ClearQuest API Reference

support@rational.com
http://www.rational.com

IMPORTANT NOTICE

DISCLAIMER OF WARRANTY

Rational Software Corporation makes no representations or warranties, either express or implied, by or with respect to anything in this guide, and shall not be liable for any implied warranties of merchantability or fitness for a particular purpose or for any indirect, special or consequential damages.

COPYRIGHT NOTICE

RESTRICTED RIGHTS LEGEND

TRADEMARKS

PATENTS

Document Version:

Printed in the U.S.A.

Contents

Using the ClearQuest API

Understanding the ClearQuest API

You can use this API to write code that runs within ClearQuest (hook code), or that runs independently of an instance of the ClearQuest application.

Type of code	Example
Hook scripts for your ClearQuest schema	Modify records that users submit, and validate the records before they are committed to the user database. (ClearQuest Designer provides an editor for you to insert hook scripts.)
External applications that run outside of ClearQuest	View or modify the data ClearQuest stores in the user database and schema repository.

Choosing a Scripting Language

The ClearQuest API is implemented as a COM library for VBScript/Visual Basic, and as a Perl package, CQPerlExt package. You can write hook scripts in VBScript or Perl. ClearQuest runs your hooks in VBScript or Perl, but not both at the same time. ClearQuest Designer allows you to switch between scripting languages. See “Choosing a scripting language” in the chapter, “Using hooks to customize your workflow” in *Administering ClearQuest.*

Note: You can write external applications in any programming environment that supports OLE automation (such as Visual Basic or Visual C++), or that can embed Perl.

Using Perl

Perl, the Practical Extraction and Reporting Language, offers a platform-independent solution for ClearQuest scripting. Hooks scripts you write in Perl support both the ClearQuest clients running under Windows and UNIX.

Note: If you are using the ClearQuest Web client, you must use VBScript. ClearQuest Web does not support the use of Perl scripts.

ClearQuest API support for VBScript is different than that for Perl. When you use Perl, be aware that:

- the prefix and syntax are different. See *Notation Conventions for Perl* on page 2.

- you must use the prefix for Entity methods and properties inside hook scripts, unlike VBScript, where the entity object is implicit.
- Perl uses an array for hook choices instead of a HookChoices object
- the eventObject is supported differently. See the section on the **EventObject Object**.

Using Perl Modules

In addition to the CQPerlExt package, ClearQuest ships with most of the Perl5 modules listed at http://www.cpan.org/modules, including the Win32 modules that enable your Perl scripts to interface with Windows systems and applications.

Note: Rational Software has no relation to this site.

Using Perl for external applications

External applications must be written using the ClearQuest Perl engine, CQPerl. Also, you cannot call Perl hooks from an external application written in CQPerl. If you use Perl for an external application, we recommend that you limit the external application to tasks that are independent of actions, such as querying, reporting, and user administration.

Notation Conventions for Perl

The table below outlines the Perl notational conventions of this document.

Prefix	Description
CQ	Prefix for objects that the ClearQuest API can access through its CQPerlExt package. For example: CQEntity
$CQPerlExt::CQ	Prefix for Perl **Enumerated Constants**. For example, $CQPerlExt::CQ_ORACLE Note: CQPerlExt treats constants as read-only variables.

This document shows the syntax of Perl using the “get” and/or “set” prefix for calls to a property. All the Perl “get” calls to a property return a value.

Using VBScript

VBScript, a subset of Microsoft Visual Basic, can be used on both the ClearQuest Windows and ClearQuest Web clients.

Note: If you are using the ClearQuest Web client, you must use VBScript. ClearQuest Web does not support the use of Perl scripts.

Notation Conventions for VBScript

The table below outlines VBScript notational conventions used in this document.

Prefix	Description
OAd	Prefix for objects that the ClearQuest API can access through its COM library. For example: OAdEntity **Note:** The Session and AdminSession objects do not use the OAd prefix. (See **Syntax for manually creating the Session object (or the AdminSession object) in an external application**)
AD	Prefix for VBScript **Enumerated Constants**. For example: AD_ORACLE

Handling Script Errors

The following sections describe error handling for VBScript and Perl.

VBScript Error Handling

Use the standard means of handling VBScript errors by using the VBScript **On Error** statement. You can then examine the **Err** error object and analyze errors at any time.

Perl Script Error Handling

Use the standard means of handling Perl errors by using the Perl **eval** statement to analyze errors. Use the following syntax:

eval *{enter statements you want to monitor}*;

At run time, if the Perl engine encounters an error in a statement in the eval block, it skips the rest of the eval block and sets **$@** to the corresponding error text.

Understanding ClearQuest API Objects

ClearQuest uses the Session object to verify the user's authority to access a given database. When a user launches the ClearQuest client application, ClearQuest automatically authenticates the user using the logon dialog box. However, developers of stand-alone applications must use the methods of the Session object to log on to the desired database.

The Session object acts as the primary root object to the remaining database objects. (To learn about the other root object, the AdminSession object, see *Accessing the Schema Repository* on page 25.) You use the Session object to:

- create or access many of the other objects in the system
- create new records or modify existing records
- create the query objects that enable you to search the database for a particular record (or set of records)

After you have started a session, the object you will work with the most is the Entity object. The Entity object represents a single data record and enables you to view or change the data in a record.

Using the methods of Entity, you can do the following.

- Acquire information about the fields of the underlying record, and about any related objects in the system (including duplicate records, attached files, and activity logs for the record).
- Acquire the metadata associated with the Entity object to determine the structure of the record.

Overview Diagram of API Objects

The following diagram shows the types of objects you use to access a user database and the relationships between them. The arrows indicate the direction in which you acquire related objects. For example, from the Session object, you can acquire five different types of objects directly: DatabaseDesc, Entity, EntityDef, QueryDef, and ResultSet.

