Task of Type **Defect**. Other Work Configurations for this project are created for each Activity type (**Develop** and **Test**), and another Work Configuration is defined for the Defect Task record. This work configuration creates a rule that states, when a Task of type Defect is created, by default create an activity of type **Develop**, and an activity of type **Test**.

Project B also has a Request of type **Defect**, and has a Work Configuration with a rule to create a Task of type Defect. However, the Work Configuration for the Task record is different. For Project B, the rule specifies to create activities of type **Design**, **Develop**, **Review**, and **Test**.

There are Primary and Secondary sets of Tasks created for Requests and Activities created for Tasks. The Primary set is the set created most often and is created the first time you create a Task for a Request or an Activity for a Task.

The Secondary set is created the 2nd-nth time you select **Request** → **CreateTask** or **Task** → **CreateActivity**.

There may be one or more Roles for a WorkConfiguration. The **Role** field is used for the following purposes:

- The **Role** → **Members** and Groups settings are used to determine the Choice List for the **Owner** field.
- The **Role** → **Primary** → **ratl_mastership** setting is used to set the default value for the Owner and is used to set the mastership for the Activity in the Submitted state if there is no default Owner.

Requests are associated with individuals, not with Roles. For example, the submitter of a Request may also be the Owner of the Request. Therefore the Role field on the WorkConfiguration record for a Request does not permit a Role to be set.

Creating resolution codes

Create a set of resolution codes that your team uses to indicate how they resolve Requests, Tasks, and Activities. Examples of resolution codes are Fixed, User Error, Unreproducible, Completed, Not a defect, and No Plan to Fix.

1. In the Rational ClearQuest client, click **File** → **New** → **ALMResolutionCode**.

2. Enter a name for the resolution code in the **Name** field. Enter a description of the code in the **Description** field.
3. Click **OK**.

Copying an existing project

Project administrators can use the project copy function available both in the Project Wizard and as a standalone utility in the ALMProject package, to create a new project based on an existing one.

The project copy function facilitates and speeds up the creation of a new project from an older project. The older project is used in this case as a template for the new project. For example, the next version of the same project will have very similar characteristics as its predecessor and most likely will be configured similarly. The copy function creates a new copy of the relevant records for the project, such as the Roles , Phases, WorkConfigurations, DefaultRequest, and DefaultTask records. Once there is a new copy of the project and its records, a project administrators can then make additional modification as needed for the new project.

An administrator can allow project managers to copy any project or can establish a best practice where template projects are created. By setting up example projects with all of the expected settings and configurations, you can provide guidance for project managers to copy from one of the templates (for example, if an organization has many projects that implement a packaged application).

The Project Wizard guides you through the options to choose to start with an existing project or create a new project. When starting with an existing project, the Wizard copies the Project and displays the current settings. If you make modifications to Project settings the Wizard updates the appropriate ClearQuest record types. If you start with a new Project, the Wizard creates all relevant ClearQuest records as you specify project settings. You can also perform project copy using the **Copy Project** button in a project record.

Before you copy a project, you must perform the following steps:

- Decide on a project name to provide for the new copy. The project name must be unique. If a project with the same name already exists, the Wizard informs you that the name is a duplicate and you have to type a new name for the project.

  A blank project name is detected by the copy function. To insure the copy succeeds, the copy function creates a unique name with the following format:

  `CopyOf_<SourceProjectID>_<unique integer>`

  where **SourceProjecID** is the ID of the project being copied and **unique integer** is the number of seconds returned by the Perl **time()** function. For example,

  `CopyOf_ALM00000292_1204732665`
- Create a new Category_*Name* and change the Category SecurityPolicy to SecurityPolicy_Category_*Name*. A category is required to copy a project.
- Create a new ReleaseLabel (for example, *ReleaseLabel01_Name* or *1.0*) or rename a ReleaseLabel (for example, *ReleaseLabel01_AA* or *1.1*).

  Any Category and Release must already exist before the project data can be copied. Also make sure the Category + Release name for the new project is unique. The **Copy Project** function will stop if the Category + Release name combination already exists as part of another project.

- Make sure all records that will be copied have valid data. Invalid values in various fields (for example as a result of a previous partially successful import) can result in copy failure or in partial copy of records. You can investigate the validity of data by opening a record with the **Modify** action, make no changes, and then clicking the **Apply** button. If the record saves successfully then the copy is successful.
- After copying a project, you must change some of the configurations for some of the cloned ALM records:
  - Set the Category record **CurrentProject**, Project record**CurrentPhase**, and Phase record **CurrentIteration** values.
  - If a new category is specified for the new cloned project, all ALMRole record SecurityPolicies are set to the **Everyone** SecurityPolicy. You may need to update the value for each ALMRole record SecurityPolicy on the cloned records as required by your policy.
  - Modify your cloned ALMRoles by changing **Members** and **Primary** to the appropriate Members.
  - Modify any of the new records to suit your new Project. For example New ALMRole ApprovedActions may be different, or new ALMRole Members or Groups may be different.
  - Create a new PhaseLabel (for example, named PhaseLabel_*Name*). Locate the new Phase and change the PhaseLabel to PhaseLabel_*Name*. Also, change SecurityPolicy to SecurityPolicy_*CategoryName*, and then save the new Phase.
  - Iteration records are not copied from the old project to the new project and have to be created manually as needed.

Using the Project Wizard or the **Copy Project** button to copy a project provides a streamlined method for creating new projects quickly and efficiently, while ensuring consistency in settings across projects.

Copying a project

To copy a project from an existing project record follow these steps:

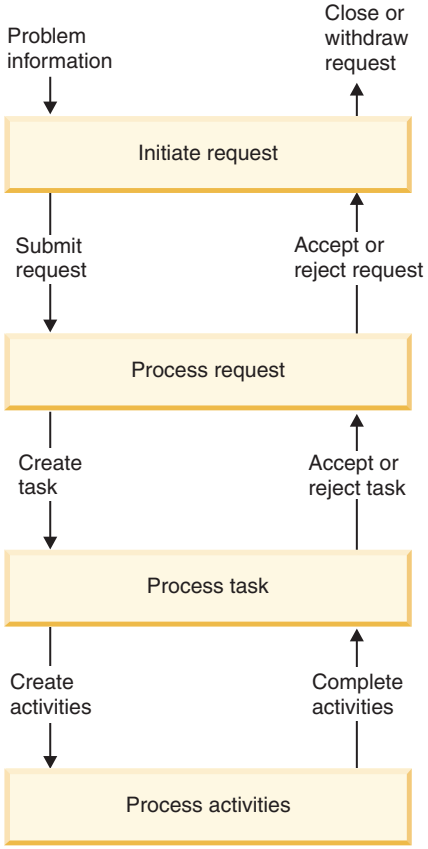
1. Open the project to be copied for modify.
2. Select the **Related Projects** tab.
3. Specify the project name, category and release.
4. Click the **Copy Project** button and wait for it to finish. Once the copy is successfully completed, you are informed of the completion and presented with the project ID of the new project.
5. Click the **Revert** button on the existing project.
6. Perform post-copy steps as outlined in “Copying an existing project” on page 36above for the copied project.

