

**Windows Version 2003.06.13, UNIX patch 2003.06.00–6**  
Windows and UNIX



**Documentation Supplement**



**Windows Version 2003.06.13, UNIX patch 2003.06.00–6**  
Windows and UNIX



**Documentation Supplement**

**Note**

Before using this information and the product it supports, read the information in “Notices,” on page 55.

**1st edition (September 2004)**

This edition applies to Windows Version 2003.06.13 and UNIX patch 2003.06.00–6, of IBM Rational ClearQuest and to all subsequent releases and modifications until otherwise indicated in new editions.

© Copyright International Business Machines Corporation 1997, 2004. All rights reserved.

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

---

# Contents

<b>Figures</b> . . . . .	<b>v</b>
--------------------------	----------

<b>Tables</b> . . . . .	<b>vii</b>
-------------------------	------------

<b>About this book</b> . . . . .	<b>ix</b>
----------------------------------	-----------

Who should read this book . . . . .	ix
Typographical conventions . . . . .	ix
Related information . . . . .	x
ClearQuest documentation roadmap . . . . .	x
Contacting IBM Rational Customer Support . . . . .	x

<b>Summary of changes</b> . . . . .	<b>xiii</b>
-------------------------------------	-------------

<b>Chapter 1. Introduction</b> . . . . .	<b>1</b>
--	----------

<b>Chapter 2. New code page features</b> . . . . .	<b>3</b>
--	----------

Setting the data code page in the ClearQuest Maintenance Tool. . . . .	3
What are code pages? . . . . .	3
Setting the data code page for a new schema repository . . . . .	3
Changing the ClearQuest data code page for existing schema repositories . . . . .	4
Supporting the EUC-JP code page . . . . .	4
Using installutil setdbcodepagetoecjpsafeshiftjis . . . . .	4

<b>Chapter 3. Simplifying deployment with new database drivers</b> . . . . .	<b>5</b>
--	----------

Functions of database property pages . . . . .	5
Setting database properties for Oracle . . . . .	6
Setting database properties for SQL Server . . . . .	6

<b>Chapter 4. Changes to command-line utilities</b> . . . . .	<b>9</b>
---	----------

Updating connect options for installutil . . . . .	9
Specifying connect options in installutil for Oracle . . . . .	9
Specifying connect options in installutil for SQL Server . . . . .	9
New subcommand — installutil registerconnectoptions. . . . .	10
Syntax . . . . .	10
New subcommand — installutil getconnectoptions . . . . .	10
Syntax . . . . .	10
Updating connect options for pdsq . . . . .	11
Specifying connect options in pdsq for Oracle. . . . .	11
Specifying connect options in pdsq for SQL Server . . . . .	11
Using cqreg refresh for ClearQuest clients on UNIX . . . . .	11

<b>Chapter 5. API and Hooks Updates</b> . . . . .	<b>13</b>
---	-----------

New content . . . . .	13
Error checking and validation . . . . .	13
Debugging your code . . . . .	14

Actions and access control . . . . .	15
Name lookup in Perl hooks . . . . .	17
Default entity. . . . .	19
Editing an existing record . . . . .	20
When the record is committed . . . . .	20
Performance considerations for using hooks . . . . .	20
InvalidateFieldChoiceList example. . . . .	22
Using Perl for external applications . . . . .	23
RATLC00702699, APAR IC37754; Documentation for the highlighting of keywords in the ClearQuest Designer script editor . . . . .	23
Corrections and other changes to documentation . . . . .	25
RATLC00708226, RATLC00706668, RAMBU00050315; Commit behavior documentation enhancements . . . . .	25
RATLC00712744; Syntax is incorrect for the GetFieldRequiredness method . . . . .	25
RATLC00712920, RATLC00710309, RAMBU00054358, RAMBU00056057; Date timestamp issues . . . . .	25
RATLC00696630, RATLC00696270, RATLC00696759, RATLC00710896; Generating reports and updates to SetHTMLFilename documentation . . . . .	25
RATLC00705428; SuperUser privilege required for SetUserName method. . . . .	26
RATLC00707609; Methods to set and get user privileges . . . . .	26
RATLC00666959, RATLC00698133, RAMBU00046241; Creating PERL and VBScript Hooks of the same name causes the creation of new hooks to fail . . . . .	27
RATLC00712994, RAMBU00036659; CQString is not MBCS, which is not suitable for internationalization. . . . .	27
RATLC00703293, APAR IC37932; SetLoginName method update . . . . .	28
RATLC00701671, APAR IC37619; Updates to the description of the UserLogon method database set argument . . . . .	28
RATLC00712943, RATLC00712310, APAR IC39076; Updates to the SetFrom method of the Mail message object . . . . .	28
RATLC00705405; Correction to "Running a Query and Reporting on its Result Set" code example . . . . .	28
RATLC00453581, RAMBU00050338, RAMBU00035392, RATLC00656939, RATLC00712567, RATLC00710254, RATLC00705491, RAMBU00009075, RAMBU00010073, RATLC00654966, RAMBU00050417; Actions and access control documentation enhancements . . . . .	29
RATLC00447393; Setting a field value or variable . . . . .	29
RAMBU00036184; Naming a field . . . . .	29
RATLC00705480, RAMBU00054500; New methods that enhance performance . . . . .	29

RATLC00450645, RAMBU0046105; ClearQuest hooks database location has changed . . . . .	33
RATLC00703780, RAMBU0010103, APAR IC41898; Package-installed hooks are read-only . . . . .	33
RATLC00699730; Code example for HasDuplicates correction . . . . .	33
RATLC00697318, RATLC00708183; StringIdToDbId method of the Session object . . . . .	34
RATLC00705438; Perl API Build method syntax . . . . .	34
RATLC00701064, RATLC00705313; Database object password methods require SuperUser privilege . . . . .	34
RATLC00698109, RATLC00696104; UNIX support for reports in a workspace . . . . .	35
RATLC00703013, RATLC00713905, RATLC00702914, APAR IC37813; cqole.odl and cqole.dll mismatch . . . . .	35
RATLC00719064; Perl SetActive method not working correctly with Boolean as documented . . . . .	35
SaveQueryDef code example correction . . . . .	36
RATLC00715405; Document the Session.ClearNameValues method . . . . .	36
SaveQueryDef method of the Workspace object issues . . . . .	37
RATLC00711964, RAMBU00022729; GetFieldRequiredness return value for read_only fields . . . . .	37
RATLC00715159, RATLC00059373; AddParamValue method allows the insertion of one string value . . . . .	37
RATLC00703830, RATLC00667284, RAMBU00053964; New documentation on error checking and validation . . . . .	38
RATLC00371877; UnmarkEntityAsDuplicate method of the Session object note . . . . .	38
RATLC00718478; ValidateQueryDefName method of the Workspace object . . . . .	38
RATLC00721299; GetFieldOriginalValue method should include note . . . . .	38
Upgrading user information from a schema repository to a user database . . . . .	38

RATLC00722670, APAR IC40986; RegisterSchemaRepoFromFile and GetLastSchemaRepoInfo documentation update . . . . .	39
Updates to "Ensuring that record data is current" section in <i>API Reference</i> . . . . .	40
RATLC00445073, RATLC00721111, APAR IC39464; Hook Performance issues and guidelines . . . . .	40
GetValueAsList return value description is incorrect in <i>API Reference</i> . . . . .	40
Document the Entity.Reload method . . . . .	40
RATLC00717324, RATLC00707206; New methods for hiding records types . . . . .	41
RATLC00696096; CtCmd code examples for UCM/ClearQuest integrations . . . . .	42
VBScript Code Example Errors in <i>API Reference</i> . . . . .	45
RATLC00715484; Version information for newer ClearQuest API methods . . . . .	45

## Chapter 6. MultiSite documentation updates . . . . . 49

Upgrading a schema version with ClearQuest MultiSite . . . . .	49
Upgrade instructions . . . . .	49
Synchronizing multiple user database families with msimportauto.bat . . . . .	50
Why should I use the msimportauto.bat script? . . . . .	50
Running msimportauto.bat . . . . .	51
repair . . . . .	52
Applicability . . . . .	52
Synopsis . . . . .	52
Description . . . . .	52
Restrictions . . . . .	53
Options and arguments . . . . .	53
Examples . . . . .	54
See also. . . . .	54

## Appendix. Notices . . . . . 55

---

## Figures



---

## Tables



---

## About this book

This book contains updates to the ClearQuest Administrator Guide and ClearQuest API Reference.

---

## Who should read this book

The information in this book is intended for ClearQuest administrators and users of the ClearQuest API.

---

## Typographical conventions

This manual uses the following typographical conventions:

- *ccase-home-dir* represents the directory into which the ClearCase Product Family has been installed. By default, this directory is /opt/rational/clearcase on UNIX and C:\Program Files\Rational\ClearCase on Windows.
- *cquest-home-dir* represents the directory into which Rational ClearQuest has been installed. By default, this directory is /opt/rational/clearquest on UNIX and C:\Program Files\Rational\ClearQuest on Windows.
- **Bold** is used for names the user can enter; for example, command names and branch names.
- A *sans-serif font* is used for file names, directory names, and file extensions.
- A **sans-serif bold font** is used for GUI elements; for example, menu names and names of check boxes.
- *Italic* is used for variables, document titles, glossary terms, and emphasis.
- A monospaced font is used for examples. Where user input needs to be distinguished from program output, **bold** is used for user input.
- Nonprinting characters appear as follows: <EOF>, <NL>.
- Key names and key combinations are capitalized and appear as follows: SHIFT, CTRL+G.
- [ ] Brackets enclose optional items in format and syntax descriptions.
- { } Braces enclose a list from which you must choose an item in format and syntax descriptions.
- | A vertical bar separates items in a list of choices.
- ... In a syntax description, an ellipsis indicates you can repeat the preceding item or line one or more times. Otherwise, it can indicate omitted information.

**Note:** In certain contexts, you can use “...” within a pathname as a wildcard, similar to “\*” or “?”. For more information, see the **wildcards\_ccase** reference page.

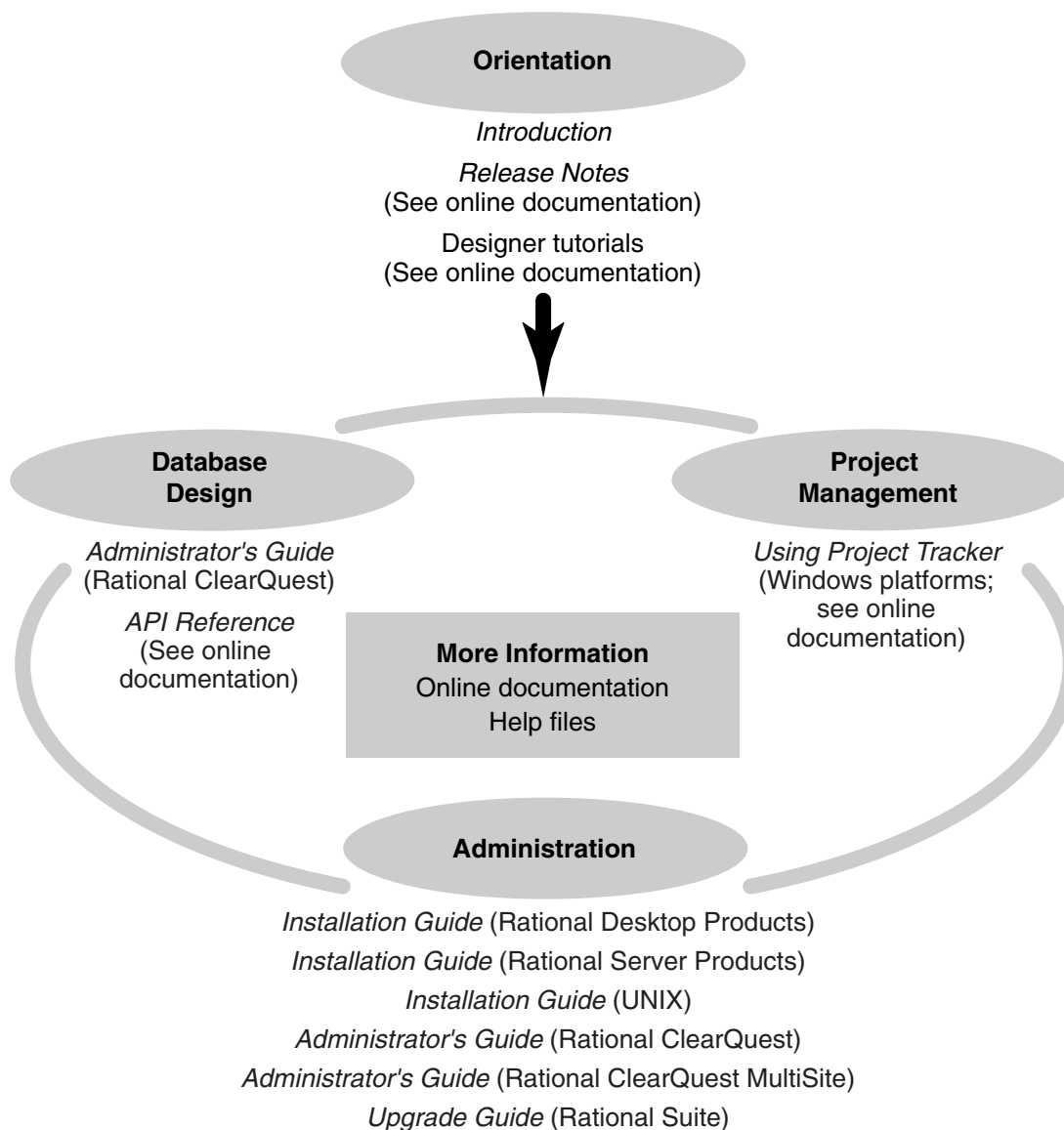
- If a command or option name has a short form, a “slash” ( / ) character indicates the shortest legal abbreviation. For example:

**1sc/heckout**

---

## Related information

### ClearQuest documentation roadmap



---

## Contacting IBM Rational Customer Support

If you have questions about installing, using, or maintaining this product, contact IBM Rational Customer Support as follows:

The IBM software support Internet site provides you with self-help resources and electronic problem submission. The IBM Rational Software Support Home page can be found at <http://www.ibm.com/software/rational/support/>.