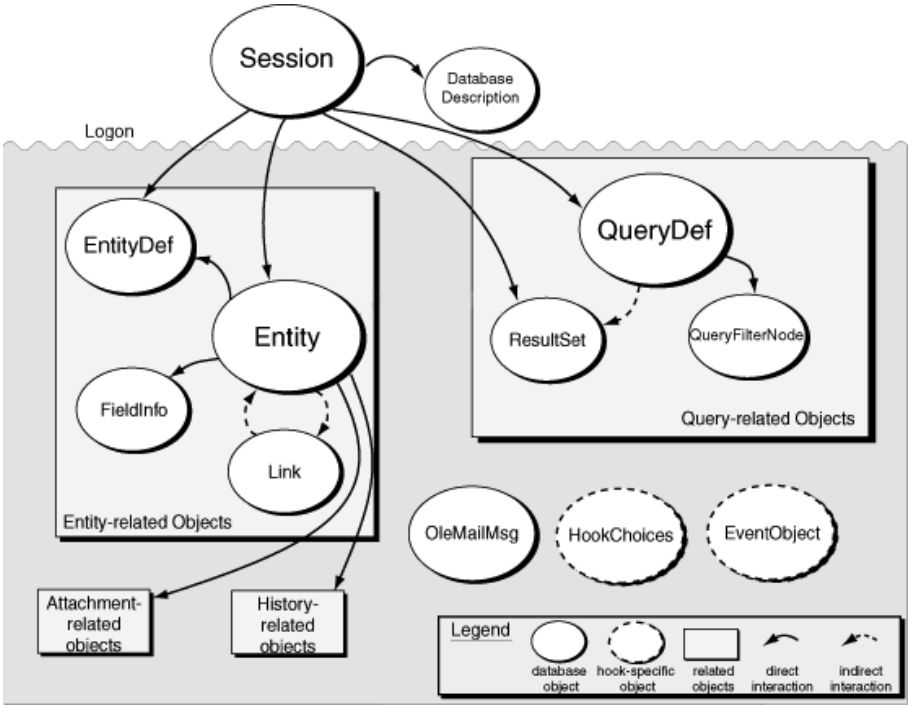

Figure 1: Using Objects to Access a User Database

In some cases, objects have an indirect relationship. For example, the QueryDef and ResultSet objects work together to run a query, but you create these objects separately using methods of the Session object. The ResultSet object uses information from the QueryDef object to perform the query.

The ClearQuest schema repository is the “master” database that contains your schemas. In addition to the objects in the preceding diagram, ClearQuest also defines a set of objects for accessing the schema repository (see *Accessing the Schema Repository* on page 25), collection objects (see *Schema Repository Collection Objects* on page 8), and additional objects (see *Additional Objects* on page 11.

Overview Table of the API Objects

The tables below give a quick overview of the API in the order in which they appear. The API include user database objects, schema repository objects, schema repository collection objects, and additional objects.

User Database Objects

User database objects are the objects your code works with the most.

User Database Object	Description
Session Object	Access the user database; build a new record
Entity Object	Work with Record data: set field values, validate, commit, revert
EntityDef Object	View read-only meta-data: actions, fields, hooks, states, and transitions applicable to a given record type.
QueryDef Object	Defines the query criteria
ResultSet Object	Contains the data the query fetches
QueryFilterNode Object	Implements comparison filters for the query

Schema Repository Objects

Schema repository (master database) objects allow you to get and set certain kinds of metadata.

Schema Repository Object	Description
AdminSession Object	You use the **AdminSession Object** to access the schema repository. (This is analogous to using the **Session Object** to access a user database.)
Database Object	The database for user data, such as defects.

Schema Repository Object	Description
Schema Object	Each schema in the schema repository is represented by a **Schema Object**. You cannot modify schemas programmatically. Use the ClearQuest Designer to make changes to a schema. The Schema object provides you with a list of schema revisions that you can use to upgrade a database.
SchemaRev Object	Each schema revision in the schema repository is represented by a **SchemaRev Object**. You cannot modify the SchemaRev object programmatically. Use the ClearQuest Designer to make changes to a schema.
Group Object	Each user group in the schema repository is represented by a **Group Object**. This object contains the basic group information, including the users belonging to the group and the databases to which the group is subscribed.
User Object	Each user account in the schema repository is represented by a **User Object**. This object contains the user's profile information, including the groups and databases to which the user is subscribed.

Schema Repository Collection Objects

The schema repository (master database) collection objects provide a convenient means to work with multiple instances of certain schema repository objects, instead of having to work with each one individually:

Schema Repository Collection Objects	Description
Databases Object	Collection of user databases
EntityDefs Object	Collection of EntityDef (record type) objects
Groups Object	Collection of user database groups
Schemas Object	Collection of schemas in the schema repository
SchemaRevs Object	Collection of schema revisions objects in the schema repository
Users Object	Collection of user database users

Attachment Objects

In ClearQuest the user can attach files to a defect (bug report) in an attachment field. A record representing a defect can have multiple attachment fields, and each field can have multiple attached files. For example, a record might have three separate attachment fields: one for source code files, one for engineering specifications, and one for documentation.

To support this functionality, the API provides four objects:

Attachment Objects	Description
AttachmentField Object	Represents a single attachment field in a record
AttachmentFields Object	Represents the attachment fields in a record
Attachment Object	Stores an attachment file and information about it
Attachments Object	Represents a set of attachments in one attachment field of a record

The AttachmentFields object is the container object for all of the other objects. It represents all of the attachment fields associated with a record. There can be only one AttachmentFields object associated with a record. This object contains one or more AttachmentField objects.

The AttachmentField object represents a single attachment field in a record. A record can have multiple AttachmentField objects, each of which includes a single Attachments object.

The Attachments object is a container object that stores one or more Attachment objects. An Attachments object is always associated with a single AttachmentField object.

An Attachment object contains a single attached file.

For more information about each object, click on the links above.

Information Objects

The Information Objects are APIs for retrieving information from both the schema repository and the user database.

Information Objects	Description
DatabaseDesc Object	Provides information about a given database, including whether it is a schema repository or a user database
EventObject Object	Provides read-only information about a record's named hook
FieldInfo Object	Provides read-only information about a field in a user database record (for example, what value it currently stores), but you typically use the Entity object instead

History Objects

In ClearQuest a defect (bug report) has history information associated with it. Each record has a history field, and this field can have multiple history entries. Each history entry is a line of text describing the modification. All history objects are read-only, because the history entries for a data record are created automatically by ClearQuest.

To support this functionality, the API provides four objects:

History Objects	Description
Histories Object	Contains History objects
History Object	Provides a string that describes the modifications a record has undergone
HistoryField Object	Represents a single history field in a record
HistoryFields Object	Contains all history-related objects

The HistoryFields object is the container object for all of the other objects. It represents all of the history fields associated with a record. There can be only one HistoryFields object associated with a record. This object contains one or more HistoryField objects.

The HistoryField object represents a single history field in a record. A record can have multiple HistoryField objects, each of which includes a single Histories object.