You’ll be informed if the project copy can not proceed or fails for any reasons. Investigate and correct the condition and then retry the copy again. If the project copy fails and the project is partially copied, you’ll need to manually create the remaining records and validate the fields in the records already copied. Alternatively you can mark obsolete the copies and retry the copy after correcting the conditions that resulted in the failure.

Managing work

The work records contained in the ALMWork package provide a way of keeping track of the requests and tasks that must be addressed.

Work completes a project, and Activities ensure that work is completed. The process of managing work can be implemented by using the Request, Task, and Activity record types with parent/child relationships between these records.



- A Request describes a need and identifies where the issue was found.
- A Request is reviewed to determine if and when it will be addressed.
- A Task is created for the Request.
- Activities are defined that collectively complete the Task.
- Each Activity status is reviewed to determine the completion of the Task.
- The completion of the Task indicates a resolution of the Request.
- The Request is accepted or rejected, depending on whether the Task is completed satisfactorily, or not.

Overview of ALM work process

In the ALM work process, requests are resolved by completed tasks and activities.

The Application Lifecycle Management (ALM) schema and packages provide an out-of-the-box process for using Rational ClearQuest to track your team's work on a product release. ALM uses defined roles, record types, and state transition models to help you manage the software development process from requirements submission through development, build management, and testing.

The basic ALM process is as follows:

1. A stakeholder submits a request against a software project. The stakeholder could be a developer, tester, writer, trainer, product manager, customer support representative, or other member of the project team or user of the

product. A request can initiate a change to the software project. A request can be a defect, a request for enhancement (RFE), or a task.

2. The triage team reviews the request and decides whether to accept it or reject it. If they accept the request, the triage administrator creates one or more tasks (one per project) that describe at a high level the work required to satisfy the request.
3. The lead developer for each project reviews the task and assesses the work required to implement it. The lead developer then activates the task and creates three activities associated with the task:
   - Dev activity
   - Test activity
   - Doc Assess activity

   The lead developer assigns the Dev activity to a developer.
4. The lead tester reviews the task and the Test activity, then assigns the Test activity to a tester. The documentation leader reviews the task and the Doc Assess activity, then assigns the Doc Assess activity to a writer.
5. The developer works on the Dev activity and makes the necessary changes to files. The developer then moves the Dev activity to the Completed state.
6. The release engineer creates a new baseline record that selects the newly completed activity and its associated change set.
7. The release engineer builds the project by using the newly created baseline. The release engineer creates a build record that identifies the baseline used and indicates whether the build succeeded or failed.
8. The tester installs and tests the build. When the build successfully passes all tests, the tester moves the Test activity to the Completed state.
9. The writer assesses the impact of the task on the documentation and makes all necessary changes. The writer then moves the Doc Assess activity to the Completed state.
10. The lead tester reviews the task, sees that the necessary activities have been completed, and moves the task to the Completed state. Alternatively, the lead tester creates additional activities, or comments on existing activities, if more work needs to be done.
11. The stakeholder who submitted the request reviews it and sees that one or more associated tasks have been completed. The stakeholder can open the task and review the resolution. From within the task record form, the stakeholder can open the associated activities and review the details of the development, documentation, and testing work done to complete the task. If everything looks satisfactory, the stakeholder accepts the request, which moves it to the Completed state. Otherwise, the stakeholder can reject the request and comment on the task, which notifies the lead tester by E-mail, with instructions for additional work.

Working with roles and process

Role definitions can be used to determine what actions each user is allowed to perform on a project.

These topics describe the ALM process and the roles involved with working on ALM requests, tasks, and activities.

Submitting requests

To initiate a change in your team's product, submit a request.

1. From within the ClearQuest client, click **File → New → Request**.
2. At a minimum, enter values in the following mandatory fields on the **Request** tab:
   - Headline
   - Priority
   - Severity
   - Description
   - Project (If a current project is associated with a category, you can select that category, which causes the Project field to be filled in with the current project.)

   By default the **Type** field is set to **Defect**. If the request is not a defect, select **RFE**, meaning Request for Enhancement, or **Task**.
3. If you did not select a category that is associated with a current project, on the **Project** tab click **Add** next to the **Project Found in** field. Either enter a keyword (the project ID) and click **Search** or click **Browse** and navigate to a public query, such as **FindALMProject**, to run to return a list of projects. Select the project associated with this request and click **OK**. If the request is a defect, be sure to complete the **Phase Found in**, **Iteration Found in**, and **Build Found in** fields if you know that information.
4. Click **OK**.

Triaging requests

In your role as triage administrator or project manager, periodically query for newly submitted requests and determine whether to accept or reject them. For example:

1. In the ClearQuest client, navigate to the **Public Queries → ALM → ToDo** folder and double-click the **Triage** query to run it.
2. Select a request from the Query results grid and review the request details.
3. If you decide that your team should work on the request, click the **Utility** icon on the request record form and select **CreateTask** to create a task for the work to be done to complete the request. Alternatively, you can create a task at a later time by clicking **File → New → ALMTask**. From within the task record form you can specify the request associated with the task. If you decide that your team should not work on the request, click the **Utility** icon and select **RejectUnreproducible** or **RejectWorksAsDesigned**.

Marking a request as a duplicate

In reviewing requests, you might know of another request that identifies the same defect or request for enhancement. You can mark the request as a duplicate and specify the ID of the similar request. The submitter of the request can then review it and determine whether to withdraw it.

1. In the request record form click the **Utilities** toolbar icon and select **MarkAsDuplicate**.
2. On the **Comments** tab double-click the entry for the new comment.
3. In the comment record form, modify the placeholder comment with one that describes why you think this request is a duplicate. Click **Add** next to the **Duplicate of Record ID** field. Select the request that you believe addresses the

same issue as the new one. Click the **Response Requested** checkbox if you want the owner of the request to respond to you. Click **OK**.

4. Click **Apply**.

Marking a request as duplicate completed

After a user marks a request that you own as a duplicate, review the request and decide what action to take.