Voice Support is available to all current contract holders by dialing a telephone number in your country (where available). For specific country phone numbers, go to <http://www.ibm.com/planetwide/>.

**Note:** When you contact IBM Rational Customer Support, please be prepared to supply the following information:

- Your name, company name, ICN number, telephone number, and e-mail address
- Your operating system, version number, and any service packs or patches you have applied
- Product name and release number
- Your PMR number (if you are following up on a previously reported problem)



---

## Summary of changes

This is a first edition.



---

## Chapter 1. Introduction

This documentation supplement explains ClearQuest and ClearQuest MultiSite features introduced in Windows service release 2003.06.13 and UNIX patch 2003.06.00–6. The information in this document supplements the information in the versions of the ClearQuest Administrator Guide, ClearQuest API Reference, and ClearQuest MultiSite documentation that were released with ClearQuest Version 2003.06.00 for Windows and UNIX.



---

## Chapter 2. New code page features

This chapter contains information on two new features in Rational ClearQuest v2003.06.13 related to data code pages. With this service release, data code pages for ClearQuest schema repositories can be set or modified using the ClearQuest Maintenance Tool instead of the installutil command line utility. In addition, this service release adds support for the EUC-JP database code page.

---

### Setting the data code page in the ClearQuest Maintenance Tool

Prior to IBM Rational ClearQuest v.2003.06.13, the primary way to set a ClearQuest data code page was by using several subcommands from the installutil command line utility. With ClearQuest v.2003.06.13, the ClearQuest Maintenance Tool allows you to set the ClearQuest data code page when you create or modify a schema repository.

#### What are code pages?

Code pages specify what character set can be used on a computer or in an application, and how those characters are stored in binary format. There can be three types of code pages in a ClearQuest environment:

The database code page is the code page setting for the vendor database. It determines which characters can be stored in the database that house the ClearQuest schema repository and user data.

The client code page, or the operating system code page for ClearQuest clients, determines which characters the client can read from the ClearQuest database, display for the user, process and write back to the ClearQuest database.

The ClearQuest data code page, created in ClearQuest v2002.05.01 Patch 2, helps ClearQuest enforce consistency between the data code page and the client code pages for a given ClearQuest schema repository and the associated user databases. In a sense, the ClearQuest data code page stands between the database code page and the client code page, to help ClearQuest check that they are consistent. For more information about code pages, see the IBM Rational ClearQuest Administrator's Guide.

#### Setting the data code page for a new schema repository

When you create a new schema repository, the ClearQuest Maintenance Tool automatically determines the code page of the operating system on which it is running, and offers you a choice to set the ClearQuest data code page to that value or to ASCII.

For example, if you are running the ClearQuest Windows client in English, the code page of the operating system is 1252 (ANSI-Latin1). The ClearQuest Maintenance Tool allows you to set the data code page to either 1252 (ANSI-Latin1) or ASCII. However, if you are running the ClearQuest Maintenance Tool on a Japanese 932 (Shift-JIS) client, the Maintenance Tool will display 932 (Shift-JIS) as the only choice because the Japanese version of the ClearQuest client already contains Japanese characters.

To set the ClearQuest data code page for new schema repositories using the ClearQuest Maintenance Tool::

1. From the **Schema Repository** menu, select **Create**.
2. Enter a name for the schema repository connection highlighted in the **Existing Connections** area and click **Next**.
3. In the Schema Repository Properties area, select a database vendor and enter the required database properties and click **Next**.
4. In the ClearQuest data code page dialog for the new connection, select either **ASCII** or the **platform code page** for the machine on which the Maintenance Tool is running and click **Next**.
5. Continue with the Create Connection wizard to either create a sample user database or click **Finish** to complete the schema repository creation process..

## Changing the ClearQuest data code page for existing schema repositories

To change the ClearQuest data code page for existing schema repositories using the ClearQuest Maintenance Tool:

1. In the **Existing Connections** area, select the connection for which you want to change the code page. Then from the **Schema Repository** menu, select **Change Code Page**.
2. For the **Logon** information fields, enter the **user name** and **password** for the schema repository and click **Next**. Only users with Super Privileges are allowed to change the code page value for the schema repository.
3. In the ClearQuest data code page dialog, select to change the setting to either **ASCII** or the **platform code page** for the machine on which the Maintenance Tool is running and click **Finish**.
4. ClearQuest displays a warning about changing code page values. Click **OK** to continue.
5. ClearQuest then confirms the change and enters the data code page value in a log. Click **Done** to close the dialog.

---

## Supporting the EUC-JP code page

With version 2003.06.13, ClearQuest can work with vendor databases that use the EUC-JP database code page.

To set a schema repository to work with the EUC-JP database, use the new `installutil setdbcodepagetoecjpsafeshiftjis` subcommand to set the ClearQuest data code page to a value of 60932, a Rational-defined value for a subset of the Microsoft Shift-JIS character set that can be written safely to EUC-JP databases.

This setting allows ClearQuest clients on machines using the Shift-JIS code page (932) to read and write to the ClearQuest databases. Clients that are not on a machine using the Shift-JIS code page (932) can only read from the databases and cannot modify them.

## Using `installutil setdbcodepagetoecjpsafeshiftjis`

To set the ClearQuest data code page to 60932 — EUC-JP, the syntax for `installutil setdbcodepagetoecjpsafeshiftjis` is as follows:

```
installutil setdbcodepagetoecjpsafeshiftjis -dbset 2003.06.13 admin_user  
admin_password
```

---

## Chapter 3. Simplifying deployment with new database drivers

With IBM Rational ClearQuest version 2003.06.13 for Windows and clearquest\_p2003.06.00–6 for UNIX, IBM Rational is providing new database drivers with ClearQuest for Oracle and SQL Server. These drivers simplify the deployment of ClearQuest clients, and also improve performance. The new drivers are based on technology from DataDirect, and replace the database drivers from OpenLink that were used in previous releases of ClearQuest. For environments using Oracle databases, you do not need to install Oracle client software on the same systems as the ClearQuest Windows clients.

To facilitate the use of the new drivers, Rational has changed the database property pages for Oracle and SQL Server. This chapter explains the fields in the database property pages used for Oracle and SQL Server. Use this section as a supplement to the ClearQuest documentation and online help from earlier releases. The workflows and procedures described in the existing documentation and online help remain the same except for the database property pages discussed here.

If you have an environment with existing pre-2003.06.13 ClearQuest databases and clients, see the IBM Rational Suites *Upgrade Guide* and IBM Rational ClearQuest Product Family *Installation Guide* for UNIX for more information on setting these database properties.

---

### Functions of database property pages

To facilitate the use of the new drivers, Rational has changed the database property pages for Oracle and SQL Server. Database property pages open when you perform functions related to either connections or schema repositories using the ClearQuest Maintenance Tool, such as:

- Creating a new connection
- Editing a connection
- Duplicating a connection
- Creating a new schema repository
- Moving a schema repository
- Upgrading a schema repository
- Updating a schema repository

Database property pages also open when you perform the following user database related functions using the ClearQuest Designer, such as:

- Upgrading a user database
- Moving a user database
- Viewing the properties of a user database
- Updating the properties of a user database

Information on these functions is available in the IBM Rational ClearQuest *Administrator's Guide*.

---

## Setting database properties for Oracle

In ClearQuest version 2003.06.13, the fields shown on the database properties page for Oracle are:

- Vendor (Oracle)
- Server
- SID
- User Name
- Password
- Connect Options

After selecting Oracle in the Vendor field, complete the remaining fields as follows:

1. In the **Server** field, enter the machine name of the server where the Oracle database is running. It may have a domain name added to it, for example, dbserv.xxx.companyname.com.
2. In the **SID** (Oracle System Identifier) field, enter the name of the database instance that will be used for the schema repository.
3. In the **User Name** field, enter the user name you created for the database.
4. In the **Password** field, enter the password for the user name.
5. In the **Connect Options** field, the default **LOB\_TYPE** (or data type) is **CLOB**.

When you are creating either a new schema repository or user database with Oracle, the Connect Options field displays **LOB\_TYPE=CLOB**. CLOB stands for Character Large Object and is the default value. An alternate selection is LONG. However, CLOB is Oracle's preferred method for storing large objects. It simplifies the way database administrators set up searching on multiline text fields.

You can also use the Connect Options field to change the port number. If the port number of the database is different from the default for Oracle, which is 1521, then the port number should be entered in this field, in the form **PORT=port\_number**

The connect options field can also be used to enter a series of arguments that will help ClearQuest clients connect with ClearQuest databases. This is particularly important when it is necessary to have ClearQuest clients installed in a prior release connect to a ClearQuest database that was created in version 2003.06.13. In the Connect Options field, the argument names are not case sensitive, but the values are case sensitive. Arguments must be separated by semicolons, with no spaces in between.

The arguments that you can enter in the Connect Options field for Oracle include:

`HOST=host;SID=sid;CLIENT_VER=[8.1,9.2];SERVER_VER=[8.1,9.2,10.1];LOB_TYPE=[long, clob];PORT=port_number`

---

## Setting database properties for SQL Server

In ClearQuest release v2003.06.13, the fields shown on the database properties page for SQL Server are:

- Vendor (SQL Server)
- Physical Database Name
- Database Server Name
- Administrator Name
- Administrator Password

- Connect Options

After selecting SQL\_Server in the **Vendor** field, complete the fields as follows:

1. In the **Physical Database Name** field, enter the name of the database for the schema repository.
2. In the **Database Server Name** field, enter the machine name of the server where the SQL Server database is running.
3. In the **Administrator Name** field, enter the user name you created for the SQL Server database.
4. In the **Administrator Password** field, enter the password for the user name.
5. Leave the **Connect Options** field blank, if you are going to use the default port number and database instance for SQL Server on that machine. ClearQuest will determine these values automatically.

The arguments that can be entered in the Connect Options field for SQL Server are **PORT** and **INSTANCE**. ClearQuest uses port number 1433 as the default port. If the port number of the database is different from the default, then the port number should be entered in this field in the form **PORT=port\_number**. Note that argument names are not case sensitive, but the values are case sensitive. Arguments must be separated by semicolons, with no spaces in between.

ClearQuest also uses the default instance, which generally uses the same setting as the default port number. For SQL Server databases on Windows, you can specify an instance other than the default in the form, **INSTANCE=instance\_name**. For SQL Server databases on UNIX, you can specify an instance other than the default by specifying a non-default port number.



---

## Chapter 4. Changes to command-line utilities

This chapter contains changes to the **installutil** and **pdsq** command line utilities related to the new database driver implementation for ClearQuest v.2003.06.13 on Windows and patch 2004C on UNIX.

It also contains information about new commands for ClearQuest clients on Windows and UNIX:

- (Windows) **installutil registerconnectoptions** and **installutil getconnectoptions**
- (UNIX) **cqregrefresh**

---

### Updating connect options for installutil

The **installutil** command line utility includes a number of subcommands that can be useful when setting up or modifying databases. The command syntax has been updated for **installutil** with ClearQuest v.2003.06.13 for Windows and patch 2004C for UNIX. The changes affect how the connect options are specified.

Examples of the **installutil** subcommands that have been updated include:

- **installutil convertschemarepo**
- **installutil convertuserdb**
- **installutil unlockschemarepo**

### Specifying connect options in installutil for Oracle

If you are using an Oracle database, use the following arguments and values for connect options:

```
HOST=host;  
SID=sid;  
SERVER_VER=[8.1,9.2,10.1];  
CLIENT_VER=[8.1,9.2];  
LOB_TYPE=[long,clob];  
PORT=port_number
```

Note that the argument names are not case sensitive, but the values are case sensitive. Also, arguments must be separated by semicolons, with no spaces in between.

### Specifying connect options in installutil for SQL Server

If you are using a SQL Server database, use the following arguments and values for connect options:

```
PORT=port_number;  
INSTANCE=instance_name
```

Note that the argument names are not case sensitive, but the values are case sensitive. Also, arguments must be separated by semicolons, with no spaces in between.

---

## New subcommand — installutil registerconnectoptions

Use the installutil registerconnectoptions subcommand to modify connect option parameters for SQL Anywhere, SQL Server and Oracle on a ClearQuest Windows client. The subcommand operates on a per-client and per-session basis and should only be used in cases when you need to override a connection from a ClearQuest v.2003.03.16 client. Note that the argument names are not case sensitive, but the values are case sensitive. Also, arguments must be separated by semicolons, with no spaces in between.

A similar command, installutil registeroracleoptions, is available for the Oracle database, but you should use the installutil registerconnectoptions for Oracle as well as for SQL Anywhere and SQL Server.

### Syntax

**installutil registerconnectionoptions db\_vendor connect\_options**

Where	Represents
db_vendor	The database vendor name
connect_options	Oracle: <ul style="list-style-type: none"><li>• HOST=host;</li><li>• SID=sid;</li><li>• SERVER_VER=[8.1,9.2,10.1];</li><li>• CLIENT_VER=[8.1,9.2];</li><li>• LOB_TYPE=[long,clob];</li><li>• PORT=port_number</li></ul> SQL Server: <ul style="list-style-type: none"><li>• PORT=port_number;</li><li>• INSTANCE=instance_name</li></ul> SQL Anywhere: <ul style="list-style-type: none"><li>• SERVER_VER=[5.0,8.0]</li></ul>

---

## New subcommand — installutil getconnectoptions

Use the installutil getconnectoptions subcommand to view the database and connect option parameters you have set on a ClearQuest v.2003.06.13 Windows client for either a SQL Anywhere, SQL Server or Oracle database.

### Syntax

**installutil getconnectionoptions db\_vendor -all**

Where	Represents
db_vendor	Displays the connect option settings for a database vendor name. The database vendor is either Oracle, SQL Server or SQL Anywhere.
-all	Use -all in place of db_vendor to display connect options settings for all database vendors.

---

## Updating connect options for pdsql

**pdsql** is a command line SQL utility. It can be used to open an SQL session with any database vendor supported by ClearQuest. **pdsql** supports all common SQL commands. The command syntax has been updated for pdsql with ClearQuest v.2003.06.13 for Windows and patch 2004C for UNIX. The changes affect how the connect options are specified.