The Histories object is a container object that stores one or more History objects. A Histories object is always associated with a single HistoryField object.

A History object contains a string that describes the modifications to the record.

For details about each object, click on the links above.

See Also:

HistoryFields Property of the **Entity Object**

Additional Objects

Additional objects provide APIs to list choices, links between records, email notification, creating charts and reports and workspace for manipulating saved queries, charts, and reports.

Additional Objects	Description
HookChoices object	Lists choices in a CHOICE-LIST hook
Link Object	Connects an original record (parent) with the duplicate (child) record.
OleMailMsg Object	Supports an email notification hook
CHARTMGR Object	Provides an interface for creating charts
ReportMgr Object	Provides an interface for generating reports
WORKSPACE Object	Provides an interface for manipulating saved queries, reports, and charts

Working with Sessions

Users access a ClearQuest database through a Session object. This object provides methods for logging on to the database, viewing records (entities), and creating queries. You can also use the **Session Object** to store variables for the session.

Getting a Session Object

The Session object is the entry point for accessing ClearQuest databases. If you are writing an external application, you must create a Session object and use it to log on to a database. Once you have logged on to a database, you can use the Session object to

- create new records or queries
- edit existing records
- view information about the database

For script hooks (VBScript and Perl), ClearQuest creates a Session object for your hooks automatically when the user logs on to the database. The session object is available through the entity object. In the context of a hook, to get a session object from an entity object, use the following syntax:

Scripting Language	Syntax for making a call to an Entity object in a hook
VBScript	set currentSession = GetSession VBScript hooks implicitly associate the Entity object with the current record.
Perl	When writing ClearQuest hooks, a session object is created and made available through the context variable $session. You do not need to perform any explicit call to create it. If you need a session object in some other context (such as when writing a standalone program) you can get a session object by using the following syntax: $session=$entity->GetSession();

For external applications, you must create a Session object manually. If you want to use the adminSession object, the same rule applies

Language Example	Syntax for manually creating the Session object (or the AdminSession object) in an external application
Visual Basic	set currentSession = CreateObject("CLEARQUEST.SESSION") set adminSession = CreateObject("CLEARQUEST.ADMINSESSION")
Perl	$CQSession = CQPerlExt::CQSession_Build(); $AdminSession= CQPerlExt::CQAdminSession_Build(); When you are done with the object, destroy it: CQSession::Unbuild($currentSession); CQAdminSession::Unbuild($currentAdminSession);

Logging on to a Database

To protect your databases from unauthorized users, ClearQuest requires that you log on to a database before accessing its records. For hooks, this user authentication is handled automatically by the ClearQuest client application. However, external applications must log on programmatically by using the Session object.

To determine which database to log on to, and to perform the log on, follow these steps:

1. Get a list of the databases associated with a schema repository by calling the **GetAccessibleDatabases Method** of the Session object.

   This method returns a collection of DatabaseDesc objects, each of which contains information about a single user database.

2. Get the name of the database and enter an empty string ("") for the database set (the set of databases to which a database belongs) by using the methods of the **DatabaseDesc Object**.

3. Log on to the database by calling the **UserLogon Method** of the Session object.

You must have a valid login ID and password to log on to the database. As soon as you log on, you can start looking through records and creating queries. (See the description of the **UserLogon Method** for usage information.)

Note: If your external application uses Session methods, the general rule is to call UserLogon before calling other Session methods. However, there are two Session methods that you can call before calling UserLogon: **GetAccessibleDatabases Method**, and **OutputDebugString Method**.

Using Session-Wide Variables

ClearQuest supports the use of session-wide variables for storing information. After you create session-wide variables, you can access them through the current Session object using functions or subroutines, including hooks, that have access to the Session object. When the current session ends, all of the variables associated with that Session object are deleted. The session ends when the user logs out or the final reference to the Session object ceases to exist.

To access session-wide variables, use the **NameValue Property** of the Session object.

To create a new variable, pass a new name and value to the **NameValue Property**. If the name is unique, the Session object creates a new entry for the variable and assigns to the variable the value you provide. If the name is not unique, the Session object replaces the previous value with the new value you provide.

To check whether a variable exists, use the **HasValue Method** of the Session object.

The following example shows how to create a new variable and return its value. This example creates the named variable "Hello" and assigns the value "Hello World" to it.

Example (in VBScript)

```
Dim myValue
curSession = GetSession()

myValue = "Hello World"

' Create and set the value of the "Hello" variable
curSession.NameValue("Hello") = myValue

' Get the current value
```

```
Dim newValue
newValue = curSession.NameValue("Hello")
```

Example (in Perl)

```
# You can use $session instead of defining
# $curSession = $entity->GetSession();

myValue = "Hello World";

# Create and set the value of the "Hello" variable
$session->SetNameValue("Hello", $myValue);

# Get the current value
$newValue = session->GetNameValue("Hello");

# Optional
$session->OutputDebugString($newValue);
```

Ending a Session (for External Applications)

Because hooks execute at predefined times during the middle of a session, when you write a hook, your hook code does not end a session. The session ends automatically when the user logs off.

However, when you write an external application, you must end the current session by deleting the Session object that you have created. There is no explicit method for logging off the database.

Your external application should end a session properly:

- Delete any objects that you explicitly created and do not need any more, including a Session object.

Note: With VBScript, the session ends when the user logs out or the final reference to the Session object ceases to exist.

Working with Multiple Sessions

Because each Session object is associated with a particular user, you can create multiple Session objects for different users. Each Session object you create can access only the information available to the associated user.

You cannot use one Session object to operate on the objects returned by another Session object. All of the objects you create with a Session object are bound to that Session object and cannot be used by other sessions. For example, if you have two sessions, A and B, and you use session B to get an Entity object, session A cannot access that Entity object.

Working with Queries

A query specifies criteria for fetching data from the database. You can search for data in a ClearQuest database by using queries in hooks. You can create and run a query to fetch data from the ClearQuest database according to the search criteria that you provide in the query. The process of working with queries consists of four major steps.

1. Build a query (QueryDef) to specify what data you want.
2. Create a result set object to hold the data.
3. Execute the query, which populates a result set with the data it fetches from the database.
4. Move through the result set.

   Note: If you write a hook that operates only on the current Entity object, you do not need to use a query.

Creating Queries

Creating a query involves the creation of at least three separate objects: a **QueryDef Object**, a **QueryFilterNode Object**, and a **ResultSet Object**. More complex queries might also involve the creation of additional QueryFilterNode objects.