1. On the request record form's **Comments** tab, double-click the entry to see the suspected duplicate request. Review the details in the request record form. If the user requested a response, enter a response. If you need a response from the user to your comment, click the **Response Requested** checkbox.
2. If you determine that your request is a duplicate, click the **Utilities** toolbar icon and select **DuplicateComplete**, which moves the request to a Completed state.

Submitting tasks

After the triage team has accepted a request, start the process of completing that request by creating a task that identifies the work required.

1. From the ClearQuest client Navigator view, navigate to the **Public Queries** → **ALM** → **ToDo** folder and double-click the **Triage** query to run it.
2. Select a request from the Query Results grid.
3. On the record form click the **Utility** toolbar icon and select **CreateTask**. On the **Task** tab enter or change values in at least the mandatory fields:
   - Headline
   - Owner
   - Priority
   - Description
4. On the **Project** tab click **Add** next to the **Project** field. Either enter a keyword and click **Search** or click **Browse** and navigate to a query, such as **FindALMProjects**, to run to return a list of projects. Select the project to be associated with this task and click **OK**. Optionally, select the phase in which the task is to be completed, and select the iteration within that phase.
5. If you want to identify any related requests, on the **Related Records** tab click **Add** next to the **Request** field. Either enter a keyword and click **Search** or click **Browse** and navigate to a query to run to return a list of requests. Select the request to be associated with this task and click **OK**.
6. Click **OK**.

Activating tasks

In your role as Lead Developer, periodically query for newly submitted tasks and then decide whether to activate them.

1. In the ClearQuest client Navigator view, navigate to the an appropriate query (for example the **Dev Lead** query in the **Public Queries** → **ALM** → **ToDo** folder) and double-click the query to run it.
2. Select a task from the Query Results grid and review the details.
3. If you decide not to work on the task, click the **Utilities** icon on the task record form and select **Reject Task**, which moves the task to the Completed state. Click **Apply**.
4. To activate the task, click the **Change State** icon on the task record form and select **Activate Task**. The state is changed to Activated. Click **Apply**.

5. To create activities for the task, click the **Utilities** icon and select **Create Activity**. The ClearQuest client creates three activities with the same headline as the task. The three activities are of the following types, which represent the different teams involved in completing the task:
   - Dev
   - Doc Assess
   - Test
6. If the default Dev activity owner is not correct, double-click the Dev type activity in the **Activities** table to open that record form. Click the **Modify** icon and assign the activity to a developer by selecting that developer's name in the **Owner** field.

   Alternatively, you can create an activity at a later time by clicking **File** → **New** → **ALMActivity**. From within the activity record form you can specify the task associated with the activity.
7. Click **Apply**.

Assigning documentation and testing activities

In your role as Documentation Lead or Test Lead, periodically query for activities that the Development Lead has created and then assign those activities.

1. In the ClearQuest client Navigator view, navigate to an appropriate query such as the **Doc Lead** or the **Test Lead** query in the **Public Queries** → **ALM** → **ToDo** folder and double-click the query to run it.
2. Select a task from the Query Results grid and review the details.
3. In the **Activities** table, double-click the Doc Assess or Test type activity to open that record form. Click the **Modify** icon and assign the activity to a writer or tester by selecting that person's name in the **Owner** field.
4. Click **Apply**.

Submitting activities

While working on a task, you occasionally need to create activities to identify work that needs to be done to complete the task.

1. From the task record form, click the **Utility** toolbar icon and select **CreateActivity**.
2. On the **Activity** tab enter values in at least the mandatory fields:
   - Headline
   - Owner
   - Priority
   - Description
3. On the **Related Records** tab you can associate other activities by clicking **Add** next to the **RelatedActivities** field. Either enter a keyword and click **Search** or click **Browse** and navigate to a query to run to return a list of tasks. Select the task to be associated with this activity and click **OK**.

   Doc Assess type activities assess the need for documentation work. Activating and completing a Doc Assess activity indicates that the assessment has been completed. For example, you might use a Doc Assess activity to instruct a writer to create an entry in the Release Notes. Click **New** next to **RelatedActivities** to create an activity for documentation work; set the activity type to Dev; and make sure that the activity task is set to the same task as the Doc Assess activity. You can use a query to retrieve the activities for a Doc Assessor (that is commonly a technical writer).

4. Click **OK**.

Querying for work assigned to you

In your role as developer, tester, or writer, periodically query the user database to see which activities have been assigned to you.

1. In the ClearQuest client Navigator view, navigate to an appropriate query (for example, in the **Public Queries** → **ALM** → **ToDo** folder) and double-click a query to run it. The ToDo folder contains these queries:
   - **Dev**
   - **Tester**
   - **DocAssessor**
   - **DocDev**

   These queries return tasks that contain activities assigned to you.
2. Select a task from the Query Results grid. In the **Activities** field double-click the Dev, Doc Assess, or Test activity to open that record form and read details about the activity.
3. When you are ready to work on the activity, click the **Change State** icon and select **Activate**.
4. Click **Apply**.

Commenting on a request, task, or activity

When working on a request, task, or activity, you can enter comments or questions and ask the owner of the record to respond.

1. In the record form click the **Utilities** toolbar icon and select **QuestionOrComment**.
2. On the **Comments** tab double-click the entry for the new comment.
3. In the comment record form, double-click the entry in the **Question/Comment/Response** field, then click **Modify**. Enter your comment. Click the **Response Requested** checkbox if you want the owner to respond to you. Click **OK**.
4. Click **OK**.

Responding to comments

Users can add comments or questions to your requests, tasks, and activities and request that you respond to them. Periodically, you should run a query that finds your records that users have commented on.

1. In the ClearQuest client Navigator view, navigate to an appropriate query (for example, **Public Queries** → **ALM** → **ToDo** folder, **ToDo_Unanswered Questions** query) and double-click the query to run it.
2. Select a record from the Query Results grid. Click the **Comments** tab. Click the entry in the **Comments** table to open the comment.
3. Click the **Utilities** toolbar icon and select **Respond**. Enter a response in the **Question/Comment/Response** field. Your response can be a question.

   The **Public Queries** → **ALM** → **General** → **AnsweredQuestions** query provides a way for users to see responses to their questions.
4. Click **OK**.

Completing a development or documentation activity

After you finish work on your development or documentation activity, change its state to Complete.