### Specifying connect options in pdsql for Oracle

If you are using an Oracle database, use the following arguments and values for connect options:

```
HOST=host;
SID=sid;
SERVER_VER=[8.1,9.2,10.1];
CLIENT_VER=[8.1,9.2];
LOB_TYPE=[long,clob];
PORT=port_number
```

Note that the argument names are not case sensitive, but the values are case sensitive. Also, arguments must be separated by semicolons, with no spaces in between.

### Specifying connect options in pdsql for SQL Server

If you are using a SQL Server database, use the following arguments and values for connect options:

```
PORT=port_number;
INSTANCE=instance_name
```

Note that the argument names are not case sensitive, but the values are case sensitive. Also, arguments must be separated by semicolons, with no spaces in between.

---

## Using cqregrefresh for ClearQuest clients on UNIX

The **cqregrefresh** subcommand can be used on ClearQuest clients on UNIX to refresh a connection whenever connection information changes.

Times when **cqregrefresh** should be used include when user databases are created or modified, and when schema repositories and user databases are moved.

**cqregrefresh** should also be used for each UNIX install area that uses the changed database connection. It does not need to be used on individual ClearQuest clients. Usually the updated connection information will be set in each user's login session.

The following example shows what the output is when you use **cqregshow -all** to display information on all the connections for a client:

```
axon 11: cqreg show -all
Database Set Name: WH0axonSS
Master Database: MASTR
Description:
Vendor:      SQL_SERVER
Server:      cqwds2
ConnectOptions:
Database:    WH0axonSS
User Database: PASNY
Description:  Sample Database
```

Vendor: SQL\_SERVER  
Server: cqws2  
ConnectOptions:  
Database: WH0axonSS

The **cqregrefresh—dbset** example shows how to update the connect options information for a particular connection:

```
axon 12: cqreg refresh -dbset UCM
```

The **cqregrefresh—all** example shows what the output is when you update the connect options information for all connections:

```
axon 13: cqreg refresh -all
Default version is 2003.06.00
Refresh database set WH0axonSS
Db registry PASNY needs to be updated
Db registry PASNY was updated successfully
Db registry MASTR needs to be updated
Db registry MASTR was updated successfully
```

---

## Chapter 5. API and Hooks Updates

This section contains new information that applies to the ClearQuest Perl and COM APIs, including information on working with and writing hook code in ClearQuest Designer. It includes updates to the Hooks chapter of the *Administrator's Guide* and to the *API Reference*.

The categories of API and Hook updates for this release are:

- New Content - includes new sections containing hooks or scripting information that is not available in the IBM Rational ClearQuest *Administrator's Guide* or IBM Rational ClearQuest *API Reference* for this release. There are new sections for actions and access control, nested actions, naming conventions, commit behavior, error checking and validation, and performance considerations.
- Corrections and other changes to documentation - include content updates to existing ClearQuest APIs as well as information on newly implemented (or newly supported and documented) APIs.

---

### New content

This section includes enhancements to the current ClearQuest API documentation available in the ClearQuest *Administrator's Guide* and the *API Reference*, including new content and additions to existing sections.

- Error checking and validation
- Debugging your code
- Actions and access control
- Name lookup in Perl hooks
- Default entity
- Editing an existing record
- When the record is committed
- Performance considerations for using hooks
- InvalidateFieldChoiceList example
- Using Perl for external applications
- RATLC00702699, APAR IC37754; Documentation for the highlighting of keywords in the ClearQuest Design...

### Error checking and validation

For many methods and properties of the ClearQuest API, you must check the return value to validate whether or not the call returns an error.

- For calls to functions that return an object, you need to check for the condition if the specified object does not exist. For example, if you call the `Item` method of a collection object, if the object that you specify is not in the collection, the return value is:
  - For Perl, an undefined object. You can use

```
if (undef($result)) { ... };
```

to detect this condition.
  - For VB, an error (`E_INVALIDARG`) that can be handled by the **On Error** statement.

- For calls to functions that have a String return value, the value is empty if there is no error, or a String containing the description of the error. You can check the result of calling the method and if the value is not empty, you can retrieve the error in a variable, as a String value.

For example the Entity object **SetFieldValue** method is defined as returning a String value. It returns an empty String if changes to the field are permitted and the operation is successful; otherwise, if the operation fails, this method returns a String containing an explanation of the error.

To trap the error, your code must check the return value. For example:

```
strRetVal = SetfieldValue ("Invalid_field", "Invalid value")
If "" <> strRetVal Then
    REM handle the error
End If
```

If an invalid field is specified, an error is returned. For example:

The Defect SAMPL00000123 does not have a field named "Invalid\_field".

## Debugging your code

You can debug your schema customization effort from within ClearQuest using a number of different utilities. One common method is to output text at strategic locations in the code, using **MsgBox** or **OutputDebugString**.

- **MsgBox**

This function is available on Windows only.

The **MsgBox** function lets you place a Windows Message Box on the screen with the output you specify. The execution of the hook pauses until the **OK** button on the Box is clicked (for example, **MsgBox "My Text."**). The message box only displays where the hook is executed.

When writing VBScript hooks, you can use the message box (**MsgBox**) function to output debugging information. By calling this utility with a string parameter, a popup dialog containing the text is displayed.

**Note:** Do not invoke this utility through ClearQuest Web. If you use the **MsgBox** function, you can ensure that your code is not executed in a Web session context with the **\_CQ\_WEB\_SESSION** session variable. See "Using hooks to detect a Web session" in the *Administrator's Guide* for more information.

- **DBWIN32**

The Windows debugging utility **dbwin32.exe** is included with ClearQuest for Windows. It is located in the ClearQuest installation directory. When **dbwin32.exe** is active, it displays all messages generated by the **OutputDebugString** method of the **Session** Object, which you can use to output debugging messages from a hook while it is running. By calling the **OutputDebugString** method, the related debug statements appear in the **DBWin32** console. Use this after launching **DBWin32** to see messages.

- **ClearQuest Designer hook compiler**

This utility catches some syntax errors.

- **Internet Explorer 4.0 debugger**

You can use the Internet Explorer 4.0 debugger to debug your hook code. You can download and install this debugger at the following address:

<http://msdn.microsoft.com/scripting> > Script Debugger

A hook runtime error launches the debugger (if it is not launched, you will need to read the debugger documentation). To force the debugger to be launched, add a **stop** statement to your hook code, and the debugger will be launched at that point.

- Microsoft Development Studio VBScript debugger  
General debugging of VBScript hooks can be done with the Microsoft VBScript Debugger. If you have Microsoft Visual Studio installed, you can use its VBScript debugger to debug your hook code.

## Actions and access control

An Access Control hook is used to determine whether a specific user is permitted to execute an action on records of a given record type. This hook is called before the user tries to execute the action.

Access to an action for a specific record type can be restricted through ClearQuest Designer by setting the authorization of the Access Control field in the Actions table for that record type.

By default, all users have access to all actions. However, you can restrict access to an action to specific user groups. For example, you can limit the ability to close defects to one specific user group.

Access to an action can also be restricted by using an access-control hook. For example, to restrict the ability to edit an Entity (that is, a record), an action access control hook can be written so that **EditEntity** (or **SetFieldChoiceList**, **SetFieldValue**, or **BuildEntity**) could be accessed only by users with the appropriate privileges. Or a hook could restrict access to the action **Open for Development** to the owner of the record.

Hooks always run with SuperUser privileges and therefore, are not subject to the usual access control or field behavior restrictions. For example, a hook could modify a field that is normally read-only. However, a hook cannot modify ClearQuest system fields, such as the History field.

When a hook executes, required fields remain required. However, a hook can dynamically change a required field so that it is no longer required, or can change a non-required field to required.

A hook does not change field validation rules, so data must still comply with those rules.

### Primary actions

Primary actions are main or top-level actions that are initiated by a user. Base and nested actions execute within primary actions and are not initiated directly by users.

- Access controls can be modified for actions created when a package is applied, just like they can be modified for any other type of action. However, any access control restrictions placed in a base action will apply to all other actions for that record type.
- Access control hooks are not run for nested actions. See Hooks in nested actions for more information.

**Note:** In order for a user to be able to run a primary action (modify, submit, delete, import, change\_state, duplicate, and unduplicate), the current user must be in the access control list for the primary action as well as for all the base actions. See Base actions for more information.

## Base actions

A base action is a secondary action that is triggered by a primary or top-level action. A base action is automatically triggered by every other action for that record type.

Base actions allow an action hook to be written once and then re-used with multiple actions. For example, writing a base action and adding a notification hook to send an email will cause an email to be sent when any action is performed on the record.

Each step of an action (initialization, access control, validation, commit, and notification) will execute the hooks of all base actions for that record type, followed by the hook for the main action itself.

A base action cannot be initiated directly by a user, so it does not appear in the list of possible actions presented to the user in the Actions menu.

There can be multiple base actions for a record type. Some base actions can be added to a schema when a package is applied.

If a record has multiple base actions, they do not run in a specific order but are followed by the main action that triggered them.

**Note:** Any access control restrictions placed in base actions apply to all other actions.

## Nested actions

A nested action is any action started when an action is already in progress.

Nested actions can be started only when a hook calls the **BuildEntity** or **EditEntity** methods of the **Session** object. Some actions can be both a primary action (initiated directly by the user) and a nested action (initiated by a hook).

**Note:** Nested actions trigger all base actions for that record type, just as primary actions do.

**Hooks in nested actions:** Nested actions differ from primary actions in that action access control hooks and notification hooks are not executed for nested actions.

The Action Access Control hook is not run if a hook starts a nested action. Because all hooks execute with the SuperUser privilege, the privilege level is already at its highest (SuperUser). There is no need to run the access control hook for the nested action.

Access for a nested action is also granted when no access control hook is fired.

Notification hooks do not normally execute for a nested action. Notification hooks are commonly used to send an email. Having each nested action send an email would result in many emails sent for what the user considers to be one action. You can override this behavior and allow nested actions to execute notification hooks by setting the **CQHookExecute** session variable to a value of **1**.

Setting the **CQHookExecute** session variable can be done with the following code:

- VBScript:

```

dim session
set session = GetSession
session.NameValue("CQHookExecute") = 1

```

- Perl:

```
$session->SetNameValue("CQHookExecute","1");
```

Within a Commit hook, the commit at the database level is not done when the nested action is committed, but is combined with the outer level commit so that all changes are included as one atomic transaction.

In all other hook types, a nested action is committed at the database level, independent of the outer level commit. The only way to combine changes made in a nested action with those of the top-level action, as a single database transaction, is to have the nested action inside a Commit hook.

See the *ClearQuest Administrator's Guide* for more information on execution order of hooks and when a record is committed. For setting field values, see the **SetFieldValue** method of the Entity object.

## Name lookup in Perl hooks

Variables in Perl are of several types. Common types include:

- **my** variables, which are local to the subroutine in which they are declared.
- Local variables, which are local to the file in which they are declared (but global to all subroutines within that file).
- Global variables, which are not declared explicitly.

You must specify global variables with unique names. You can also use local variables, using the **my** convention. For example:

```
my ($uvComponent);
```

It is possible for existing Perl hooks to have name look-up problems. A ClearQuest action, Submit for example, uses a single Perl interpreter to execute all the action hooks, like Action Initialization, and all the field hooks, like Field Value Changed, that are written in Perl. If the hook code does not declare local variables with the **my** keyword before it uses them, the variable can be shared between hooks unintentionally.

In releases before v2003.06.00, when one Perl hook called another, ClearQuest compiled the second hook in the wrong Perl namespace. In release v2003.06.00 and later, all hooks are compiled in the same Perl namespace. This can affect how different hooks can interfere with each other if global variables are used. For example:

```

sub defect_Initialization
{
    $variable = "1";
    $entity->SetFieldValue("A", $variable);
}
sub a_ValueChanged
{
    $entity->SetFieldValue("B", $variable);
    $entity->SetFieldValue("C", $variable);
}
sub b_ValueChanged
{
    $variable = "3";
}

```

The variable called `$variable` in these hook subroutines is a package scope variable. This means that subroutines in the same package will share the same variable.

In releases before v2003.06.00, nested hooks were in a different Perl package from the initial hook, and therefore did not share their global variables with the initial hook. This meant that the above example was interpreted as if it had the following namespace qualifiers:

```
package main;
sub defect_Initialization
{
    $main::variable = "1";
    $entity->SetFieldValue("A", $main::variable);
}
package CQEntity;
sub a_ValueChanged
{
    # $CQEntity::variable is not set, so defaults to an empty string.
    $entity->SetFieldValue("B", $CQEntity::variable);
    $entity->SetFieldValue("C", $CQEntity::variable);
}
sub b_ValueChanged
{
    $CQEntity::variable = "3";
}
```

A Field Value Changed hook is called immediately upon changing the associated field (that is, before returning from `SetFieldValue`), so the above code executes in this order:

```
$main::variable = "1";    # From defect_Initialization
    # $main::variable is set to "1"
$entity->SetFieldValue("A", $main::variable); # From defect_Initialization
    # Sets A to "1"
    # ClearQuest calls a_ValueChanged before returning
$entity->SetFieldValue("B", $CQEntity::variable); # From a_ValueChanged
    # $CQEntity::variable is uninitialized
    # Sets B to ""
    # ClearQuest calls b_ValueChanged before returning
$CQEntity::variable = "3"; # From b_ValueChanged
    # $CQEntity::variable changes from "" to "3"
$entity->SetFieldValue("C", $CQEntity::variable); # From a_ValueChanged
    # Sets C to "3"
```

As a result, fields A, B and C are set to "1", "" and "3", respectively.