To create a query, follow these steps:

1. Create a QueryDef object and fill it with the search parameters.

   To create this object, you can use either the **BuildQuery Method** or the **BuildSQLQuery Method** of the Session object.

   Note: We recommend that use the **BuildQuery Method**. The **BuildSQLQuery Method** generates a ResultSet object directly from an SQL query string.

2. Use the methods of QueryDef to add search criteria and to specify the fields of each record you want the query to return.
3. Create a ResultSet object to hold the returned data.

To create this object, call the **BuildResultSet Method** of the Session object. On creation, the ResultSet object creates a set of internal data structures using the information in the QueryDef object as a template. When the query is run, the ResultSet object fills these data structures with data from the query.

4 Run the query by calling the ResultSet object's **Execute Method**.

5 Access the data using other methods of this object. (See **Navigating through the Result Set**.)

Note: If you use the BuildSQLQuery method to create a query based on SQL syntax, your query string must contain all of the desired search parameters. The BuildSQLQuery method returns a ResultSet object directly, instead of returning a QueryDef object.

Defining Your Search Criteria

You define a query's search criteria. As the query runs, ClearQuest compares your criteria to the fields of each record in the database. Each time a record in the database matches your criteria, ClearQuest returns the record in the ResultSet object.

For examples of building a query with the API, see Building Queries for Defects and Users in the chapter entitled Examples of Hooks and Scripts.

Using Query Filters

Each comparison is implemented by a filter, which is an instance of the **QueryFilterNode Object**. A filter allows you to compare a field to a single value or to a range of values. The operator you choose for the filter determines the type of comparison to perform. For a list of valid operators, see the **CompOp Constants** enumerated type.

To create a hierarchical tree of filters, join them together with a Boolean operator and nest some filters within other filters. Each filter consists of either a single condition or a group of conditions joined together with an AND or an OR operator. As you build your filters, you can nest more complex groups of filters to create a complex set of search logic.

Running Queries

Rather than returning the entire record, ClearQuest returns only those fields of the record that you specified by calling the **BuildField Method** of the QueryDef object (see *Creating Queries* on page 17). The Execute method returns results in no particular order. Therefore, the ResultSet object uses a cursor-based system to allow your code to move through the records one by one.

To perform the search (execute the query), call the **Execute Method** of the ResultSet object.You can now use the methods of ResultSet to obtain information about the fields of the record.

Working with a Result Set

Here are the steps to follow when using a ResultSet object:

1. Create the ResultSet object.
2. Run the query to fill the ResultSet with data.
3. Navigate (move) through the resulting data until you find the record you want.
4. Retrieve the values from the fields of the record.

Creating a Result Set

To create a ResultSet object, you use either the **BuildResultSet Method** or the **BuildSQLQuery Method** of the **Session Object**. Both of these methods return a ResultSet object that is ready to run the query but which contains no data.

Running the Query

To run the query, you call the **Execute Method** of the ResultSet object. This method fills the ResultSet with data from the database. The result set might be larger than is optimal for the memory management of certain computers. Therefore, as you navigate through the result set, ClearQuest transparently loads only the data you need. As you request new data, ClearQuest transparently fetches them.

Navigating through the Result Set

To move to the first record in the result set, call the **MoveNext Method**, which initializes the cursor and moves it to the first record. You can now use the methods of ResultSet to obtain information about the fields of first record.

To move to subsequent records, use the MoveNext method again.You can now use the methods of ResultSet to obtain information about the fields of the current record.

Note: If you plan to view or modify a record, your query must ask ClearQuest to return the ID field of the record. With this ID, you can then use the **GetEntity Method** of the Session object to obtain the corresponding Entity object. See *Working With Records* on page 21.

Retrieving Values from Fields

When you have the cursor at the row you want, use the **GetColumnValue Method** to fetch the value for a field of that record.

If you created your query using ...	The order of the columns corresponds to ...
the QueryDef object	the order in which you added fields using the **BuildField Method**.
a SQL statement	the SQL statement (To discover which column has the data you want, use the ResultSet object: **GetNumberOfColumns Method**, the **GetColumnType Method**, and the **GetColumnLabel Method**.)

Working With Records

Databases use records to organize and store information. In ClearQuest, the term record (entity) refers to a structure that organizes the information available for a single instance of a record type (entity), such as “defect”. ClearQuest records can contain data from multiple database tables.

ClearQuest uses instances of the Entity class to organize and manage record data. Each instance of the Entity class provides access to the values in the fields of the record, a list of the duplicates of the record, the history of the record, and any files attached to the record.

Note: To use the methods of the Session object, you must already know the definition of the record. You can use methods of the **Session Object** to have a query find records that match criteria you define, and then work with the records in the query’s result set. To learn how to use the API for queries, see **Working with Queries**.

Getting Entity Objects

To create an Entity object, you use the Session object’s **BuildEntity Method**. Calling this method creates a new Entity object and initiates a Submit action, making it possible to edit the default values in the Entity object.

To obtain an existing Entity object whose ID you know, you can use the Session object’s **GetEntity Method** or **GetEntityByDbId Method**. If you do not know the ID of the record, you can use the Session object's **BuildQuery Method** to create a query and search for records that match a desired set of criteria. Entity objects found using these techniques are read-only. To edit an Entity object, you must call the Session object's **EditEntity Method**.

After you acquire an Entity object, you can call its methods to perform tasks such as the following.

Task	Entity object method to call
Examine or modify the values of a field	**GetFieldValue Method**, **SetFieldValue Method**
Validate and commit the record	**Validate Method**, **Commit Method**

Task	Entity object method to call
Determine which fields must be filled in by the user	**GetFieldRequiredness Method**
Determine the acceptable values for each field, and which fields have invalid values	**GetFieldType Method**, **GetInvalidFieldValues Method**
Determine which fields have been updated	**GetFieldsUpdatedThisAction Method**, **GetFieldsUpdatedThisGroup Method**, **GetFieldsUpdatedThisSetValue Method**
Find other data records that are considered duplicates of this one	**GetDuplicates Method**
Find the original data record, if this one is a duplicate	**GetFieldOriginalValue Method**

Creating a New Record

To create a new record, call the **BuildEntity Method** of the Session object. The BuildEntity method creates the record with a unique ID for the given user database and initiates a "submit" action for the record. During the submit action, the record is available for editing.