1. In the ClearQuest client Navigator view, navigate to the appropriate query (for example, **Public Queries** → **ALM** → **ToDo** folder) and double-click the query (for example **Doc Assessor** or **Tester**) to run it. The Doc Assessor query returns tasks whose Dev type activities are completed but whose Doc Assess type activities are not completed. The Tester query returns tasks whose Dev type activities are completed but whose Test type activities are not completed.
2. Open the task record that contains the activity; and open the Doc Assess or Tester type activity record by double-clicking it in the **Activities** table.
3. In the **Resolution** tab of the activity record form enter a description of the work you did to address the activity in the **Resolution Summary** field. Select an entry in the **Resolution Code** field that characterizes your action.
4. Click **Apply**.

Tracking work and builds

These topics describe how to create baseline and build records, and how to track and complete activities, tasks, and requests.

A Project record type can be used to track the production of an actual product release. A Project might include a product name, release information, and the set of all iterations associated with the Project. It could also include component information for a Project.

Every time the source code is modified the application is built and verified to be of sufficient quality to begin testing. Once verified the build is deployed to the test servers for testing. This pattern of delivering source code changes, building the application, and testing occurs regardless of the scope or magnitude of a release (for example, whether it's a major revision of an existing application, a patch or a hotfix). When errors are found, defects are logged and source code is modified to fix the defect. Again, the application must be built and deployed back to the test servers for testing.

The ALM **Baseline** and **Build** records can be used with projects using IBM® Rational ClearCase/ClearQuest Unified Change Management (UCM) integrations, as well as with those that do not. Baseline and Build records can also be used to ensure successful delivery of Projects or Product releases, or of information for customers. For example, Baseline and Build records can enable the automated transfer of information from Development to Testing, informing Testers about what product builds contain the fixes to specific Requests.

The ALM schema supports a workflow model where iterative, parallel development and testing occurs. Changes are built and then tested:

- Testers work on and complete test Activities
- Developers deliver changes after they work on and complete development Activities
- Builders create Baselines, run Builds, and validate and promote Builds.

In this pattern all of the activities are related. As developers implement functionality and fix defects, the release engineers need to know what source to include in the build or when to conduct the build (that is, when all expected work

is complete). When problems emerge with the build the release engineers need to identify the cause of the problem whether it's in the build script itself, or by some error introduced by the development team. At the same time, the testers need to know when there is a suitable build to test, what functionality is included in the build and what tests to run against that build. Every time a defect is reported, knowledge about the test cases used that uncovered the defect is needed along with references back to the original requirement. And when the developer fixes the defect and delivers the code, the cycle begins again.

In software development projects builds are occurring constantly. The common questions for the development teams are about what is implemented in the Build and what is tested in the Build. The **Build** record allows teams to capture information about each build, including its status. The **Baseline** record lets you track what activities are delivered in each build, and can be used to capture the state of Activities at a given time. The use of the Baseline, Build and Activity records enables testers to know what to test and to track which tests have been run against the build

Iterations

Tasks and activities can be assigned to iterations of the project.

Assigning tasks and activities to iterations of a project allows a project team to balance their workload across iterations. You can also create charts to see how the work is distributed across the team. The charts can help the project managers spread the work load evenly across team members and avoid high risk situations. Charts can also help project managers ensure that all work is assigned.

You can create charts to help manage workloads. For example, if there were five activities that are not assigned, there could be a trigger for the project manager that sends a notification that some work may be unaddressed or unassigned. A chart could help illustrate that wok activities are evenly balanced among the team members. By running charts project managers can govern their project more effectively, ensure that all work is assigned and everyone on a team is active, and prevent overwhelming single users with too much work.

Building and maintaining baselines

These topics describe how to create baselines that select completed activities and how to create build records.

Once a baseline is available, a build is conducted. From the Build record, you can create a reference to the Baseline record. This links the baseline and the build result. For builds that have passed, a test team can review the baseline record and set of completed activities. Each activity includes a link back to the Primary task which may also include testing activities. With this new visibility into the build contents, Testers can focus their test efforts.

A baseline should be marked Obsolete if it was rejected and did not build, or if it is no longer needed.

The Baseline record can be used to map to UCM baselines. When a baseline is created in UCM, comparing differences between the set of activities from the previous baseline identifies which activities are new. This list of activities can be

populated in the Baseline record. For non-UCM users, ClearQuest queries can be used to identify the list of activities, and they can be manually added to the baseline record.

Using an ALM work process with the UCM model provides support for component-based development, and enhances the flow of information concerning what fixes are available, what builds to download, and where the updates are available. Additional enhancements could enable users in any location to detect when a Fix in a Build becomes available, regardless of where it was fixed. To make this work, build information could be associated to a Product Release or Project, so that users know when a Request has been fixed in a later release of the product (that is, later than the version in which the problem was found). Moreover, given a Project name or ID, users could see if an updated version is available and, if so, where it can be installed from.

Creating baselines

In your role as release engineer, you create baselines frequently to capture the latest completed activities. You can then create and test builds based on those baselines. For example:

1. In the ClearQuest client, click **File → New → ALMBaseline**.
2. Enter a name for the baseline in the **Baseline** field. Enter the location of the source that will be used to perform a build. Click **Add** next to the **Project** field. Either enter a keyword and click **Search** or click **Browse** and navigate to a query to run to return a list of projects. Select the project associated with this baseline and click **OK**.
3. Click the **Activities** tab. Click **Add**. Either enter a keyword and click **Search** or click **Browse** and navigate to a query to run to return a list of activities. (For example, the **Public Queries → ALM → ToDo Manual RE Baseline Create** query retrieves all activities that have been completed since the last baseline was created.) Select one or more activities to include in the baseline and click **OK**.
4. Click **OK**.

Related concepts

"create_baseline_record.pl" on page 49

Creating a build record

In your role as release engineer, you create build records to record the status of builds based on specific baselines.

1. In the ClearQuest client, click **File → New → Build**.
2. Enter a name for the Build record in the **Build** field. Click **Add** next to the **Project** field. Either enter a keyword and click **Search** or click **Browse** and navigate to a query to run to return a list of projects. Select the project associated with this build and click **OK**. Click **Add** next to the **Baseline(s)** field. Either enter a keyword and click **Search** or click **Browse** and navigate to an appropriate query in the **Public Queries → ALM** folder (such as **Manual RE Build Create**) to run to return a list of baselines. Select the baseline included in this build and click **OK**.
3. Click the **Results** tab. Select an entry in the **Build Status** field to indicate whether the build was successful.
4. Click **OK**.

ALM with UCM

The ALMActivity record includes a Unified Change Management tab. This is an optional setting for teams that are using UCM.