In release v2003.06.00 and later, Perl hooks are compiled into the same namespace. The above example is now interpreted as if it has the following namespace qualifiers:

```
package main;
sub defect_Initialization
{
    $main::variable = "1";
    $entity->SetFieldValue("A", $main::variable);
}
sub a_ValueChanged
{
    $entity->SetFieldValue("B", $main::variable);
    $entity->SetFieldValue("C", $main::variable);
}

sub b_ValueChanged
{
    $main::variable = "3";
}
```

This code executes in this order:

```
$main::variable = "1";    # From defect_Initialization;
    # $main::variable is set to "1"
$entity->SetFieldValue("A", $main::variable); # From defect_Initialization
    # Sets A to "1"
    # ClearQuest calls a_ValueChanged before returning
$entity->SetFieldValue("B", $main::variable); # From a_ValueChanged
    # Sets B to "1"
    # ClearQuest calls b_ValueChanged before returning
$main::variable = "3";    # From b_ValueChanged
    # $main::variable changes from "1" to "3"
$entity->SetFieldValue("C", $main::variable); # From a_ValueChanged
    # Sets C to "3"
```

As a result, fields A, B and C are set to "1", "1" and "3", respectively.

To avoid unintentional sharing of variable values, you must declare those variables intended to be local to a hook function in this form:

```
sub d_ValueChanged
{
    my $temp = $entity->GetFieldValue("d")->GetValue();
    $session->OutputDebugString("d now set to $temp\n");
}
```

where \$temp is declared to be local to the d\_ValueChanged function, and this assignment to \$temp does not change the value of another variable of the same name. Using the my syntax makes the variable visible only within a specified block of code.

**Note:** \$entity and \$session are global variables defined by the ClearQuest core.

## Default entity

The default entity for a hook is created by ClearQuest before a hook starts, and represents the current record upon which an action is being done.

For Perl, you retrieve this object using:

```
$entity
```

If you are creating another entity in the context of one record's action (such as executing a call to the **BuildEntity** method to submit a new record), you need to maintain the scope of your entity variables. There are two approaches:

- Use the same variable name for both entities, but declare one of them using **my**. That makes only one of them accessible at any time, because the **my** declaration creates a new variable with local scope which masks the existing global definition until the variable goes out-of-scope (for instance, at the end of the current function).
- Use different variable names for each entity.

For VBScript, you can use the **me** declaration. For example:

```
call DoSomething(me)
```

If you need to explicitly reference the default entity object in order to pass it as an argument, you can use the me declaration to perform operations. For example:

```
Msgbox me.GetDisplayName()
Dim xme
Set xme = me
Msgbox xme.LookupStateName
```

## Editing an existing record

This section of the ClearQuest *API Reference* should include the following additional information on database locking.

Only one user at a time can edit a record. If two users attempt to edit a record at the same time, ClearQuest allows only one of them to commit their changes. The first user who validates and commits their changes is successful. When the other user tries to commit their changes, they receive an error stating that the record has been updated while they were editing, and their changes cannot be safely committed.

## When the record is committed

The Commit hook executes after the database has been updated with changes to the current record, but before the update transaction has been committed to the database. This means that you cannot use a Commit hook to modify the current record; such modifications are not applied to the record.

Work done in a Commit hook is done while locks exist in the database, and those locks may prevent other users from running queries, creating new records, or modifying existing records. For performance reasons, it is best to minimize the work done in a Commit hook.

Use a Commit hook only for actions against other records that you want to be part of the same database transaction as the main action. For example, resolving a duplicate defect when the parent defect is resolved. You must ensure that you are placing the appropriate calls in the correct context. For example, you would not call Revert from a Commit hook, nor would you call Commit from any action other than a Commit hook.

## Performance considerations for using hooks

ClearQuest supports the use of VBScript or Perl for writing your custom hook code. However, there are performance and functional trade-offs that should be considered when choosing the scripting language and the types of operations to use in hooks. Although this is not an exhaustive discussion on the topic, the following guidelines should be applied to any schema modifications. For more information on the topic of ClearQuest Schema Performance, see the IBM developerWorks Web site.

- ClearQuest Web

The ClearQuest native Windows client can execute hooks written in either VBScript or Perl. The same is true for the ClearQuest Web server, since it runs in a Windows environment. However, the ClearQuest native UNIX/Linux clients can execute only Perl script. Therefore, if a ClearQuest deployment requires any native UNIX/Linux clients, hooks must be written in Perl. If you are deploying ClearQuest Web exclusively, or only in combination with Windows clients, there are performance advantages on the ClearQuest Web server to choosing VBScript as your hook scripting language.

- Database Access

Accessing the database is typically the most time consuming operation a hook performs. Examples of operations that require database access are:

- LoadEntity and GetEntity operations

Retrieving an entity (record) requires at least one query of the database for the primary record, plus one query for each REFERENCE or REFERENCE\_LIST field. Entities are retrieved explicitly through Session

methods such as **GetEntity** or **LoadEntity**, but can also be retrieved implicitly by accessing the field value of a **REFERENCE** field. The following example implicitly loads the entity referred to by the **product** field, and then retrieves the value of the component field from the loaded **product** record:

```
$component = $entity->GetFieldValue("product.component")->GetValue();
```

The time to load an entity is determined by the complexity of the schema, primarily by the number of reference list fields in the selected entity. In most instances, if only a subset of an entity's fields are required, it is more efficient to query for those field values instead of retrieving the entire entity.

- Queries

Although more efficient than retrieving entire entity records, queries still require database access, and therefore have an impact on your overall schema performance. Every effort should be made to minimize the number of database round-trips. For instance, rather than running the same query multiple times at various locations in the hook code, a query can be executed once, and the **ResultSet** values can be cached in a **Session** variable. Also, retrieve only the fields that are essential for each record. Avoid specifying multiline text fields in query result sets, as this requires an additional database round-trip for each multiline text field to be retrieved.

- Choice lists with the Recalculate Choice List option set

When you choose the **Recalculate Choice List** option in the properties of a choice list, the hook code required to repopulate the valid list of choices is executed each and every time any other field of the record changes value. This has the potential to cause a large amount of unnecessary query traffic to and from the database. A more efficient method to ensure that the choice list is valid is to determine the other fields that can affect the values in this choice list, and force the choice list to be recalculated only when those field values change.

For example, if you are collecting data in an **Automobile** record you might have a field for the **Manufacturer** of the automobile, and another field for the **Model**. The valid list of choices for the **Model** field depends only on the **Manufacturer** selected. The inefficient method of ensuring that the choice list for the **Model** field is always valid is to select **Recalculate Choice List** for this field. Instead, you can write a **VALUE\_CHANGED** hook for the **Manufacturer** field that invalidates the choice list for the **Model** field. See **InvalidateFieldChoiceList** example for more information on using this method.

- Cascading Hooks

Cascading hooks are caused by having several dependent or nested relationships between fields. Consider the automobile **Manufacturer** and **Model** dependency discussed earlier in this section. Extending that example, suppose that once a **Model** is selected, the list of valid choices for **Body Style** or **Color** or **Engine** could change. It is easy to see how changing one field value on a form could cause a cascade of hooks to be executed and re-executed for the other fields. The depth of these nested field relationships should be minimized, and care should be taken in the implementation of the schema in order to avoid unnecessary or redundant execution of hook code.

- AdminSession objects

Getting **AdminSession** objects has an impact on performance and there may be alternatives for retrieving data. For example, instead of using the **AdminSession** object and underlying **User** object and **Group** object methods to retrieve user or group information, you can create queries for **User** and **Group** records (stateless record types) that are in a user database.

If you must use an AdminSession object, you can cache it in a Session variable instead of creating new AdminSession objects for each login or hook invocation that requires it. Additionally, if the data you retrieve through the AdminSession object is not changing, then you can cache the data as values in Session variables.

## InvalidateFieldChoiceList example

In the following example, a Defect record type has the two fields, **product** (a reference to the **Product** record type) and **owner** (a reference to User). Each Product record type has a field called **contributors** (a reference list to User).

In order for the **owner** choice list field to be updated whenever the value for Product is changed, you can use the **InvalidateFieldChoiceList** method, instead of using the Recalculate Choice List option.

For example, in the Defect record type, you add a value changed hook for the field product

- **Perl**

```
sub product_ValueChanged {
    my($fieldname) = @_;
    # $fieldname as string scalar
    # record type name is Defect
    # field name is product
    # Make sure that the choice list for owner is based on
    # this new value for product.
    $entity->InvalidateFieldChoiceList("owner");
}
```

and you add a choice list hook for the field owner.

```
sub owner_ChoiceList
{
    my($fieldname) = @_;
    my @choices;
    # $fieldname as string scalar
    # @choices as string array
    # record type name is Defect
    # field name is owner
    # Is the value of product set? If not, return an empty list.
    my $productFieldInfo = $entity->GetFieldValue("product");
    return @choices unless
    $productFieldInfo->GetValidationStatus() == $CQPerlExt::CQ__KNOWN_VALID;
    return @choices unless $productFieldInfo->GetValue() ne "";
    # Field product is set and valid.
    # Get the list of contributors on this product.
    @choices = $entity->GetFieldValue("product.contributors")
        ->GetValueAsList();
    return @choices;
}
```

- **VBScript**

```
Sub product_ValueChanged(fieldname)
    ' fieldname As String
    ' record type name is Defect
    ' field name is product
    InvalidateFieldChoiceList "owner"
End Sub
```

and you add a choice list hook for the field owner

```
Sub owner_ChoiceList(fieldname, choices)
    ' fieldname As String
    ' choices As Object
    ' record type name is Defect
    ' field name is owner
```

```

        ' Is the value of product set? If not, return an empty list.
Dim productFieldinfo
set productFieldinfo = GetFieldValue("product")
if productFieldinfo.GetValidationStatus() <> AD_KNOWN_VALID then exit sub
productFieldInfovalue = productFieldinfo.GetValue()
if productFieldInfovalue = "" then exit sub

        ' Field product is set and valid.
        ' Get the list of contributors on this product.
Dim productFieldvalues
productFieldvalues = GetFieldValue("product.contributors").GetValueAsList()
for each contributor in productFieldvalues
    choices.AddItem contributor
next

End Sub

```

Each time you start an action, **owner\_ChoiceList** is run once, and every time you change product, the owner choice list is invalidated. The user interface then requests the choice list, which forces the choice list hook to be re-executed.

## Using Perl for external applications

In version 2003.06.00 and later, you can use the ClearQuest API with either CQperl.exe or ratlperl.exe. Using CQperl implicitly adds the correct include paths to CQPerlExt.pm (the Perl package that provides the ClearQuest API). If you use ratlperl on UNIX, you must set the correct path. If you use Perl for an external application, IBM Rational recommends that you limit the external application to tasks that are independent of actions, such as querying, reporting, and user administration.

## RATLC00702699, APAR IC37754; Documentation for the highlighting of keywords in the ClearQuest Designer script editor

This section describes how to customize the coloring and display of hook code in the ClearQuest Designer script editor for Perl or VBScript hook code.

When working with hook code in the scripting editor in ClearQuest Designer, there is a limited set of colors in the presentation. The tool displays all comments in green, read-only text in grey, and all other text in black. These color settings are defined in two ini files in the Rational/ClearQuest directory:

- VBScript.ini, for the COM API
- PerlScript.ini for the Perl API

You can enhance the color settings by customizing these files. Each color setting is defined as a color group (**ColorGroup**). Each of the ini files includes a set of defined color groups.

Two forms of customization are:

- Modifying the colors of an existing color group
- Adding new color groups

Each color group has a Foreground attribute that defines the color of the text and a Background attribute that defines the background color for the text.

## Modifying the colors of an existing color group

You can specify the color settings of the text color and its background for an existing color group by modifying the Foreground or Background attributes of that color group. Each of these attributes requires three color values (red, green, blue) in the range of 0-255. For example:

```
ColorGroup = Comment  
Foreground = 238,183,17  
Background = 239,239,211
```

You can modify the values for any given color group to customize the settings that ClearQuest provides. For example, change:

```
Foreground = 238,183,17
```

to

```
Foreground = 238,128,0
```

## Adding color groups

You can add color groups that define new classes of information to help distinguish different classes of text (defined as different color groups) in the script editor. For example, you can define a new color group to highlight enumerated constants (that are defined in ClearQuest.bas) by defining the following CQConstant color group:

```
[CQConstant]  
Foreground = 75,0,255  
Background = 239,239,211  
DisplayName = CQConstant  
Configurable = 1  
BackColorAutomatic = 1  
ForeColorAutomatic = 0  
Configurable = 1
```

Within the ini file is a variable (**NumGroups**) that is set to the number of color groups. For example:

```
NumGroups=14
```

For each new color group you define, you must increment the value of NumGroups by one.

Add the enumerated constants to the Keyword list and set them to the defined ColorGroup name. For example:

```
[Keywords] AD_SUPER_USER = CQConstant
```

In addition to making the constants more visible, this solution can help identify spelling errors, since a misspelled constant would not be defined as a Keyword and therefore not highlighted.

**Note:** The definitions in the ini files are not directly linked to the actual ClearQuest APIs. Therefore, changes to an API are not reflected in the ini files. Depending on your customizations, manual updates to the ini files may be required.

**Note:** Customizing the VBScript.ini and PerlScript.ini files has no effect on any code that is part of an installed package. Package installed code is read-only.

---

## Corrections and other changes to documentation

This section includes:

- Content updates to existing ClearQuest API functions that are documented in the *API Reference*.
- New content, including newly supported or documented methods that should be included in the *API Reference*.
- Newly implemented methods that should be included in the *API Reference*.

### RATLC00708226, RATLC00706668, RAMBU00050315; Commit behavior documentation enhancements

Additional information should be included in the ClearQuest *Administrator's Guide* on **Commit** behavior, in the Hooks chapter, in the "When the record is committed" section. See When the record is committed for the updated documentation on **Commit** behavior.

For additional information, see Editing an existing record and Hooks in nested actions.

### RATLC00712744; Syntax is incorrect for the GetFieldRequiredness method

The VBScript syntax for the **GetFieldRequiredness** and the **GetFieldType** methods of the **Entity** object is incorrect in the ClearQuest *API Reference*. Parentheses should be included for the arguments:

- `entity.GetFieldType (field_name)`
- `entity.GetFieldRequiredness (field_name)`

### RATLC00712920, RATLC00710309, RAMBU00054358, RAMBU00056057; Date timestamp issues

The function for the current timestamp is incorrect in the ClearQuest *Administrator's Guide*. The correct code for the **GetCurrentDate** function is:

```
sub GetCurrentDate {  
    my($sec, $min, $hour, $mday, $mon, $year, $wday, $yday, $time) =  
        localtime();  
    return sprintf("%4d-%2.2d-%2.2d %2.2d:%2.2d:%2.2d", $year + 1900,  
        $mon + 1, $mday, $hour, $min, $sec);  
}
```

### RATLC00696630, RATLC00696270, RATLC00696759, RATLC00710896; Generating reports and updates to SetHTMLFilename documentation

The following information helps resolve issues raised with generating reports using the API and receiving an Unknown exception. This additional information applies to the **SetHTMLFilename** method of the report manager (**ReportMgr**) object.