Editing an Existing Record

To edit an existing record, follow these steps:

1 Acquire the Entity object you want to edit by using the methods of the Session object.

Note: To use the methods of the Session object, you must already know the definition of the record. You can use methods of the **Session Object** to have a query find records that match criteria you define, and then work with the records in the query's result set. To learn how to use the API for queries, see **Working with Queries**.

2 Call the **EditEntity Method** of the Session object.

Only one user at a time can edit a record. If you are creating a new record, you have permission to modify the contents of the record. However, if you are using the EditEntity method to modify an existing record while someone else is modifying it, the

record is locked. If another user has a prior lock on the record, you can modify the record, but you cannot commit the record to the database with your changes.

Using the methods of the **Entity Object**, you can perform these tasks:

- View or modify the values in the record's fields.
- Get additional information about the type of data in the fields or about the record as a whole.
- Change the behavior of a field for the duration of the current action.

Saving Your Changes

After you create or edit a record, save your changes to the database by following these steps:

1. Validate that data in the record by calling the **Validate Method** of the Entity object.

   This method returns any validation errors so that you can fix them before you attempt to save your changes.

2. Call the **Commit Method** of the Entity object.

   This method writes the changes to the database, ends the current action, and checks in the record so that it cannot be edited.

Reverting Your Changes

If validation of a record fails, you will not be able to commit the changes to the database. The safest solution is to revert the record to its original state and report an error.

To revert a record, call the **Revert Method** of the Entity object.

Viewing the Contents of a Record

If you do not want to edit the contents of a record, you can get the record and look at the values in its fields. To view a record, get the record using one of the methods of the **Session Object**.

To view the contents of a record by using a Session object method, follow these steps:

1. Use the **GetEntity Method** to acquire the record.
2. Use methods of the returned Entity object to access the record's fields.

   To get a list of record types by name, use the following methods of the **Session Object**.

To list the names of ...	call this Session object method
All record types	**GetEntityDefNames Method**
Record types that have states	**GetReqEntityDefNames Method**
Record types that are stateless	**GetAuxEntityDefNames Method**
Record types that belong to a record type family	**GetQueryEntityDefNames Method**
Record types you can use to create a new record	**GetSubmitEntityDefNames Method**

To get the EntityDef object associated with a particular record type, use the **GetEntityDef Method**.

Ensuring that Record Data is Current

In a multi-user 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 Entity contains a snapshot of the field's data.

To refresh your snapshot of a record, call GetFieldValue again to get a new FieldInfo object.

Viewing the Metadata of a Record

To learn how to access metadata (information about a record and its fields), see **Accessing the Schema Repository**.

Accessing the Schema Repository

Normally, you modify the schema repository (master database) using the ClearQuest Designer. However, it is possible to get information from, and make limited changes to, the schema repository using the ClearQuest API. **Performing User Administration**, for example, is among such tasks. For the complete list and definitions for Schema objects and collections. see **Schema Repository Objects**.

ClearQuest defines a set of objects for the schema repository:

Because the schema repository is different from your user databases, you cannot use the normal Session object to log on to the schema repository and access its contents. Instead, you must use an **AdminSession Object**, which provides access to the schema repository information.

Using the **AdminSession Object**, you can access information about the user databases associated with the schema repository. Each user database is represented by a **Database Object**. You can use this object to get and set information about the database, including the login IDs, passwords, and database settings.

Logging on to the Schema Repository

You must log on to the schema repository before you can access its contents. The AdminSession object controls access to the schema repository. The AdminSession object is similar in purpose to the Session object, but provides access to schemas and user profiles instead of to records.

You log on to the schema repository using the **Logon Method** of the AdminSession object. To use this method, you must know the login name and password. For more information, see the **Logon Method**.

Getting Schema Repository Objects

Most of the schema repository information can be found in the properties of various objects. For example, the AdminSession object has properties that return a complete list of the databases, schemas, users, and groups associated with the schema repository. The AdminSession object also has methods that retrieve database, user, and group objects

whose name you already know. You can also use methods of the AdminSession object to create new databases, user accounts, and groups.

Calling each of these methods creates a new object of the corresponding type. You can then set data. The information in these objects is saved immediately to the schema repository. If you are setting information related to users and groups, you must update your user databases.

Updating User Database Information

ClearQuest immediately updates data in the *schema repository*, but not data of *user databases*. To update the contents of a user database, you must call specific methods of the Database object. The Database object allows you to update the following:

- Users, groups, and database information for a specific database.
- The schema revision the database uses.

To update the user and group information associated with the user database,

1. In the schema repository, make the changes you want to the user information.
2. Call the **UpgradeMasterUserInfo Method** of the user Database object. This method copies the changes from the schema repository to the user database.

Performing User Administration

You can perform user administration and update the database from ClearQuest Designer. You can use either the User administration dialog in ClearQuest Designer, or the API, to create new user accounts and groups and manipulate the attributes of existing accounts. When you use the API, new objects you create are automatically updated in the schema repository, but they are not updated in any associated user databases until you specifically call the **UpgradeMasterUserInfo Method** of the corresponding **Database Object**.

To create a new account, call the **CreateUser Method**. This method returns a new **User Object**, which you can fill in with the user's account information, including the user's name, phone number, email address, and access privileges. You can also subscribe the user to one or more databases.

To get a User object for an existing user, call the **GetUser Method** of the **AdminSession Object**, or iterate through the objects in the **Users Property**.

To create a new group, call the **CreateGroup Method**. This method returns a new **Group Object**, to which you can add new users.

To get an existing group, call the **GetGroup Method**, or iterate through the **Groups Property**.

To add a user to a group, call the **AddUser Method** of the **Group Object**.

Note: You cannot remove User or Group objects from the schema repository. Once you create these objects, they remain permanently.

Common API Calls to Get User Information

ClearQuest also uses records to store user administration information. If you are writing hook code, this information can be useful for controlling user privileges and access permissions. You can get user administration information about the user logged into the current session by using the following methods of the **Session Object**.

User Administration Task	Session object method to call
Get the name of the current user	**GetUserLoginName Method**

User Administration Task	Session object method to call
Get a list of groups to which the user belongs.	**GetUserGroups Method**
Get a user's email address	**GetUserEmail Method**
Get a user's full name	**GetUserFullName Method**
Get a user's phone number	**GetUserPhone Method**
Get any additional information about the user	**GetUserMiscInfo Method**

AdminSession Object

An AdminSession object allows you to create a session object associated with a schema repository.