Unified Change Management (UCM) is a feature of Rational ClearCase that:

- Provides a flexible, out-of-the-box source code management (SCM) model for managing change across activities and associated assets
- Provides a level of abstraction that helps with code assets
- Eliminates the need to develop and maintain scripts in a ClearCase environment
- Automates project and developer workspace setup.
- Integrates with other Rational tools providing a suite of development tools to enhance the development process

When the ClearCase/ClearQuest UCM integration is used with ClearQuest ALM, as developers check out and check in files, the ALMActivity records track the work done. The ALMActivity record is related to an ALMTask that points to a particular ALMProject. The ALMActivity record is similar to the UCM activity and all activity types are UCM enabled.

An ALMActivity maps to a UCM activity, an ALMBaseline maps to a UCM baseline, and a BTBuild maps to an actual build. When activities are completed, you create an ALMBaseline, which also creates a UCM baseline. If you create a build using the latest UCM baseline, a corresponding BTBuild record is created. The BTBuild record contains a reference to the ALMBaseline from which the build was created.

For Projects using UCM Integration, set **Project → UCMProject**. When the UCM integration is turned on for a UCM project, all UCM activities are tracked by ALMActivity records. When a UCM activity is delivered to the UCM project integration stream, the corresponding ALMActivity is completed.

By using the ALMBaseline record type to map UCM baselines, when a baseline is created in UCM, you can find the new activities in the baseline. This list of UCM activities can be populated in the ALMBaseline record. If you are not using UCM, you can use queries to identify the list of activities, and then manually add the activities to the baseline record.

The ALMBaseline record lists the ALMActivities delivered to the integration stream since the last baseline was created. The release engineer then creates a build using the last baseline. A corresponding BTBuild record is also created. The BTBuild record lists the baseline used to create this build. The record also lists the ALMActivities included since the last build.

Note: In UCM a stream is similar to a branch in other asset or software configuration management systems. The baseline records are threaded on the stream for sequencing.

When delivering an activity to any stream with a project policy for transitioning to Complete after delivering, the activity is transitioned to the Complete state, even if a developer still needs to continue working on the activity. This state transition prevents additional checkouts. The developer can:

- Do an alternate deliver to only share changes with another developer and to continue making changes using the same activity.

- Deliver to a feature stream for sharing the changes with the team working on the same features.

For example, a Developer using UCM logs in and finds UCM activities, specifies a default activity, or adds a file to a development stream and adds it to source control. The Developer can also view the activity in ClearQuest as an ALMActivity (with State = Activated).

- The Developer can use a ClearCase client to deliver changes and then complete the delivery. After the Activity is completed, the release engineer (or Builder) creates a baseline of the code.
- The Builder, logs in to ClearQuest and creates a new ALMBaseline. The Builder specifies project VOB, Project, Release values, and Activity IDs for the new baseline.
- Given a UCM baseline, one or more builds may be created from it. For each build, the Builder creates a BTBuild record.
- The Tester completes Test type Activities. The Test Activity includes a reference to the BTBuild containing the Developer fix (if a build was created). The Tester installs the build, and completes the Test Activity.

Creating baselines and ALMBaseline records

Given a UCM project, create an initial baseline, and then create an initial ALMBaseline record to anchor the UCM stream and PVOB to a series of ALMBaseline records.

For milestones or nightly builds you create a UCM baseline, and then create a new ALMBaseline record. The creation of the ALMBaseline record checks for the last baseline record created. If this is the second baseline on the given PVOB and stream, the first record would be the initial baseline. Given the current baseline, and the last baseline found, a ClearCase UCM comparison (**diffbl** operation) is used to compare the two baselines. All ALMActivities delivered since the last baseline record are added to the new baseline record.

Existing UCM projects

Older projects that were not created as ALM projects may have many existing activities and baselines. You may want to import some or all of these. Of the baselines you import you must import them in order from their stream. Not every baseline needs to be imported, but they do need to be imported in the order they were created. The create_baseline_record.pl script when given a baseline finds the new activities in the baseline, by comparing the baseline with the previous baseline record in ALM on the same stream.

If you want only new activities tracked, you can create a new ALMBaseline record on the same stream and this new baseline, rather than the original baseline from the older project will be the previous baseline for a new comparison. Only activities created since this new initial baseline will show on the new passed in baseline record. You can create this initial baseline record in either of these ways:

- Manually create it in ALM, by filling out the PVOB or Location and Stream fields so the create_baseline_record.pl script can find it.
- Use create_baseline_record.pl to create the initial baseline. The create_baseline_record.pl can create an initial seed baseline record by passing in the appropriate options. The `-nodiffbl` option specifies to create the passed in baseline and not try to find and run a comparison with a previous baseline.

Because this option does not examine the baseline, you must also include information for the `-ucmstream` *`stream_name`* argument. For example:

```
ratlperl create_baseline_record.pl -user RE -pw secret -dbname ALM -dbset CQ.ALM.HOST -projecti
```

This command creates an ALMBaseline record with the following values:

```
Project id: ALM00000123
Name: proj_01_02_24_2008
ucm_stream: proj_01_int
PVOB or Loc:\pvob01
```

After you create an initial ALMBaseline record, newer baselines records can be created in their order of creation on the stream by calling create_baseline_record.pl with the required options and the new baseline name. A baseline is compared with the previous baseline record found, and the new activities are added to the new baseline record.

"Activities" on page 7

"Baselines" on page 9

"Creating baselines" on page 46

"Builds" on page 9

"Using ALM without UCM" on page 53

"create_baseline_record.pl"

create_baseline_record.pl

The create_baseline_record.pl script is an example of how to use the ClearQuest API to create and populate an ALMBaseline record.

The create_baseline_record Perl script creates an ALMBaseline record. This operation connects the new ALMBaseline record back to the UCM baseline. It populates the ALMBaseline record with the new ALMActivities it finds. The ALMBaseline record includes references to the ALMActivity records found in the UCM baseline. The script runs a comparison with the previously stored ALMBaseline record. The script searches the UCM activities to find the activities delivered or rebased to a stream, and creates references to ALMActivities on the ALMBaseline record. You do not need to create an ALMBaseline record for every UCM baseline.

- Activities are listed in a BTBuild record

  If you create a BTBuild record that is based on the ALMBaseline record, The create_build_record script creates a reference from the BTBuild record to the ALMBaseline record that was used to create the BTBuild. This association ensures that ALMActivities are listed in a BTBuild record. The BTBuild represents the build that the UCM and ALMActivities can be found in.
- BTBuild references appear on an ALMActivity record

  The Fixed_In_Baseline field on the ALMActivity record displays which ALMBaselines and BTBuilds the ALMActivity is part of. For each found ALMBaseline record there may be several BTBuild records found.
- The ALMActivity record references the ALMTask record.