You must call this method before calling the **ExecuteReport** method to set the name and location of the report output file. You specify output path information in the **htmlPathName** parameter of the **SetHTMLFileName** method:

- VBScript  
`reportMgr.SetHTMLFileName htmlPathName`

- Perl  
`$reportMgr->SetHTMLFileName(htmlPathName);`

You must specify a directory for the HTML file or add a backslash ("\") before the file name. For Perl, use two backslashes ("\\"). For example:

- VBScript:  
`c:\test.html`  
`\\test.html`
- Perl:  
`c:\\temp\\my-report.html`  
`\\my-report.html`

This SetHTMLFileName method argument requires a full path name to the file to be created. If the file is to be exported to the current directory, a file separator needs to be included before the name of the file. For example (Perl):

```
$CQReportMgr->SetHTMLFileName("\\Output.html");
```

## RATLC00705428; SuperUser privilege required for SetUserName method

The SetUserName method of the Workspace object requires SuperUser privileges. The section in the ClearQuest *API Reference* for this method should include the following note.

**Note:** Users must have SuperUser privileges to call this method.

## RATLC00707609; Methods to set and get user privileges

Two new methods in the User object for setting and retrieving user privileges are available in the Perl API, but are not documented in the ClearQuest *API Reference*.

- GetUserPrivilege
- SetUserPrivilege

### GetUserPrivilege

- **Description**

Tests whether the User has a specified user privilege. Returns True if the User has the specified user privilege; False if they do not have the specified user privilege.

**Note:** This method is for Perl only. It is not available for VBScript.

- **Syntax**

**Perl**

```
$user->GetUserPrivilege(priv);
```

- 

Identifier	Description
user	A User object.
priv	A Long containing a <b>UserPrivilegeMaskType</b> constant.
Return value	Returns a Boolean True if the User has the specified user privilege; False if they do not have the specified User privilege.

- **See Also**

- SetUserPrivilege

- HasUserPrivilege method of the Session Object
- UserPrivilegeMaskType constants

## SetUserPrivilege

- **Description**

Sets a User privilege to True if authorizing the User privilege to the user's account; False if not allowing the User privilege.

**Note:** This method is for Perl only. It is not available for VBScript.

- **Syntax**

**Perl**

```
$user->SetUserPrivilege(priv, bValue);
```

Identifier	Description
user	A User object.
priv	A Long containing a <b>UserPrivilegeMaskType</b> constant.
bValue	A Boolean set to True if the User has the specified user privilege; False if they do not have the specified User privilege.
Return value	None on success, or else an exception.

- **See Also**

- GetUserPrivilege
- HasUserPrivilege method of the Session Object
- UserPrivilegeMaskType constants

## RATLC00666959, RATLC00698133, RAMBU00046241; Creating PERL and VBScript Hooks of the same name causes the creation of new hooks to fail

Creating a Perl and VBScript record hook with the same name causes the following problems when a field has one of the hooks assigned to it on a form:

- The error message

All forms will be saved and closed before you can delete or rename a field. Continue?

is displayed. Because the dialog box has an **OK** button only, you cannot cancel the action, and the field is not removed from the form.

- The new hook (with the requested name) is created along with another hook, New1. This prevents the creation of any new hooks for either language, and you cannot delete the New1 hook because it does not appear in the list of record scripts.

The workaround is to restart ClearQuest Designer and delete the script. When the scripts are parsed, the New1 script is added to the tree view and can then be deleted.

## RATLC00712994, RAMBU00036659; CQString is not MBCS, which is not suitable for internationalization

**CQString** is the string type used by the ClearQuest Perl API. This type is used for the Perl API, so any Perl hooks or external Perl scripts use it to pass string data to and from the ClearQuest core.

CQString uses single-byte characters. To make it use wide characters, the core must be compiled with `_UNICODE`, but ClearQuest is compiled with `_MBCS`.

## **RATLC00703293, APAR IC37932; SetLoginName method update**

The **SetLoginName** method does not work as described in the ClearQuest *API Reference*. The following sentence should be removed and is not accurate:

If either a blank user name or password is supplied, no error will occur and only the parameter specified will be changed.

Neither argument is optional. If you add the password parameter, then the value you specify becomes the new password. If you leave out the name parameter, a type mismatch error is returned.

## **RATLC00701671, APAR IC37619; Updates to the description of the UserLogon method database set argument**

The description in the ClearQuest *API Reference* for the `database_set` argument of the **UserLogon** method should be updated as follows:

A String that specifies the name of the database set or connection string.

**Note:** You can use an empty string ("" ) if you have only one database set or to refer to the default database set. The default database set name is the one that matches the product version number (for example, 2003.06.00).

## **RATLC00712943, RATLC00712310, APAR IC39076; Updates to the SetFrom method of the Mail message object**

The **SetFrom** method section of the ClearQuest *API Reference* should include the following additional notes.

**Note:** When sending SMTP e-mail on a Web server, the server machine name is used as the "From" part of the message instead of the submitter e-mail address, unless **SetFrom** is explicitly used.

**Note:** The **SetFrom** method does not work with MAPI.

## **RATLC00705405; Correction to "Running a Query and Reporting on its Result Set" code example**

In the Perl code example in the "Running a Query and Reporting on its Result Set" section of the ClearQuest *API Reference*, the following statement

```
While($status == AD_SUCCESS)
```

should be

```
While($status == CQPerlExt::CQ_SUCCESS)
```

## **RATLC00453581, RAMBU00050338, RAMBU00035392, RATLC00656939, RATLC00712567, RATLC00710254, RATLC00705491, RAMBU00009075, RAMBU00010073, RATLC00654966, RAMBU00050417; Actions and access control documentation enhancements**

New content should be included in the ClearQuest *Administrator's Guide* that helps resolve issues with actions, access control, nested actions, and notification hooks not running.

For new information on actions and access control, base actions, nested actions information, see Actions and access control and its subsections.

**Note:** IBM recommends that you do not set any access control on Base actions. You can modify the access control to actions, including actions that may be added to your schema, by applying packages. However, any access control restrictions placed in base actions apply to all other actions.

## **RATLC00447393; Setting a field value or variable**

When setting a variable to be local in a Perl hook, use the **my** declaration.

Additional information to what the ClearQuest *API Reference* provides on setting a field value (that is, using the **SetFieldValue** method) is available in this release. See Name lookup in Perl hooks.

## **RAMBU00036184; Naming a field**

You cannot declare a constant or variable with the name being the same as an existing field name.

## **RATLC00705480, RAMBU00054500; New methods that enhance performance**

New methods are available that provide improved performance of existing functionality. The following methods provide shortcuts to functions provided by other existing APIs.

- **Entity** object methods:
  - **GetFieldStringValue**
  - **GetFieldStringValueAsList**
  - **GetFieldStringValues**
  - **SetFieldValues**
- **GetAllColumnValues** method of the **ResultSet** object

### **GetFieldStringValue**

- **Description**

Returns the list of values of the specified field as a single String.

This method is equivalent to first calling the **GetFieldValue** method to obtain a **FieldInfo** object and then calling the **GetValue** method of the **FieldInfo** object. This is a more direct and more efficient way to get the value of a field.

- **Syntax**

**VBScript**

*entity.GetFieldStringValue field\_name*

## Perl

```
$entity->GetFieldStringValue(field_name);
```

Identifier	Description
entity	An Entity object representing a user data record. Inside a hook, if you omit this part of the syntax, the Entity object corresponding to the current data record is assumed (VBScript only).
field_name	A String that identifies a valid field name of an Entity.
Return value	A String that contains the value or values stored in the field.

- **See Also**
  - GetValue of the FieldInfo Object
  - GetFieldValue
  - GetFieldStringValues
  - GetFieldStringValueAsList
  - SetFieldValues
  - SetFieldValue

## GetFieldStringValueAsList

- **Description**

Returns a list of string values for the specified field.

This method is equivalent to first calling the **GetFieldValue** method to obtain a **FieldInfo** object (and then calling **GetValueAsList** method of the **FieldInfo** object). This is a more direct and more efficient way to get the value of a field.

- **Syntax**

### VBScript

```
entity.GetFieldStringValueAsList field_name
```

### Perl

```
$entity->GetFieldStringValueAsList(field_name);
```

The *field\_name* argument is a String that identifies a valid field name of an Entity.

Identifier	Description
entity	An Entity object representing a user data record. Inside a hook, if you omit this part of the syntax, the Entity object corresponding to the current data record is assumed (VBScript only).
field_name	A String that identifies a valid field name of an Entity.
Return value	For VBScript, a 1-element Variant Array is returned. The Variant contains the list of values, separated by vbLF. If the field contains no values, this method returns an Empty Variant. For Perl, a reference to an array of strings containing the values in the list.

- **See Also**
  - GetValueAsList of the FieldInfo Object
  - GetFieldStringValues
  - SetFieldValues
  - SetFieldValue

## GetFieldStringValues

- **Description**

This **Entity** method allows multiple field values to be retrieved with one call. The *field\_names* parameter is a String array of field names, and the result is a String array of field values, in the same order as the input array. If there is an error retrieving any one of the named fields (for example, if you specify an invalid name of a field), an exception is thrown.

- **Syntax**

**VBScript**

*entity*.**GetFieldStringValues** *field\_names*

**Perl**

*\$entity*->**GetFieldStringValues**(*field\_names*);

Identifier	Description
entity	An Entity object representing a user data record. Inside a hook, if you omit this part of the syntax, the Entity object corresponding to the current data record is assumed (VBScript only).
field_names	For VBScript, a Variant containing an array of strings. Each String identifies a valid field name of this Entity object. For Perl, a reference to an array of strings containing the valid field names.
Return value	For VBScript, a Variant containing an array of strings. Each String contains the value or values stored for each of the specified field names. For Perl, a reference to an array of strings containing the value or values stored for each of the specified field names.

- **See Also**

- GetFieldStringValue
- GetFieldStringValueAsList
- SetFieldValues
- SetFieldValue

## SetFieldValues

- **Description**

Places the specified values in the named fields. This method allows multiple field values to be set with one call. The two input string arrays are parallel lists, where *field\_names* lists the field names and *new\_values* lists the field values. For example, item N in *field\_names* provides the field name and item N in *new\_values* provides the value for that field.

The return value is an array of result messages for each field. Each result message is the same message that is returned by a single call to the **SetFieldValue** method. If there are no errors, the result is a String array of the same number of elements as *field\_names*, with each element being an empty String.

- **Syntax**

**VBScript**

*entity*.**SetFieldValues** *field\_names*, *new\_values*

**Perl**

*\$entity*->**SetFieldValues**(*field\_names*, *new\_values*);

Identifier	Description
entity	An Entity object representing a user data record. Inside a hook, if you omit this part of the syntax, the Entity object corresponding to the current data record is assumed (VBScript only).
field_names	The list of field names for values to be set. For VBScript, a Variant containing an array of strings. Each String contains a valid field name of this Entity object. For Perl, a reference to an array of strings containing the valid field names.
new_values	The list of field values to set for the specified field names. For VBScript, a Variant containing an array of strings. Each String contains a field value. For Perl, a reference to an array of strings containing the new values.
Return value	For VBScript, a Variant containing an array of result messages for each field. For Perl, a reference to an array of strings containing the result messages for each field. If changes to the field are permitted, this method returns an empty String; otherwise, this method returns a String containing an explanation of the error.

- **See Also**

- GetFieldStringValues
- GetFieldStringValueAsList
- GetFieldStringValue
- SetFieldValue

## GetAllColumnValues

- **Description**

Returns all column values in the result set as an array of strings (for Perl, a reference to an array of strings).

The result contains a pair of strings for each item:

- The first string of each pair is the column name (in uppercase).
- The second string is the column value.

For example, given a query asking for Id and Headline, a successful result of calling this method would be an array containing 4 elements:

- "ID"
- "SAMPL00001234"
- "HEADLINE"
- "This is a test"

By returning a pair of strings for each column, you can design code independent of the order of the items in the result set.

The *MoveNext* parameter controls whether this function calls the **MoveNext** method before retrieving the data. If there is an error executing the **MoveNext** method, the return value of **GetAllColumnValues** is a result array containing one element, the String form of the **ErrorFetchStatus** value that would have been returned from the **MoveNext** method. A non-error result always has an even number of elements in the array.

- **Syntax**

### VBScript

```
resultset.GetAllColumnValues MoveNext
```

### Perl

```
$resultset->GetAllColumnValues(MoveNext);
```

Identifier	Description
resultset	A ResultSet object, representing the rows and columns of data resulting from a query.
MoveNext	A Boolean that specifies whether or not to call the MoveNext method before retrieving all of the column values for a row in the result set.
Return value	For Visual Basic, a Variant array of strings containing the column names and column values. For Perl, a reference to an array of Strings containing the column names and column values.

- **See Also**
  - MoveNext
  - Execute
  - GetColumnLabel
  - GetColumnType
  - GetNumberOfColumns

## **RATLC00450645, RAMBU0046105; ClearQuest hooks database location has changed**

The location of the ClearQuest Hooks database that is listed in the *ClearQuest API Reference* and in the *ClearQuest Administrator's Guide* is no longer correct nor in service. You can access the hooks by navigating to:

- <http://www.ibm.com/developerworks/rational/library/4236.html>
- <http://www.ibm.com/developerworks/rational/products/clearquest> and selecting "IBM Rational ClearQuest hooks index."

## **RATLC00703780, RAMBU0010103, APAR IC41898; Package-installed hooks are read-only**

The following information on package-installed hooks should be added to the *ClearQuest Administrator's Guide* hooks chapter:

When you install a package, hooks may be added to your schema. However, these scripts are part of the Package and not part of your hook code.