The AdminSession object is the starting point if you want to modify the information in a schema repository. Unlike the Session object, you must create an instance of AdminSession explicitly even if you are writing a hook. You create an AdminSession object as follows:

```
set adminSession = CreateObject("ClearQuest.AdminSession")
```

This creates an uninitialized admin session object, to use it you have to log into the database. To Login:

```
adminSession.Logon
void Logon login_name, password, databaseSetName
```

All three arguments are strings. Invoking this method logs you into the master database in the specified database set.

You can get various information such as users, databases associated with this master database. The AdminSession API hierarchy is:

```
AdminSession
   |-----Users
   |      |-----User
   |-----Groups
   |      |-----Group
   |-----Databases
          |-----Database
```

After creating the AdminSession object, you must log on to the schema repository using the Logon method of the AdminSession object. To log on to the database, you must know the administrator's login name and password, as well as the name of the database set containing the schema repository. Once you have logged on successfully, you can use the methods of the AdminSession object to get information from the schema repository.

Note: To learn about user administration, see **Performing User Administration** in the **Using the ClearQuest API** chapter.

Working With Databases

The AdminSession object also maintains a list of the user databases associated with the schema repository. If you know the name of the database, you can get its corresponding **Database Object** by calling the **GetDatabase Method**. If you do not know the name of the database, you can iterate through the objects in the **Databases Property** to find the one you want. You can also disassociate a user database from the schema repository by calling the **DeleteDatabase Method**.

See Also:

Session Object
User Object
Users Object
Group Object
Groups Object
Database Object

AdminSession Object Properties

The following list summarizes the AdminSession Object properties:

Property name	Access	Description
Databases Property	Read-only	Returns the collection of databases associated with the schema repository.
Groups Property	Read-only	Returns the collection of groups associated with the schema repository.
Schemas Property	Read-only	Returns the collection of schemas associated with the schema repository.
Users Property	Read-only	Returns the collection of users associated with the schema repository.

Databases Property

Description:

Returns the collection of databases associated with the schema repository. This is a read-only property; it can be viewed but not set.

Each element in the returned collection is a **Database Object**.

Perl Syntax:

$adminSession->**GetDatabases**();

VBScript Syntax:

adminSession.**Databases**

Identifier	Description
adminSession	The AdminSession object representing the current schema repository access session.
Data Type	A **Databases Object** containing the collection of all databases defined in this schema repository.

Example (in VBScript):

```
set adminSession = CreateObject("ClearQuest.AdminSession")
adminSession.Logon "admin", "admin", ""

set databaseList = adminSession.Databases
numDBs = databaseList.Count
For x = 0 to numDBs
   set dbObj = databaseList.Item(x)
   dbName = dbObj.DatabaseName
   OutputDebugString "Found database: " & dbName
Next
```

See Also:

Database Object
Databases Object

Groups Property

Description:

Returns the collection of groups associated with the schema repository. This is a read-only property; it can be viewed but not set.

Each element in the returned collection is a **Group Object**.

VBScript Syntax:

adminSession.**Groups**

Perl Syntax:

$adminSession->**GetGroups**();

Identifier	Description
adminSession	The AdminSession object representing the current schema repository access session.
Data Type	A **Groups Object** containing all of the groups in the schema repository.

Example (in VBScript):

```
set adminSession = CreateObject("ClearQuest.AdminSession")
adminSession.Logon "admin", "admin", ""

set groupList = adminSession.Groups
numGroups = groupList.Count
For x = 0 to numGroups
   set groupObj= groupList.Item(x)
   groupName = groupObj.Name
   Session.OutputDebugString "Found group: " & groupName
Next
```

Example (in Perl):

```
#Create a ClearQuest admin session
$AdminSession= CQPerlExt::CQAdminSession_Build();

#Logon as admin
$adminSession->Logon( "admin", "admin", "" );

#Get the group "Engineers" object
$groupObj = $adminSession->GetGroup( "Engineers" );
```

See Also:

Group Object

Groups Object

Schemas Property

Description:

Returns the collection of schemas associated with the schema repository. This is a read-only property; it can be viewed but not set.

Each element in the returned collection is a **Schema Object**.

VBScript Syntax:

adminSession.**Schemas**

Perl Syntax:

*$adminSession->***GetSchemas**();

Identifier	Description
adminSession	The AdminSession object representing the current schema repository access session.
Data Type	A **Schemas Object** containing all of the schemas in the master database.

Example (in VBScript):

```
set adminSession = CreateObject("ClearQuest.AdminSession")
        adminSession.Logon "admin", "admin", ""

set schemaList = adminSession.Schemas
numSchemas = schemaList.Count
For x = 0 to numSchemas
   set schemaObj = schemaList.Item(x)
   schemaName = schemaObj.Name
   OutputDebugString "Found schema: " & schemaName
Next
```

Example (in Perl):

```
#Create a ClearQuest admin session
$adminSession = $entity->CreateSession
                    "ClearQuest.AdminSession" );
```

```
#Logon as admin
$adminSession->Logon( "admin", "admin", "" );

#Get the list of schemas in the repository.
$schemaList = $adminSession->GetSchemas();

#Get the number of schemas in the repository
$numSchemas = $schemaList->Count();

#Iterate through the schemas in the repository
for ( $x=0; $x<$numSchemas; $x++ ) {
   #Get the specified item in the collection of schemas
   $schemaObj = $schemaList->Item( $x );
   #Get the name of the schema
   $schemaName = $schemaObj->GetName();
   #Output, via debugger, that the user was found
   $debugString = "Found schema: " . $schemaName;
   $adminSession->OutputDebugString( $debugString );
}
```

See Also:

Schema Object
Schemas Object

Users Property

Description:

Returns the collection of users associated with the schema repository. This is a read-only property; it can be viewed but not set.

Each element in the returned collection is a **User Object**.

VBScript Syntax:

adminSession.**Users**

Perl Syntax:

$adminSession->**GetUsers**();

Identifier	Description
adminSession	The AdminSession object representing the current schema repository access session.
Data Type	A **Users Object** containing all of the users in the schema repository.