  The **Activities** field of the ALMTask record displays which **Fixed in Baselines** and BTBuilds the ALMActivity participates in.

You can build, test, and release at the component level by creating baselines. A product offering might then include all the component baselines in the offering. In this example, the product is composed of all the components. To simulate this in non-UCM you can use these command line options: -add_composed_of_baseline

<*baseline*> and -add_pvob_or_location <*location*>. You must use both of these options for each composite baseline. Given a baseline and pvob name, a corresponding ALM record reference is added to the new baseline record in the **Composed of Baselines** field.

The create_baseline_record.pl script can be run from the command line and has a `-help` option for information on all of the command line options.

Setting up a baseline record in a new project

A new project can be any project with no prior UCM work done in it, or a new project that is created to work with ALM. Imported baselines in UCM are labels imported from ClearCase and include initial component baselines. For imported baselines to determine the stream, you can pass the stream into the create_baseline_record.pl script with the `-ucmstream` option, or you can create a new baseline in the project. You need at least one completed activity to create a baseline. If you pass this new baseline into the create_baseline_record.pl script, the script can determine the stream since it was created in the project and not imported. If, at a later time, you import a label, you must make a baseline in the project afterwards and use this new baseline with the create_baseline_record.pl script.

Note: If you force a baseline with the mkbl -identical option, it causes dependencies on all components in the project which may prevent you from reconfiguring the project in the future.

Command line examples

For Windows (cmd.exe):

```
set ALM="C:\path\to\almscripts"
ratlperl "%ALM%\create_baseline_record.pl" ^
-user ReleaseEngineer -pw "" -dbname ALM -dbset CQMS.ALM.HOST ^
-projectid ALM00000002 ^
-baseline BASELINE01 ^
-pvob project_vob01 ^
-logfile %ALM%\logs\BASELINE01.log
```

For the UNIX system and Linux (/bin/sh):

```
ALM="/path/to/almscripts"
cqperl "${ALM}/create_baseline_record.pl" \
-user ReleaseEngineer -pw "" -dbname ALM -dbset CQMS.ALM.HOST \
-projectid ALM00000002 \
-baseline BASELINE01 \
-pvob project_vob01 \
-logfile "${ALM}/logs/BASELINE01.log"
```

Composite baselines:

Composite baselines are UCM baselines that are composed of other UCM baselines.

A UCM composite baseline groups member baselines under a single baseline. To create a UCM composite baseline, UCM baseline dependencies are first set, and the top level UCM component that will contain the composite baseline is set to depend on member baselines. For example, one UCM component can be set to depend on baselines from other UCM components. When a baseline is made from this component, it is composed of baselines from the components it depends on.

The create_baseline_record.pl script detects composite baselines. If a UCM baseline includes member baselines it is determined to be a UCM composite baseline. The member baselines of the UCM composite baseline are placed into the Composed of Baselines field on the ALMBaseline record.

Only composite baselines of ordinary baselines are currently supported. Composite of composite baselines are not currently supported.

Any composite ALMBaseline record you want to include into another composite must first be decomposed into ordinary ALMBaseline records, and then a composite of these ordinary ALMBaselines be made. Ordinary ALMBaseline records can participate in many composite ALMBaseline records.

ALMActivities from UCM member baselines (that is, Composed of Baselines) appear on the ALMTask records. A BTBuild record is created from the ALMBaseline record. Each BTBuild record has a unique ID. The display of ALMActivities on the ALMTask record includes a header of Composite.BTBuild.Build_System_ID. Entries in this column display the name of the BTBuild record created off the ALM composite baseline.

Builds off the ordinary ALMBaseline records can run during the builds off the composite ALMBaseline records. Ordinary builds leave the Composite.BTBuild column empty.

UCM composite baselines can be used for Release oriented or Component oriented projects. ALM composite baseline records can be used in either type of orientation, but with the restriction that composites of composites are not currently supported.

For a release oriented project, ordinary baselines used on a main UCM integration stream will pickup all activities delivered to it from any substream. Therefore on the activity you would see the builds created off the substream and later, the builds created off the top integration stream.

Activities are never lost. They report to any baseline they appear in as long as all projects use the UCM integration. This includes:

- Activities delivered through interproject rebase operations
- Activities delivered through deliver-baseline operations
- Activities delivered through alternate deliveries methods

"create_baseline_record.pl" on page 49

"ALM with UCM" on page 47

update_baseline_record.pl:

The update_baseline_record.pl script is an example of how to use the ClearQuest API to update an ALMBaseline record.

The update_baseline_record.pl script updates an existing ALMBaseline record. The script requires a UCM baseline with a set promotion level, and PVOB name to be passed to it. The script reads the status of the UCM baseline and updates the corresponding ALMBaseline record field values for Promotion Level and Obsolete.

If the UCM promotion level is Rejected or the Obsolete field is set, the ALMBaseline record is not used as a candidate for the comparison run by the create_baseline_record script.

The ClearCase UCM baseline promotion levels must match with values in ALM. (For example, the Rejected promotion level must be used in ClearCase to mark a bad baseline.) ALM Baseline scripts depend on the promotion value to be defined and use it for creating (or refusing to create a baseline if the promotion level is Rejected) ALMBaseline records and correctly populating the activities list by doing a comparison with the previous baseline.

The script can be used from a command line build automation process and should not be modified. The script has a `-help` option for information on all of the command line options.

create_build_record.pl

The create_build_record.pl script is an example of how to use the ClearQuest API to create and populate a BTBuild record.

The create_baseline_record Perl script creates a BTBuild record. The script requires an existing ALMBaseline record name, a PVOB name, and the name for the new BTBuild record, as well as values for the ALMBuildStatus, ALMBuildType, and ALMProject fields. One or more BTBuild records can be created from and associated to an ALMBaseline record. Creating a BTBuild record off an ALMBaseline record sets up the associations to allow an ALMActivity list on a ALMTask record to display which BTBuilds the ALMActivity can be found in.

- For a given ALMBaseline record one or more BTBuild records can be created from it.
- Each BTBuild is associated to each of the ALMActivity records that are listed on the ALMBaseline record.
- Each ALMTask record that lists the ALMActivity record lists each BTBuild record the ALMActivity participates in.

The -url option to populates the **Build Web URL** field on **Build Details** tab of the BTBuild record that the script creates.

The script should not be modified, and can be run from the command line. It has a `-help` option for information on all of the command line options.