Package-owned scripts cannot be deleted. They are read-only and not part of the code owned by a schema. For this reason, there is no relationship between the default language setting you choose for your hook code and the language that hooks owned by a Package are implemented in.

## **RATLC00699730; Code example for HasDuplicates correction**

The code example for the **HasDuplicates** method in the *ClearQuest API Reference* is incorrect. Here is the correct code:

```
$originalID = $entity->GetDisplayName();
if ($entity->HasDuplicates())
{
    $session = $entity->GetSession();
    $duplicateLinkList = $entity->GetDuplicates();
    $cnt = $duplicateLinkList->Count();
    # Output the IDs of the parent/child records
    for ($i = 0; $i<$cnt; $i++)
    {
        $itm = $duplicateLinkList->Item($i);
        $duplicateObj = $itm->GetChildEntity();
    }
}
```

```

$duplicateID = $duplicateObj->GetDisplayName();
$session->OutputDebugString("Parent ID:".$originalID." child
    Id:".$duplicateID);
}
}

```

## RATLC00697318, RATLC00708183; StringIdToDbId method of the Session object

The description in the *API Reference* for the **StringIdToDbId** method of the **Session** object includes information about the format of IDs for stateless record types. However, the method does not work for stateless record types. The **StringIdToDbId** method (and also the **DbIdToStringId** method) accept IDs of the form used for stateful record types, either a record number by itself or with the database name at the front (for example, SAMPL00001234).

- The documentation for both the **StringIdToDbId** and **DbIdToStringId** methods should include the following note:

**Note:** This method does not currently support stateless record types.

- The description in the *API Reference* for the **StringIdToDbId** method should be: Returns the database ID (DbId) translated from string ID. The DbId is a unique number assigned to every record by ClearQuest. For stateful records, the string ID is the display name (for example, SAMPL00001234).

**Note:** This method does not currently support stateless record types.

## RATLC00705438; Perl API Build method syntax

The correct Perl syntax for getting a ClearQuest Session is:

```
$CQsession = CQSession::Build();
```

While there are also examples available using

```
$CQsession = CQPerlExt::CQSession_Build();
```

the correct syntax is to use the **Build** method of the **Session** object. For example:  
 use CQPerlExt; my \$sessionObj = CQSession::Build(); CQSession::Unbuild(\$sessionObj);

**Note:** This syntax applies for all **Build** methods available (such as, in the **AdminSession** object) in the ClearQuest Perl API.

## RATLC00701064, RATLC00705313; Database object password methods require SuperUser privilege

The following Database object password properties require the SuperUser privilege.

- **DBOPassword**
- **ROPassword**
- **RWPassword**

The *API Reference* should include the following note in each section for these methods.

**Note:** You must have SuperUser privileges for this method to return the password. For users without the SuperUser privilege, an exception is thrown.

## RATLC00698109, RATLC00696104; UNIX support for reports in a workspace

The ClearQuest Perl API now supports creating and editing report functionality for UNIX. Perl support has been added for the following two methods:

- CQWorkSpaceMgr->**GetReportMgrByReportDbId**(\$report\_dbid);
- CQReportMgr->**GetQueryDef**();

These methods have a note in the *API Reference* that they are for Windows only. This is no longer true and the sentence should be removed. The methods are now supported for UNIX.

## RATLC00703013, RATLC00713905, RATLC00702914, APAR IC37813; cqole.odl and cqole.dll mismatch

In version 2003.06.00, there was a version mismatch between two files affecting the ClearQuest COM API (cqole.odl and cqole.dll) that causes some discrepancies between what is documented in the *API Reference* and what is visible in an object browser. In particular, the **GetSuiteProductVersion** method of the **Session** object and the methods supporting code page settings in the **Session** and **AdminSession** objects do not appear in cqole.odl nor in an object browser, although they are included in cqole.dll.

With the fix in version 2003.06.13, the new **Session** and **AdminSession** object methods are now visible in an object browser.

In the *API Reference*, the **GetSuiteProductVersion** method of the **Session** object is documented. For backwards compatibility, there should also be the **GetSuiteVersion** method. This method returns the same value as **GetSuiteProductVersion**.

### GetSuiteVersion

- Description

Returns the Suite version string. This is the same version string as the one returned by the Suite version DLL file and displayed in the **About** box of ClearQuest. You do not need to be logged in to a database to use this method.

**Note:** This method is for COM only. For Perl, see the **ProductInfo** Object.

- Syntax

#### VBScript

session.GetSuiteVersion

Identifier	Description
session	The Session object that represents the current database-access session.
Return value	A String containing the Suite version.

## RATLC00719064; Perl SetActive method not working correctly with Boolean as documented

The Perl **SetActive** methods of the **Group** and **User** objects do not work correctly with a Boolean argument, as documented in the *API Reference*. The methods fail if you use True or False for the argument. The workaround is to use 1 instead of True and 0 instead of False.

## SaveQueryDef code example correction

The code example in the *API Reference* for the **SaveQueryDef** method of the **Workspace** object does not define the *RootFolder* variable before it is included as an argument in the call to **SaveQueryDef**. The following line should be added:

```
my $RootFolder;
```

The correct code example:

```
use CQPerlExt;
my $CQSession = CQSession::Build();
my $RootFolder = "Public Queries";
$CQSession->UserLogon($ologon, $opw, $odb, "");
$workspace = $CQSession->GetWorkSpace();
$queryDef = $CQSession->BuildQuery("Defect");
@owner = ("jswift");
@state = ("Closed");
@dbfields = ("ID","State","Headline");
foreach $field (@dbfields) {
    $queryDef->BuildField($field);
}
$filterNode1 = $queryDef->BuildFilterOperator($CQPerlExt::CQ_BOOL_OP_AND);
$filterNode1->BuildFilter("Owner", $CQPerlExt::CQ_COMP_OP_EQ, \@owner);
$filterNode1->BuildFilter('State', $CQPerlExt::CQ_COMP_OP_NOT_IN, \@state);
$resultSet = $CQSession->BuildResultSet($queryDef);
$resultSet->Execute();
$workspace->SaveQueryDef("delete me", $RootFolder, $queryDef, 1);
print "'$RootFolder/delete me' copied\n";
}
CQSession::Unbuild($CQSession);
```

## RATLC00715405; Document the Session.ClearNameValues method

The ClearQuest *API Reference* does not include documentation for the **ClearNameValues** method of the **Session** object. It should include the following information:

- **Description**

Clears all name values for the current session. A name value defines a session-level variable. Once it is set, it is accessible as long as the session is still alive. This method clears up all defined values for the current session. For more information on name values, see the "NameValue" method and the "Using Session Variables" sections in the ClearQuest *API Reference*.

- **Syntax**

**VBScript**

```
session.ClearNameValues
```

**Perl**

```
$session->ClearNameValues();
```

Identifier	Description
session	The Session object that represents the current database-access session.
Return value	None.

## SaveQueryDef method of the Workspace object issues

There are some issues with the **SaveQueryDef** method of the **Workspace** object.

### VBScript:

```
workspace.SaveQueryDef qdefName, qdefPath, queryDef, overwrite
```

### Perl:

```
$workspace->SaveQueryDef(qdefName, qdefPath, queryDef, overwrite);
```

- **RATLC00707958, APAR IC40118**

The **SaveQueryDef** method returns an error if the query already exists, with either a **0** or **1** value specified for the *overwrite* parameter.

The last parameter to the **SaveQueryDef** method is a Boolean value that specifies whether or not to overwrite an existing **QueryDef** object with the same name and path (0 = no overwrite, 1 = overwrite). Specifying the value = 1 should not return an error.

- **RATLC00708730**

The **SaveQueryDef** method does not copy a query to the **Public Queries** folder. It creates a query in a new (additional) **Personal Queries** folder (not in the **Public Queries** folder) when you specify a pathname (the *qdefPath* argument) of the folder in a **Public Queries** location.

- **RATLC00712832, APAR IC40152**

Using the **SaveQueryDef** method to save a query to a **Public Queries** folder returns inconsistent results. When using the **SaveQueryDef** method to save a **QueryDef** object, the query does not appear in all ClearQuest client interfaces, nor for all users.

## RATLC00711964, RAMBU00022729; GetFieldRequiredness return value for read\_only fields

**GetFieldRequiredness** method of the **Entity** object appears to return an incorrect value for **read\_only** fields. When using the **GetFieldRequiredness** method, it does not return a value of **3** to indicate that a field is **read\_only**. This is currently the correct behavior and the following note should be included in the *API Reference*.

**Note:** Because hooks operate with Administrator privileges (**SuperUser**), they can always modify the contents of a field, regardless of its current behavior setting. If the field is **read\_only** to a ClearQuest user but is modifiable in the context of a hook, then the return value is not **read\_only**.

## RATLC00715159, RATLC00059373; AddParamValue method allows the insertion of one string value

The **AddParamValue** method of the **ResultSet** object can be used to assign one or more values to a parameter. However, you must call this method for each individual value. For example, for a query with a dynamic filter on the **State** field with two states, you call the method two times:

```
$resultset->AddParamValue(1, "Submitted");  
$resultset->AddParamValue(2, "Resolved");
```

## **RATLC00703830, RATLC00667284, RAMBU00053964; New documentation on error checking and validation**

New documentation is available on error checking, validation, and the **SetFieldValue** method. See Error checking and validation for additional information to what is provided in the *API Reference*.

## **RATLC00371877; UnmarkEntityAsDuplicate method of the Session object note**

When a record is unmarked as a duplicate using the **UnmarkEntityAsDuplicate** method in a ClearQuest script, it does not remove the association in the Parent-Child **Entity** table. The **UnmarkEntityAsDuplicate** method removes the association information from the Child, but not from the Parent entity.

The documentation for the **UnmarkEntityAsDuplicate** method in the *API Reference* should include the following note:

**Note:** This method removes the duplicate information from the Child entity, but does not remove the duplicate information from the Parent entity.

## **RATLC00718478; ValidateQueryDefName method of the Workspace object**

The **ValidateQueryDefName** method of the **Workspace** object can be used to ensure that a given query (**QueryDef** object) name and path are valid in the workspace. However, the description in the *API Reference* describes the return value as **None**. This is only true if the name and path are valid.

The method throws an exception if the **QueryDef** name or path is not valid. It checks the name for invalid characters and ensures that the query itself does not already exist in the folder named by the path parameter.

If the **QueryDef** path is empty, there is not a complete or consistent validation.

## **RATLC00721299; GetFieldOriginalValue method should include note**

Documentation for **GetFieldOriginalValue** method in the *API Reference* should include the following note that the method does not work in an access control hook.

**Note:** Calling this method from an action access control hook returns the original value of the record's field regardless of whether or not the current action is a change-state action.

## **Upgrading user information from a schema repository to a user database**

When making changes to user information in a schema repository, in order to propagate the changes from the schema repository to the user databases you must upgrade the user databases with one of the following methods:

- **UpgradeInfo** method of the User object

Upgrades one user (that is, one User object) in all databases the user is subscribed to. It does not update group memberships, and only updates user properties such as is-active, e-mail, full name, and phone.

- **UpgradeMasterUserInfo** method of the **Database** object  
Upgrades all user and group information for one database, including the user and groups records and group memberships.

For existing users, you can propagate changes with either of the following methods:

- Get a list of user databases in the schema repository, iterate through each one calling the **UpgradeMasterUserInfo** method of the **Database** object (*CQDatabase->UpgradeMasterUserInfo*).
- Call the **UpgradeInfo** method of the **User** object (*CQUser->UpgradeInfo*).

For newly created users, if you create new ClearQuest users by using the **CreateUser** method of the **AdminSession** object (*CQAdminSession->CreateUser*) and set privileges, password, and other user information that you want propagated from a schema repository to user databases, you must use the **UpgradeMasterUserInfo** method of the **Database** object (by iterating through a list of user databases in the schema repository and calling *CQDatabase->UpgradeMasterUserInfo* for each user database).

The **UpgradeInfo** method was introduced (in version 2003.06.00) to provide a way to propagate user/group information from the schema repository to all affected user databases. However, the method only works for an existing user and not for a newly-created user.

**Note:** You cannot update Group membership using the **UpgradeInfo** method of the **User** object. Only the properties that can be set by the methods of the **User** object are updated by calling **UpgradeInfo**. Group membership is changed with the methods of the **Group** object, and the **UpgradeMasterUserInfo** method of the **Database** object must be used to update Group information settings.

## **RATLC00722670, APAR IC40986; RegisterSchemaRepoFromFile and GetLastSchemaRepoInfo documentation update**

The **RegisterSchemaRepoFromFile** and **GetLastSchemaRepoInfo** methods of the **AdminSession** object are described in the *API Reference* and should include the following updated information:

- **RegisterSchemaRepoFromFile**  
Creates a new database set (also known as a connection in ClearQuest Maintenance Tool) using the file path argument. Returns an error message if an error occurred. The file path argument is the path name of a connection profile file created by ClearQuest Maintenance Tool.

**Note:** This method does not currently work with versions of ClearQuest Maintenance Tool that support multiple database sets in a profile file.

- **GetLastSchemaRepoInfo**  
Returns schema repository information for the current connection.  
It can be useful to save the schema repository connection information in a file. This is called a schema repository location file (that is, a profile file). The name of this file is stored in the schema repository and whenever the schema repository location changes, the file is automatically updated. This method is used to save and retrieve information from the file.

For versions of ClearQuest that do not support multiple database sets, the **GetLastSchemaRepoInfo** method finds the most recently registered schema repository (from the current or prior releases) to be used during an upgrade to propagate database connections.

**Note:** This method does not currently work for versions of ClearQuest that support multiple database sets.

## Updates to "Ensuring that record data is current" section in *API Reference*

The "Ensuring that record data is current" section in the *API Reference* should be updated to reflect the newly documented Reload method of the **Entity** object. The correct text should be:

In a multiuser system, you can view the contents of a record without conflicting with other users. However, if another user is updating a record while you access a field of that record, you might get the field's old contents instead of the new contents. The **FieldInfo** object returned by the **GetFieldValue** method of the **Entity** object contains a snapshot of the field's data.