Example (in VBScript):

```
set adminSession = CreateObject("ClearQuest.AdminSession")
adminSession.Logon "admin", "admin", ""

set userList = adminSession.Groups
numUsers = userList.Count
For x = 0 to numUsers
   set userObj = userList.Item(x)
   userName = userObj.Name
   OutputDebugString "Found user: " & userName
Next
```

Example (in Perl):

```
#Create a ClearQuest admin session
$AdminSession= CQPerlExt::CQAdminSession_Build();
#Logon as admin
$adminSession->Logon( "admin", "admin", "" );

#Get the collection of users associated with the schema #repository.
$userList = $adminSession->GetUsers();
#Get the number of users in the collection.
$numUsers = $userList->Count();

#Iterate through the users in the collection
for ( $x=0; $x<$numUsers; $x++ ) {
   #Get the specified item in the collection of users
   $userObj = $userList->Item( $x );
   #Get the login name of the user
   $userName = $userObj->GetName();
   #Output, via debugger, that the user was found
   $debugString = "Found user: " . $userName;
   $adminSession->OutputDebugString( $debugString );
```

}

See Also:

User Object
Users Object

AdminSession Object Methods

The following list summarizes the AdminSession Object methods:

Method name	Description
CreateDatabase Method	Creates a new database and associates it with the schema repository.
CreateGroup Method	Creates a new group and associates it with the schema repository.
CreateUser Method	Creates a new user and associates it with the schema repository.
DeleteDatabase Method	Disassociates the specified database from the schema repository.
GetDatabase Method	Returns the database object with the specified name.
GetGroup Method	Returns the group object with the specified name.
GetUser Method	Returns the user object with the specified name.
Logon Method	Logs the specified user into the schema repository.

CreateDatabase Method

Description:

Creates a new database and associates it with the schema repository.

The new database object does not have any of its properties set. You can set the basic database information (such as timeout intervals, login names, and passwords) by assigning appropriate values to the properties of the returned Database object. You must also call the returned object's **SetInitialSchemaRev Method** to assign a schema to the database. See the **Database Object** for more information on creating databases.

VBScript Syntax:

adminSession.**CreateDatabase** *databaseName*

Perl Syntax:

*$adminSession->***CreateDatabase**(*databaseName);*

Identifier	Description
adminSession	The AdminSession object representing the current schema repository access session.
databaseName	A String containing the name you want to give to the new database.
Return value	A **Database Object** representing the new database.

Example (in VBScript):

```
set adminSession = CreateObject("ClearQuest.AdminSession")
   adminSession.Logon "admin", "admin", ""

set newDatabaseObj = adminSession.CreateDatabase "NEWDB"
```

Example (in Perl):

```
#Create a ClearQuest admin session
$AdminSession= CQPerlExt::CQAdminSession_Build();
#Logon as admin
$adminSession->Logon( "admin", "admin", "" );

#Create the database "NEWDB" object
$newDatabaseObj = $adminSession->CreateDatabase( "NEWDB" );
```

See Also:

DeleteDatabase Method
GetDatabase Method
Databases Property
SetInitialSchemaRev Method of the Database Object

CreateGroup Method

Description:

Creates a new group and associates it with the schema repository.

The new group is subscribed to all databases by default. When you use the methods of the Group object to add users and subscribe the group to one or more databases, the groups or users are subscribed only to those you specify.

VBScript Syntax:

adminSession.**CreateGroup** *groupName*

Perl Syntax:

*$adminSession->***CreateGroup**(*groupName);*

Identifier	Description
adminSession	The AdminSession object representing the current schema repository access session.
groupName	A String containing the name you want to give to the new group.
Return value	A new **Group Object**.

Example (in VBScript):

```
set adminSession = CreateObject("ClearQuest.AdminSession")
adminSession.Logon "admin", "admin", ""

set newGroupObj = adminSession.CreateGroup "Engineers"
```

Example (in Perl):

```
#Create a ClearQuest admin session
$AdminSession= CQPerlExt::CQAdminSession_Build();
#Logon as admin
$adminSession->Logon( "admin", "admin", "" );

#Create the group "Engineers" object
$newGroupObj = $adminSession->CreateGroup( "Engineers" );
```

See Also:

GetGroup Method
Groups Property

Group Object
ApplyPropertyChanges Method of the **Database Object**

CreateUser Method

Description:

Creates a new user and associates it with the schema repository

The returned object contains no information. To add user information to this object, assign values to its properties. For information on user properties, see the **User Object**.

VBScript Syntax:

adminSession.**CreateUser** *userName*

Perl Syntax:

*$adminSession->***CreateUser**(*userName);*

Identifier	Description
adminSession	The AdminSession object representing the current schema repository access session.
userName	A String containing the name you want to give to the new user.
Return value	A new **User Object**.

Example (in VBScript):

```
set adminSession = CreateObject("ClearQuest.AdminSession")
adminSession.Logon "admin", "admin", ""

set newUserObj = adminSession.CreateUser "jsmith"
```

Example (in Perl):

```
#Create a ClearQuest admin session
$AdminSession= CQPerlExt::CQAdminSession_Build();
#Logon as admin
$adminSession->Logon( "admin", "admin", "" );
```

```
#Create the user "jsmith" object
$newUserObj = $adminSession->CreateUser( "jsmith" );
```

See Also:

GetUser Method
Users Property
User Object

DeleteDatabase Method

Description:

Disassociates the specified database from the schema repository.

This method does not actually delete the specified database. Instead, it removes all references to the database from the schema repository.

VBScript Syntax:

adminSession.**DeleteDatabase** *databaseName*

Perl Syntax:

*$adminSession->***DeleteDatabase**(*databaseName);*

Identifier	Description
adminSession	The AdminSession object representing the current schema repository access session.
databaseName	A String containing the name of the database you want to delete.
Return value	The **Database Object** that was disassociated from the schema repository.

Example (in VBScript):

```
set adminSession = CreateObject("ClearQuest.AdminSession")
adminSession.Logon "admin", "admin", ""

set newDatabase = adminSession.CreateDatabase "NEWDB"
```

...

```
' Delete the database that was created earlier.
set oldDB = adminSession.DeleteDatabase "NEWDB"
```

Example (in Perl):

```
#Create a ClearQuest admin session
$AdminSession= CQPerlExt::CQAdminSession_Build();
#Logon as admin
$adminSession->Logon( "admin", "admin", "" );

#Create a new database "NEWDB" and perform some other tasks
$newDatabase = $adminSession->CreateDatabase( "NEWDB");
...

#Delete the database that was created earlier.
$ oldDB = $adminSession->DeleteDatabase( "NEWDB" );
```

See Also:

CreateDatabase Method
GetDatabase Method
Databases Property
Database Object

GetDatabase Method

Description:

Returns the database object with the specified name.