Command line examples

For Windows (cmd.exe):

```
set ALM="C:\path\to\almscripts"
ratlperl "%ALM%\create_build_record.pl" ^
-user ReleaseEngineer -pw "" -dbname ALM -dbset CQMS.ALM.HOST ^
-projectid ALM00000002 ^
-pvob project_vob01 ^
-baseline BASELINE01 ^
-build Build_BASELINE01 ^
-buildstatus "Passed" ^
-buildtype "Platform"
```

For the UNIX system and Linux (/bin/sh):

```
ALM="/path/to/almscripts"
cqperl "${ALM}/create_build_record.pl" \
-user ReleaseEngineer -pw "" -dbname ALM -dbset CQMS.ALM.HOST \
-projectid ALM00000002 \
-pvob project_vob01 \
-baseline BASELINE01 \
-build Build_BASELINE01 \
-buildstatus "Passed" \
-buildtype "Platform"
```

Related tasks

"Creating ALMType labels" on page 33

update_build_record.pl:

The update_build_record.pl script is an example of how to use the ClearQuest API to update a BTBuild record.

The update_build_record.pl script updates an existing BTBuild record. The script requires the ID of an existing BTBuild record be passed to it, and updates the BTBuild record with data supplied as arguments, such as the ALMBuildStatus field.

The script can be used from a command line build automation process and should not be modified. The script has a `-help` option for information on all of the command line options, including a complete list of the fields that can be updated with the script.

Using ALM without UCM

You can use the ALMBaseline and BTBuild record types without using UCM.

By using ClearCase UCM the ALMBaseline and BTBuild records can automatically detect activities that are included in builds. However, you can also use the ALMBaseline and BTBuild record types to manage change and activities with systems that are not using UCM. The term, *non-UCM* refers to any system using a configuration or asset management solution other than UCM.

When you create a ALMBaseline record, you can use queries to identify the list of activities, and then manually add the activities to the ALMBaseline record.

Note: When adding an ALMActivity to an ALMBaseline record the ALMActivity ID must be valid or ALMBaseline is not updated with the activity added.

Creating baselines and builds

The ALMBaseline record is used to hold data on a baseline. In non-UCM this can be a Label placed on a repository. This label must be static for the life of the project. It should not be moved or be reapplied.

The ALMBaseline record's unique key is a combination of the BaselineName and PvobOrLocation fields. In UCM the PVOB holds the process information for a UCM project. In non-UCM the PvobOrLocation is the Location, which can be a component or project area, that makes the Label unique. For example, if you have two components, **Gui** and **Core**, that are built separately, but your nightly build labels are generic (such as NightlyBuild_2008Jan15) then you could create baseline records with the BaselineName and PvobOrLocation values:

```
BaselineName=NightlyBuild_2008Jan15  Location=Gui
BaselineName=NightlyBuild_2008Jan15  Location=Core
```

Given a baseline record, one or more builds may be derived from it. For example, if you build for three platforms, then for one baseline record you would need three build records.

Example

Libraries Ltd. is a software library producer. They create .jar files and release groupings of these files as archives. The company's change management (CM) system is file based. Each .jar file can be defined as a *component*. The archive that contains a grouping of .jar files can be defined as an *offering*. The component team .jar files are stored in directories (for example, Jar\Gui_01.jar, Jar\Gui_02.jar, ...) Component level Testers will test each .jar file at the component level. Components do not necessarily know what (product) offering they may be part of. The offerings are created by the release engineer (or Builder) who created the archive files that contain the components. Offerings are stored in directories (for example, Products\Sparkle_01 and Products\Dazzle_01). Product level Testers will test the archive files, and all the .jar files within it at the product level.

The overall work process includes these steps:

- Create an ALMProject (for example, named nonUCM_GuiJar).
- Create an ALMRequest and a ALMTask for the request.
- Create an ALMActivity for the development work (for example, with Activity ID = ALM00000486).
- Complete an ALMActivity. The Developer modifies the code and closes the Activity.
- Create an ALMBaseline. Builder creates the jar file GUI_Jar_02.jar and creates a baseline record (GUI_Jar_02), adding in the Activities completed. The Builder can run a query (based on Developer Category and Release) and then click a Task in Result Set grid and view the Activities field. After the baseline is created, one or more builds can be built.
- Create a BTBuild from the baseline. The Builder creates a new BTBuild, that references the appropriate ALMProject and ALMBaseline. The Activities tab on the BTBuild record shows all the activities that the ALMBaseline includes. The ALM tab on the BTBuild shows the connection to the ALM Baseline.
- Test a build. The Tester can view an ALMTask and see what build the new functionality can be found in.

Creating a composite baseline means taking existing baselines and adding them to the **Composed of Baselines** field on a new baseline record. For example a product level baseline could include all the component level baselines.

In our example, the **Composed of Baselines** includes baseline GUI_Jar_02 from the component baseline. The Builder can then create a new BTBuild record off the new Dazzle_01 baseline. It is the same process as that used to create the build off the Gui component. The same ALMTask record reveals to the product level Tester in which build the new functionality can be found in.

"Activities" on page 7

"Baselines" on page 9

"Creating baselines" on page 46

"Builds" on page 9

"ALM with UCM" on page 47

"create_baseline_record.pl" on page 49

Testing

These topics describe the testing tasks.

Completing the test activity

In your role as release tester, after you have successfully installed and tested the build, complete the test activities associated with the tasks included in the build.

1. In the ClearQuest client, click **Edit → Find Record**. Enter the activity record ID in the Find Record window and click **OK**.

   If you do not remember the activity record ID, you can run a query (for example, in the ALM OOTB sample database, in the ClearQuest Navigator view, navigate to the **Public Queries → ALM → ToDo** folder and double-click the **Tester** query to run it). A Tester query returns tasks whose Dev type activities are completed but whose Test type activities are not completed. Open the Test type activity record by double-clicking it in the **Activities** table.
2. In the **Resolution** tab of the activity record form enter a description of the work you did to address the activity in the **Resolution Summary** field. Select an entry in the **Resolution Code** field that characterizes your action.
3. Click **Apply**.

Completing the task

In your role as a Task owner you are responsible for periodically querying your tasks for: development activities that have been completed; reviewing the status of the documentation and test activities; and moving your tasks to a Completed state when appropriate.

Using the ALM OOTB sample database as an example:

1. From the ClearQuest client Navigator view navigate to the appropriate query for completing tasks (for example, **Public Queries → ALM → ToDo** folder) and double-click the **Completing Tasks** query to run it. When prompted, select the project category and release that the query should search.