Calling **GetFieldValue** to get the field value again does not refresh the cached data, but only returns the previously cached value. You must call the Reload method of the **Entity** object to refresh the cached information to see any changes that another user might have made.

## RATLC00445073, RATLC00721111, APAR IC39464; Hook Performance issues and guidelines

New content on performance guidelines should be included in the *API Reference*. See Performance considerations for using hooks for this information. Also see InvalidateFieldChoiceList example for a code example for recalculating a choice list.

## GetValueAsList return value description is incorrect in *API Reference*

The return value description for the **GetValueAsList** method of the **FieldInfo** object is incorrect in the *API Reference*. The correct return value description should be:

For Visual Basic, a Variant array is returned. The Variant contains the list of values, separated by vbLF (for scalar fields, returns a 1-element Variant array). If the field contains no values, this method returns an Empty Variant.

## Document the Entity.Reload method

The ClearQuest *API Reference* does not include documentation for the **Reload** method of the **Entity** object. It should include the following information:

- **Description**

Refreshes the current in-memory copy of the record with the latest value from the database. For more information, see Updates to "Ensuring that record data is current" section in API Reference.

- **Syntax**

**VBScript**

`entity.Reload`

Perl

```
$entity->Reload();
```

Identifier	Description
entity	An Entity object representing a user data record. Inside a hook, if you omit this part of the syntax, the Entity object corresponding to the current data record is assumed (VBScript only).
Return value	None.

- **See Also**
  - Commit
  - IsEditable
  - Validate
  - EditEntity of the Session Object

## RATLC00717324, RATLC00707206; New methods for hiding records types

This release of ClearQuest includes a new feature that provides record hiding for record types that a user does not have authorization to submit.

Any record type that a user cannot submit (based on access control by either group list or hook) does not appear in the record list (**Choose Record Type...** dialog) when they select **Actions > New**. With this feature, users see a restricted list of record types instead of all record types in the schema. Without this feature, users see all record types but receive an error if they attempt to submit a record type for which they do not have authorization.

With this enhancement, two new methods of the **Entity** Object are available in the ClearQuest API:

- CanSubmit
- GetEntityDefNamesForSubmit

### CanSubmit

- **Description**

Returns True if the current user is allowed to submit the named record type. The result is based on any access control applied to the record type, such as a group list or a hook.

- **Syntax**

- **VBScript**

```
entity.CanSubmit entDefName
```

- **Perl**

```
$entity->CanSubmit($entDefName);
```

Identifier	Description
entity	An Entity object representing a user data record. Inside a hook, if you omit this part of the syntax, the Entity object corresponding to the current data record is assumed (VBScript only).
entDefName	A String that specifies the name of an EntityDef.
Return value	A String that contains the value or values stored in the field.

- **See Also**

- GetEntityDefNamesForSubmit

## GetEntityDefNamesForSubmit

- **Description**

Returns the list of all record types the user is allowed to submit. Like **CanSubmit**, the result is based on any access control applied to the record type, such as a group list or a hook. The result is similar to **GetEntityDefNames**, but always contains fewer elements, since **GetEntityDefNames** includes all record types (such as **User** and **Group**) which cannot be submitted. See the **GetEntityDefNames** method of the **Session** object in the *API Reference* for more information.

- **Syntax**

- **VBScript**

```
entity.GetEntityDefNamesForSubmit
```

- **Perl**

```
$entity->GetEntityDefNamesForSubmit();
```

Identifier	Description
<i>entity</i>	An Entity object representing a user data record. Inside a hook, if you omit this part of the syntax, the Entity object corresponding to the current data record is assumed (VBScript only).
Return value	For Visual Basic, returns a Variant containing an array of Strings containing the EntityDef names the user is allowed to submit. For Perl, returns a reference to an array of Strings containing the EntityDef names the user is allowed to submit.

- **See Also**

- GetEntityDefNames of the Session Object
- CanSubmit

## RATLC00696096; CtCmd code examples for UCM/ClearQuest integrations

The *API Reference* should include the following new sections that provide information on using **CtCmd** with the ClearQuest Perl API, for integrations between ClearCase and ClearQuest.

### Using CtCmd with ClearQuest Perl scripts for ClearCase integrations

In order to facilitate integrations between IBM Rational ClearQuest and ClearCase, or integrations with other products, it is useful to be able to obtain information about objects in ClearCase VOBs and PVOBs. From ClearQuest Perl scripts, you can use **CtCmd**, a Perl module that provides an interface to ClearCase. You can locate the CtCmd Perl module at: [http://cpan.org/modules/by-category/03\\_Development\\_Support/ClearCase/CtCmd-1.03.tar.gz](http://cpan.org/modules/by-category/03_Development_Support/ClearCase/CtCmd-1.03.tar.gz)

In order to use this Perl module in ClearCase triggers for ClearQuest integrations (such as checking in schema changes), and in ClearQuest Perl hook scripts, you must build **CtCmd** on Windows (following the instructions included with the **CtCmd** kit) and install it to a globally accessible UNC path. In Using CtCmd for UCM and ClearQuest, there is code to enable using the correct module path based on the platform (*OSNAME* in the following examples), which allows the hook scripts to pass validation at check in.

**Building CtCmd to work with cqperl on UNIX:** After following the instructions for unpacking the kit, check to see if ratlperl is found using your PATH environment variable. If not, add the path:

```
setenv PATH /opt/rational/common/bin:$PATH
```

Build from the directory where you unpacked the kit:

```
ratlperl Makefile.PL
make
make test
make install
```

Note that by using **ratlperl**, make install bases the install path off of the path to **ratlperl**. In this example, after running "make install", **CtCmd** is installed to /opt/rational/common/lib/perl5/site\_perl/5.6.1/sun4-solaris-multi/ClearCase. This is acceptable as long as /opt is an exported drive accessible by all UNIX ClearQuest clients (such as /net/qsun176/opt).

You can verify the installation by invoking the example Perl script below using either **ratlperl** or **cqperl**. For example:

```
cqperl script_name;
```

#### Using CtCmd for base ClearCase and ClearQuest:

```
use English;
unshift( @INC, "/net/qsun176/opt/rational/common/lib/perl5/site_perl/5.6.1/
sun4-solaris-multi/ClearCase" );

require ("CtCmd.pm");
my $ccinst = ClearCase::CtCmd->new();
my $result;
    # get parent stream and stream type of an activity
    my $status;
    my $stream;
    my $istream;
    my $str_type;
    my $project;
    ($status, $stream) = $ccinst->exec("des","-fmt","%[stream]p",
"activity:MCK00000031\@/var/tmp/beth_pvob");

    print( "Status: " . $status . "\n" );
    ($status, $project) = $ccinst->exec("des","-fmt","%[project]p",
"stream:$stream\@/var/tmp/beth_pvob");

    print( "Status: " . $status . "\n" );
    ($status, $istream) = $ccinst->exec("des","-fmt","%[istream]p",
"project:$project\@/var/tmp/beth_pvob");

    print( "Status: " . $status . "\n" );
    print( "Activity: MCK00000031\nStream: " . $stream . "\nProject: " .
$project . "\nIntegration Stream: " . $istream . "\n" );

    # find out if stream is integration stream
    if ( $stream !~ $istream ) {
        $result = "Current stream is not the integration stream";
    }
    else {
        $result = "Current stream is the integration stream";
    }
print($result);
```

#### Using CtCmd for UCM and ClearQuest:

```

# Start of Global Script UCU_CQActBeforeChact

sub UCU_CQActBeforeChact {
# the English Perl module is necessary for the use of the OSNAME variable
use English;
if ( $OSNAME =~ /Win/i ) {
    # UNC path to CtCmd.pm built for Windows
    unshift ( @INC, "///otterpop/c/Progra~1/Perl/site/lib/ClearCase" );
}
else {
    # NFS path to CtCmd.pm built for Solaris
    # if other platforms will be used, this block should also check for the
    # Unix platform and set the include path accordingly.
    unshift ( @INC, "/net/qsun176/usr1/rational/common/lib/perl5/site_perl/5.6.1/
sun4-solaris-multi/ClearCase" );
}

    require ("CtCmd.pm");
    my ($result);
    my ($param) = @_ ;
    # ---
    # This record hook is invoked by SQUID to invoke the "Perform ClearQuest
    # Action Before Changing Activity" policy. If the activity is not in
    # a single stream project, or is not in the project's integration
    # stream, the UCM change activity should fail.
    #
    # INPUT:
    # - Param must be a string with this format:
    #   "entity-type|entity-id|project_info|stream_info" (vertical bars are
    # delimiters)
    #   which represents the entity bound to the SUM_Project which was
    #   delivered, and whether the entity is valid or not
    # OUTPUT:
    # - If the entity is valid, this returns an empty string
    # - If the entity is not valid, this returns a string
    #   to be displayed as an error message.
    # ---
    # Parse the param string, but we will ignore project_info and stream_info
    # in this example
    my $entity_type;
    my $entity_id;
    my $project_info;
    my $stream_info;
    ($entity_type, $entity_id, $project_info, $stream_info) = split ('\\|', $param);
    # Create CtCmd instance
    my $inst_cc = ClearCase::CtCmd->new();
    # get parent stream and stream type
    my $status;
    my $stream;
    my $istream;
    my $str_type;
    my $project;
    # get the stream name from the entity_id
    ($status, $stream) = $inst_cc->exec("des","-fmt","[%stream]p",$entity_id);
    # get the project name from the stream
    ($status, $project) = $inst_cc->exec("des","-fmt","[%project]p",$stream);
    # get the name of the project's integration stream
    ($status, $istream) = $inst_cc->exec("des","-fmt","[%istream]p", $project);
    # get parent project type (note: model fmt is broken in MCK, returns blank
    string)
    my $proj_type;
    ($status, $proj_type) = $inst_cc->exec("des","-fmt","[%model]p",$project);
    # find out if project is single stream or parent stream is
    # integration stream
    # NOTE: if proj_type is SIMPLE, then stream should always be an
    # integration stream
    if ( ($proj_type !~ "SIMPLE") && ($stream !~ $istream) ) {
        $result = "Unable to change activity";
    }
}

```

```

    }
    else {
        $result = "";
    }
    return $result;
}
# End of Global Script UCU_CQActBeforeChact

```

**Windows platforms:** For integrations between ClearQuest and ClearCase, there is no solution for Windows at this time using **CtCmd**. Instead, use the ClearCase Automation Library (CAL). For more information and a code example, see "Using CAL methods in ClearQuest hook scripts" in the ClearQuest *Administrator's Guide*.

## VBScript Code Example Errors in API Reference

There are two VBScript code examples in the *API Reference* that are incorrect and should be updated with the following code.

- VBScript Code example error for **GetChildEntity** method of the **Link** object. There should be a **set** statement before `duplicateObj = duplicateLink.GetChildEntity`. The object (*duplicateObj*) also needs to be defined with the **Dim** statement. The correct example is:

```

originalID = GetDisplayName
If HasDuplicates Then
duplicateLinkList = GetDuplicates

' Output the IDs of the parent/child records
Dim duplicateObj
For Each duplicateLink In duplicateLinkList
    set duplicateObj = duplicateLink.GetChildEntity
    duplicateID = duplicateObj.GetDisplayName
    OutputDebugString "Parent ID:" & originalID & _
        " child ID:" & duplicateID
Next
End if

```

- VBScript Code example error for **GetEntityDefNames** method of the **Session** object. There should not be a **set** statement before `entityDefNames = sessionObj.GetEntityDefNames`. The correct example is:

```

set sessionObj = GetSession

' Get the list of names of all record types.
entityDefNames = sessionObj.GetEntityDefNames

' Iterate over all the record types
for each name in entityDefNames
    set entityDefObj = sessionObj.GetEntityDef(name)

' Do something with the EntityDef object
Next

```

## RATLC00715484; Version information for newer ClearQuest API methods

This section lists functions in the Perl and COM APIs that became available in ClearQuest version 2003.06.00 and in version 2002.05.00.

### New in version 2003.06.00

The following note should be added to the following methods in the API Reference:

**Note:** This method became available in version 2003.06.00.

- For both the COM and Perl APIs:

- AdminSession object:
  - IsStringInCQDataCodePage
  - CQDataCodePageIsSet
  - IsUnsupportedClientCodePage
  - IsClientCodePageCompatibleWithCQDataCodePage
  - GetCQDataCodePage
  - GetClientCodePage
  - ValidateStringInCQDataCodePage
- Entity object:
  - GetFieldStringValue
  - GetFieldStringValueAsList
  - GetFieldStringValues
  - SetFieldValues
- EntityDef object:
  - LookupFieldDefNameByDbName
  - LookupFieldDefDbNameByName
- Session object:
  - IsRestrictedUser
  - SetRestrictedUser
  - IsStringInCQDataCodePage
  - CQDataCodePageIsSet
  - IsUnsupportedClientCodePage
  - IsClientCodePageCompatibleWithCQDataCodePage
  - GetCQDataCodePage
  - GetClientCodePage
  - ValidateStringInCQDataCodePage
- ResultSet object:
  - GetAllColumnValues
  - UpgradeInfo
- For the COM API only:
  - Session object:
    - GetSuiteProductVersion

### **New in version 2002.05.00**

The following note should be added to the following methods in the API Reference:

**Note:** This method became available in version 2002.05.00.

- For both the COM and Perl APIs:
  - Session object:
    - IsReplicated
    - EntityExists
    - EntityExistsByDbId
    - IsMultisiteActivated
    - RegisterSchemaRepoFromFileByDbSet
    - IsPackageUpgradeNeeded

- StringIdToDbId
- DbIdToStringId
- Workspace object:
  - GetPublicFolderName
  - GetPersonalFolderName
  - GetWorkspaceItemDbIdList
  - SiteExtendedNameRequired
  - GetWorkspaceItemName
  - GetWorkspaceItemSiteExtended
  - GetWorkspaceItemPathName
  - GetWorkspaceItemType
  - RenameWorkspaceItemByDbId
  - DeleteWorkspaceItemByDbId
  - GetQueryDefByDbId
  - InsertNewQueryDef
  - UpdateQueryDef
  - GetChartDef
  - GetChartDefByDbId
  - InsertNewChartDef
  - UpdateChartDef
  - GetReportMgrByReportDbId
  - CreateWorkspaceFolder
  - GetQueryDbIdList
  - GetChartDbIdList
  - GetReportDbIdList
  - GetWorkspaceItemParentDbId
- AdminSession object:
  - IsReplicated
  - IsMultisiteActivated
  - GetLastSchemaRepoInfoByDbSet
  - RegisterSchemaRepoFromFileByDbSet
- For the COM API only:
  - Session object:
    - GetProductVersion
    - GetSuiteVersion
    - GetStageLabel
- For the Perl API only:
  - AdminSession object:
    - GetLocalReplicaName
    - GetReplicaNames
  - Attachments object:
    - Exists
  - ClearQuest object:
    - IsWindows
    - IsUnix

- Entity object:
  - AddAttachmentFieldValue
  - DeleteAttachmentFieldValue
  - EditAttachmentFieldDescription
  - LoadAttachment
  - GetAttachmentDisplayNameHeader
  - EditEntity
- Group object:
  - RemoveUser
  - GetMasterReplicaName
  - SetMasterReplicaByName
- ProdInfo object:
  - GetSuiteProductVersion
- QueryDef object:
  - IsFieldLegalForQuery
  - GetPrimaryEntityDefName
- Session object:
  - GetEntityDefOrFamily
  - IsWindows
  - IsUnix
  - GetProductInfo
- User object:
  - SetUserPrivilege
  - GetUserPrivilege
  - GetMasterReplicaName
  - SetMasterReplicaByName

---

## Chapter 6. MultiSite documentation updates

---

### Upgrading a schema version with ClearQuest MultiSite

This procedure describes how to introduce a new schema version to a ClearQuest MultiSite clan by synchronizing the new schema to all sites before upgrading any user databases. IBM Rational requires that you follow this procedure to help ensure a stable and reliable ClearQuest MultiSite environment. In addition to following this procedure, you must also not do the following when using ClearQuest MultiSite:

- Delete record types and states
- Change the working master if all databases are not using the same schema version
- Change mastership of package-owned queries

### Upgrade instructions

1. Make the desired schema changes and test them against a local test database.
2. Notify all users that maintenance is scheduled and they must disconnect from all user databases in the ClearQuest MultiSite clan.
3. Suspend automated synchronization between all user databases in the ClearQuest MultiSite clan.
4. (Optional) Stop and restart your vendor database server to ensure that there are no open connections to the schema repository or user databases.
5. Synchronize all sites in the ClearQuest MultiSite clan. After synchronization, check the incoming and outgoing storage bays to make sure that all packets were sent and imported. Run the **lsepoch** command at each site to verify that all replicas report the same epoch estimates.
6. Back up all schema repositories and user databases in the ClearQuest MultiSite clan.
7. Check in the new schema version at the master schema repository replica, but DO NOT upgrade the user database.
8. Export and send an update packet from the MASTR family only (not the user database family) to all other sites in the clan.

```
multiutil sync replica -export -clan DEMO -site SITEA -family MASTR  
-u admin -p "" -out c:\cqms\syncA.xml SITEB  
Multiutil: Packet file 'c:\cqms\syncA.xml' generated
```

9. Import the update packet at all sites.

```
multiutil sync replica -import -clan DEMO -site SITEB -family MASTR  
-u admin -p "" c:\cqms\syncA.xml  
Multiutil: 1 transactions from SITEA have been replayed into the  
MASTR database  
Multiutil: Deleting packet c:\cqms\syncA.xml
```

**Note:** At this point, the schema version exists at all the sites in the clan, but the user databases have not been upgraded.

10. Upgrade the user databases by performing the following steps. This ensures that all replicas in the family are running the same version of the schema before synchronization is restarted.
  - a. Upgrade the user database at the working master site.
  - b. Synchronize all sites.
  - c. Upgrade the user databases at all remaining sites.
11. Restart synchronization among the user databases at your sites.
12. Confirm that all synchronizations are successful and that all user databases in the clan are using the same schema version.
13. Notify users that the replicas are available.

---

## Synchronizing multiple user database families with `msimportauto.bat`

The *Administrator's Guide* for Rational ClearQuest MultiSite does not include information about the new `msimportauto.bat` script that you can use to synchronize replicas with multiple user database families. The following sections explain when to use the tool and provide syntax examples and instructions.

### Why should I use the `msimportauto.bat` script?

In certain circumstances, successful import of user database update packets may depend on information contained in other user database packets. If your schema repository is associated with multiple user database families, import may fail if the packets are not replayed in the order they were generated.

#### Example

A particular clan, with sites in Boston and Denver has two user databases, User1 and User2. The Boston administrator generates a synchronization packet for User1 (Packet1) and then generates one for User2 (Packet2). While the packets are being created, an administrator modifies user account information; this causes schema repository oplog content to be included in both of the user database packets.

Some time later, the Boston administrator generates another pair of user database synchronization packets for User1 (Packet3) and User2 (Packet4). Again, an administrator modifies user account information while the packets are being created, and schema repository oplog content is included in both user database packets.

All four packets are sent to the Denver site. At the Denver site, the administrator runs `sync replica -import` and specifies the User1 database family. Packet1 and Packet3 are both intended for the User1 family. Import of Packet1 is successful and replays oplogs in User1 and the schema repository. However, import of Packet3 fails, because it depends on schema repository database oplogs, contained in Packet2, which have not yet been replayed at the Denver replica.

#### Solution

To avoid this situation, packets created at the exporting site must be replayed in the same sequence at the importing sites. IBM recommends that you use the `msimportauto.bat` script, which is included with this version of ClearQuest. This script scans the import directory for update packets and then attempts to import the packets to each family. If any packets are successfully imported, the imported packets are deleted from the directory and the script attempts to import the next packet. The script stops executing when all packets are replayed and the directory is empty. If a series of import attempts results in no packets being deleted from the directory, the script stops executing and import fails.

## Running msimportauto.bat

Use the msimportauto.bat script to import update packets in the correct order when a clan contains multiple user databases. The script can also be used to perform **syncreplica -export**.

### Syntax

**msimportauto** [ **-debug** *level* ][ **-MaxLoops** *num-loops* [ **-TimeToWait** *seconds* ]]

[ **-AndDoExport** ]{ **-clan** *clan-name* *clan-info* }

### Operating modes

This program operates in one of the following modes:

- **Synchronize now.** The program receives pending updates, sends pending updates (optionally, with **-AndDoExport**), and exits. Use this mode if you want to synchronize immediately or if you want to schedule program execution with an external scheduler package, such as the Windows Scheduled Tasks facility or the ClearCase scheduler.
- **Loop and wait.** The program receives pending updates, sends pending updates (optionally, with **-AndDoExport**), sleeps a specified number of seconds; it then loops back and receives, sends, and sleeps again. Use this mode if you want the program to, in effect, act as its own scheduler.

### Options and arguments

**-debug** *level*

Set the debug level:

0	Apply packets to database; don't produce any debugging output
1..9	Show diagnostic information and apply packets to database (higher numbers show more granular output)
10+	Show diagnostic information, don't apply packets to database

**-MaxLoops** *num-loops*

Specifies the number of times the script will perform a receive, send, and sleep cycle (one iteration) in loop-and-wait mode.

**-TimeToWait** *seconds*

Specifies the amount of time, in seconds, between iterations. If **-MaxLoops** is specified, but **-TimeToWait** is not, the default is 30 seconds between iterations.

**-AndDoExport**

Issue **syncreplica -export** commands for the input databases (includes export as part of the receive, send, and sleep cycle).

**-clan** *clan-name*

Specifies the clans to synchronize. Multiple clans may be specified in one command, but the **-clan** switch must be repeated.

*clan-info*

Specify *clan-info* in the following format (no spaces):

```
admin_username,admin_password;storage_class |
directory;family_1,my_site,other_site_1[,other_site_2,...[,other_site_n]
[,family_2,my_site,other_site_1...]...[,family_n,my_site,other_site_1
[,other_site_2,...[,other_site_n]]
```

*my\_site* is the local site that will be imported into and exported from.  
*other\_site\_#* specifies the other sites in the clan to be exported to and imported from.

## Examples

The following commands must be entered on one line.

- In this example, two clans, TEST and TEST1 are synchronized. TEST contains two user database families (te and te2) and TEST1 contains one (d2). Both clans use directories to store packets.

```
msimportauto -debug 1 -clan TEST
admin,"";C:\testdir\test;te,siteb,sitea;te2,siteb,sitea-clan TEST1
admin,"";c:\testdir\test;d2,sitea,siteb
```

- In this example, three clans (TESTCLAN, TESTCLAN2, and TESTCLAN3) are synchronized. Clan TESTCLAN consists of two user database families, te and te2. Clans TESTCLAN and TESTCLAN3 use the MultiSite synchronization server, while TESTCLAN2 uses the directory c:\TESTCLAN2 to store packets.

```
msimportauto -debug 0 -MaxLoops 2 -TimeToWait 30 -clan
TESTCLANadmin,""; cq_default;te,SITEA,SITEB,SITEC;te2,SITEA,SITEB
-clan TESTCLAN2 admin,"";c:\TESTCLAN2;d2,SITEA,SITEB
-clan TESTCLAN3 admin,"";cq_default;dt3,SITEA,SITEB-AndDoExport
```

---

## repair

Displays or deletes entries from the ratl\_uuid table of a replica

## Applicability

**Product**

Command type

**MultiSite**

multiutil subcommand

**Platform**

UNIX

Windows

## Synopsis

```
repair -orphaned_ratl_uuids [ -delete ] -cl/an clan-name -site site-name
-fam/ily family-name -u/ser username [ -p/assword ] password
```

## Description

If the ratl\_uuid table of a replica contains entries that are not also included in the master\_uuid table, a mkreplica command may fail in one of the following ways:

- The mkreplica -export operation succeeds, but the import operation fails
- The mkreplica -export operation fails with the following error:

```
There are num-entries entries in the ratl_uuids table that have no
corresponding rows in the master_uuids table. To remove these
'orphaned' rows from the ratl_uuids table, please backup the master
and user databases, then execute 'multiutil repair -orphaned_ratl_uuids
-delete ...', specifying the same clan, site, family, user and
```

password information.  
Multiutil: The mkreplica -export command failed.

You can use the repair command to view or delete the "orphaned" entries in the ratl\_uuid table. After you delete the entries from the ratl\_uuid table, mkreplica -export and -import operations will no longer fail.

### Locking the Replica

The repair command locks the specified database replica. Locking it ensures that while the repair command is running, no other changes are made to the replica. The database replica is unlocked after the repair command is completed.

## Restrictions

*Locks:* This command fails if the database is locked (for example, during the upgrade process) or while another ClearQuest MultiSite operation is being performed.

## Options and arguments

### Specifying the operation

#### Default

Displays all entries in the ratl\_uuids table that have no corresponding rows in the master\_uuids table.

#### -delete

Deletes all entries in the ratl\_uuids table that have no corresponding rows in the master\_uuids table.

### Specifying the clan, site, and family

#### Default

Clan: First clan replicated at this site. If there is more than one clan at the site, **-clan** is required.

Site: Current site. If there is more than one site on this host, **-site** is required.

Family: No default; you must specify a family.

#### **-cl/an** *clan-name*

Name of the replica's clan.

#### **-site** *site-name*

Name of the replica's site.

#### **-fam/ily** *family-name*

User database family: Database name given to the user database when it was created.

Schema repository family: The family name is **MASTR**.

### Specifying a user name and password

#### Default

You must specify a user name and password.

#### **-u/ser** *user*

Name of a user with Super User privileges.

**-p**/*password* *password*  
Password associated with the specified user.

## Examples

In these examples, the lines are broken for readability. You must enter each command on a single physical line.

- At the **boston\_hub** replica, display a list of all entries in the `ratl_uuids` table that have no corresponding rows in the `master_uuids` table.

```
multiutil repair -orphaned_ratl_uuids -clan telecomm -site boston_hub  
-family DEV -user susan -p passwd
```

- Delete all entries in the `ratl_uuids` table of the **boston\_hub** replica that have no corresponding rows in the `master_uuids` table.

```
multiutil repair -orphaned_ratl_uuids -delete -clan telecomm  
-site boston_hub -family DEV -user susan -p passwd
```

## See also

`mkreplica`

---

## Appendix. Notices

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

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

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

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

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

**The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:**

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

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

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

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

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created

programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation  
Department BCFB  
20 Maguire Road  
Lexington, MA 02421  
U.S.A.

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

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

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

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

#### COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

(c) (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. (c) Copyright IBM Corp. \_enter the year or years\_. All rights reserved.

Additional legal notices are described in the legal\_information.html file that is included in your Rational software installation.

Trademarks

IBM, Rational, DB2 , ClearCase, ClearCase MultiSite, ClearQuest, and RequisitePro are trademarks of International Business Machines Corporation in the United States, other countries, or both.

Java and all Java-based trademarks and logos are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

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

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

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

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



---

## Readers' Comments — We'd Like to Hear from You

ClearQuest  
Documentation Supplement  
Windows Version 2003.06.13, UNIX patch 2003.06.00-6

Publication No. GI11-5979-00

Overall, how satisfied are you with the information in this book?

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Overall satisfaction	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

How satisfied are you that the information in this book is:

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Accurate	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Complete	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Easy to find	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Easy to understand	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Well organized	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Applicable to your tasks	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Please tell us how we can improve this book:

Thank you for your responses. May we contact you? ☐ Yes ☐ No

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

---

Name

---

Address

---

Company or Organization

---

Phone No.



Cut or Fold  
Along Line

Fold and Tape

Please do not staple

Fold and Tape



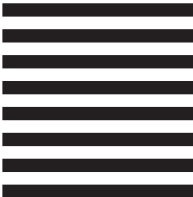
NO POSTAGE  
NECESSARY  
IF MAILED IN THE  
UNITED STATES

**BUSINESS REPLY MAIL**

FIRST-CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

IBM Corporation  
Attn: Dept CZLA  
20 Maguire Road  
Lexington, MA  
02421-3112



Fold and Tape

Please do not staple

Fold and Tape

Cut or Fold  
Along Line





Printed in USA

GI11-5979-00