   The query returns tasks that contain at least one completed development activity.
2. Select a task in the Query Results grid.
3. Double-click each of the activities in the Activities field to open the activity record forms. Review the resolution descriptions for the development activities. Verify that the activities were completed successfully. Assess the state of the documentation activities.
4. If you are satisfied that the work associated with the task has been completed successfully, click the **Change State** icon on the task record form and select **Complete**. In the **Resolution** tab enter a description of the resolution in the **Resolution Summary** field, and select an entry in the **Resolution Code** field that characterizes your actions.
5. Click **Apply**.

Accepting or rejecting a request

Throughout the development lifecycle you should check the status of Requests you submitted. When one or more of the Tasks associated with a Request have been completed, review the work done and either accept or reject it. Accepting the Request at this point moves it to the Completed state. Rejecting the Request creates a completed Task for a project, which indicates that no more work is to be done and users cannot associate Tasks with the Request.

1. From the ClearQuest client Navigator view, use a query to find open Requests. For example, navigate to the **Public Queries → ALM → ToDo** folder and

double-click the **Requestor** query to run it. When prompted, select **Opened** for the Request state and **Completed** for the Task state. Select the project category and release that the query should search.

The query returns your open Requests for which one or more Tasks have been completed.

2. Double-click each of the Tasks in the **Tasks** field to open the Task record forms. Review the resolution descriptions. From the Task record forms you can double-click activities in the **Activities** field to open Activity record forms and review resolution details for each Activity.
3. If you are satisfied with the work done for the request, click the **Change State** icon on the request record form and select **Accept**. If you are not satisfied with the work done, select **Reject**, (or **Unreproducible** or **WorksAsDesigned**) and update the **Description** field with an explanation for your rejection. You can also open a Request directly and resolve it as Accepted, Rejected, Unreproducible, and WorksAsDesigned.

   You may also withdraw a request at any time. Doing so does not necessarily stop all work related to the Request; the Development lead might decide to continue working on tasks.

   You can run a query to retrieve completed Tasks that have been rejected by the Triage administrator or Project manager (such as the **Requests Rejecting Task Solutions** query in the **Public Queries** → **ALM** → **ToDo** folder).
4. Click **Apply**.

You can ReOpen a Task that is in the Completed state and can change the Project and reset field values Comment records may need to be updated if you reopen a Request.

Sample data

A sample database is provided to help get you started with the new schema.

You can start with the sample database to familiarize yourself with the new records and their relationships. This sample database and set of importable files is based on the OpenUP (Open Unified Process) which provides the best practices in process design in the Eclipse Process Framework project. OpenUP is similar the Rational Unified process and is scaled for Agile teams. OpenUP downloads are available from the following URL: http://www.eclipse.org/epf/downloads/openup/openup_downloads.php

The available version of OpenUp is viewable in a browser, with all content stored on the local hard drive. In this sample database a set of labels, work types, work configurations, and queries are already provided. Additionally a sample project is provided. OpenUP processes are mapped to the ALM Task and Activity records to illustrate how to bring process into managing projects using ClearQuest. There are also ToDo samples, which are a subset of OpenUp.

For information on OpenUp see http://epf.eclipse.org/wikis/openup/

OpenUp

Mandatory fields for ALM record types

The following table lists the required fields for each ALM record type, in the order in which the record types must be created for a ClearQuest ALM Project.

Table 2.

Record type	Mandatory fields
ALMAdmin	**Label**, **Members**
ALMResolutionCodeLabel	**Name**
ALMResolutionCode	**ALMRecordType**, **ResolutionCodeLabel**
ALMReleaseLabel	**Name**
ALMRoleLabel	**Name** and **ApprovedActions**
ALMCategoryTypeLabel	**Name**
ALMStatusLabel	**Name**
ALMTypeLabel	**Name**
ALMIterationLabel	**Name**
ALMPhaseLabel	**Name**
ALMType	**ALMRecordType**, **TypeIndicator**, and **TypeLabel**
ALMSecurityPolicy	**Name**, **ratl_context_groups**
ALMCategory	**Name**, **CategoryTypeLabel**, **SecurityPolicy**
ALMProject	**SecurityPolicy**, **Category**, and **Owner**. **Name** and **Release** are optional fields that can help identify project uniqueness. These fields are mandatory if a project already exists with blank or non-unique Name and Release values. You can also specify values for **SuperProject** and **PriorProject** if they exist and are relevant to your new project.
ALMPhase	**Project**, **PhaseLabel**, **SecurityPolicy**
ALMIteration	**Project**, **PhaseLabel**, **IterationLabel**, **Phase**, **PriorIteration**, **SecurityPolicy**
ALMRole	**Project**, **SecurityPolicy**, **RoleLabel**, **Members** or **Groups**, **Primary**, and **ApprovedActions**.
ALMWorkConfiguration	**Project**, **SecurityPolicy**, **ALMRecordType**, **ALMTypeLabel**, and **Roles**. The **Primary Children Configs** and **Secondary Children Configs** are optional fields. The **Primary** field is used to choose the default owner on the ALMTask and ALMActivities, and it is also used to select the mastership site of these records.
ALMRequest	**Owner**, **Headline**, **SecurityPolicy**, **Project**, **Type**, **Severity**.
ALMTask	**Request**, **Owner**, **Headline**, **SecurityPolicy**, **Project**, **OldID**,**Priority**, **Type**, **Roles**.

Table 2. (continued)

Record type	Mandatory fields
ALMActivity	**Headline**, **Type**, **SecurityPolicy**, **Task**, **Roles**.
ALMBaseline	**PVOB_OrLocation**, **Owner**, **Name**, **SecurityPolicy**, **Project**.
BTBuild	**Build_System_ID**, **ALMProject**, **ALMBuildType**, **ALMBuildStatus**, **ALMBaseline**, **ALMRoles**, **ALMSecurityPolicy**, **ALMOwner**.
ALMComment	No fields are mandatory.

Index

A

action
  SetDefault 11
ALM
  applying packages 18
ALMType 33
applying packages
  ALM 18

B

baseline
  composite 50
  create baseline script 49
BTBuild record
  create_build_record script 52

C

composite
  baseline 50
copying
  project 36, 37
create
  baseline 49
create_build_record script 52
customizing
  ALM schema 18

D

duplicates 9

I

integration
  UCM 47
integrations
  UCM 45

M

modifying
  ALM schema 18

P

package upgrades
  ALM 18
project
  copying 36, 37
projects 4, 20

S

SetDefault action 11
sub-projects 4
SubProject 20
SuperProject 20

T

TypeIndicator 33

U

UCM 45
UCM integration 47
using ALM
  non-UCM 53
  with UCM 47