The *databaseName* parameter corresponds to the logical database name, that is, the string in the **Name Property** of the Database object.

VBScript Syntax:

adminSession.**GetDatabase** *databaseName*

Perl Syntax:

*$adminSession->***GetDatabase**(*databaseName);*

Identifier	Description
adminSession	The AdminSession object representing the current schema repository access session.
databaseName	A String containing the name of the database object you want.
Return value	The **Database Object** with the specified name, or null if no such database exists.

Example (in VBScript):

```
set adminSession = CreateObject("ClearQuest.AdminSession")
adminSession.Logon "admin", "admin", ""

set dbObj = adminSession.GetDatabase "NEWDB"
```

Example (in Perl):

```
#Create a ClearQuest admin session
$AdminSession= CQPerlExt::CQAdminSession_Build();

#Logon as admin
$adminSession->Logon( "admin", "admin", "" );

#Get the database "NEWDB" object
$dbObj = $adminSession->GetDatabase( "NEWDB" );
```

See Also:

CreateDatabase Method
Databases Property
Name Property of the **Database Object**

GetGroup Method

Description:

Returns the group object with the specified name.

The groupName parameter corresponds to the value in the **Name Property** of the Group object.

VBScript Syntax:

adminSession.**GetGroup** *groupName*

Perl Syntax:

*$adminSession->***GetGroup**(*groupName);*

Identifier	Description
adminSession	The AdminSession object representing the current schema repository access session.
groupName	A String containing the name of the database object you want.
Return value	The **Group Object** with the specified name, or null if no such group exists.

Example (in VBScript):

```
set adminSession = CreateObject("ClearQuest.AdminSession")
adminSession.Logon "admin", "admin", ""

set groupObj = adminSession.GetGroup "Engineers"
```

Example (in Perl):

```
#Create a ClearQuest admin session
$AdminSession= CQPerlExt::CQAdminSession_Build();

#Logon as admin
$adminSession->Logon( "admin", "admin", "" );

#Get the group "Engineers" object
$groupObj = $adminSession->GetGroup( "Engineers" );
```

See Also:

CreateGroup Method
Groups Property
Name Property of the **Group Object**
Group Object

GetUser Method

Description:

Returns the user object with the specified name.

The userName parameter corresponds to the value in the **Name Property** of the User object.

VBScript Syntax:

adminSession.**GetUser** *userName*

Perl Syntax:

*$adminSession->***GetUser**(*userName);*

Identifier	Description
adminSession	The AdminSession object representing the current schema repository access session.
userName	A String containing the name of the user object you want.
Return value	The **User Object** with the specified name, or null if no such user exists.

Example (in VBScript):

```
set adminSession = CreateObject("ClearQuest.AdminSession")
adminSession.Logon "admin", "admin", ""

set userObj = adminSession.GetUser "talbert"
```

Example (in Perl):

```
#Create a ClearQuest admin session
$AdminSession= CQPerlExt::CQAdminSession_Build();
#Logon as admin
$adminSession->Logon( "admin", "admin", "" );

#Get the user "talbert" object
$userObj = $adminSession->GetUser( "talbert" );
```

See Also:

CreateUser Method
Users Property
Name Property of the **User Object**

Logon Method

Description:

Logs the specified user into the schema repository.

Call this method after creating the AdminSession object but before trying to access any elements in the schema repository. The user login and password must correspond to the ClearQuest administrator or to a user who has access to the schema repository. The administrator can grant access to users by enabling special privileges in their account. Users with the Schema Designer privilege can modify the schemas in a database. Users with the User Administrator privilege can create or modify groups and user accounts. Users with the Super User privilege have complete access to the schema repository, just like the administrator.

VBScript Syntax:

adminSession.**Logon** *login_name, password, databaseSetName*

Perl Syntax:

*$adminSession->***Logon**(*login_name, password, databaseSetName);*

Identifier	Description
adminSession	The AdminSession object representing the current schema repository access session.
login_name	A String that specifies the login name of the user.
password	A String that specifies the user's password.
databaseSetName	A String that specifies the name of the schema repository. Normally, you should set this string to the empty string (““).
Return value	None.

Example (in VBScript):

```
set adminSession = CreateObject("ClearQuest.AdminSession")
adminSession.Logon "admin", "admin", ""
```

Example (in Perl):

```
#Create a ClearQuest admin session
$AdminSession= CQPerlExt::CQAdminSession_Build();
#Logon as admin
$adminSession->Logon( "admin", "admin", "" );
```

See Also:

CreateUser Method

Users Property

UserLogon Method of the **Session Object**

Attachment Object

An Attachment object represents a single attached file that is physically stored in the user database.

An Attachment object

- stores information about that file (description, unique key, path name, and size) in the Attachment object's properties
- provides a means to manipulate the file

Note: The ClearQuest API does not permit you to alter that data inside an attached file, but it does permit you to alter the descriptive information.

To attach files to a database, use the **Add Method** of the **Attachments Object**. (You never create instances of Attachment directly.)

To retrieve an Attachment object, use the **Item Method** of the Attachments object.

To delete an Attachment object, use the **Delete Method** of the **Attachments Object**.

To copy an existing attachment to a new file, use the Load Method.

See Also:

AttachmentField Object
AttachmentFields Object
Attachments Object
Getting and Setting Attachment Information

Attachment Object Properties

The following list summarizes the Attachment Object properties:

Property name	Access	Description
Description Property	Read/Write	Sets or returns the description of the attached file.
DisplayName Property	Read-only	Returns the unique key used to identify the attachment.
FileName Property	Read-only	Returns the path name of the attached file.
FileSize Property	Read-only	Returns the size of the attached file in bytes.

Description Property

Description:

Sets or returns the description of the attached file. This is a read-write property; its value can be set unlike the other Attachment properties.

VBScript Syntax:

attachment.**Description [= *value*]**

Perl Syntax:

$attachment->**GetDescription();**
$attachment->**SetDescription(***new_value***);**

Identifier	Description
attachment	An Attachment object, representing the attachment of a file to a record.
Data Type	A String containing the descriptive text.

Example (in VBScript):

```
' This example assumes there is at least 1 attachment field
' and 1 attachment associated with the record.
set currentSession = GetSession
set attachFields = entity.AttachmentFields
set attachField1 = attachFields.Item(0)

set attachments = attachField1.Attachments
numAttachments = attachments.Count
For x = 0 to numAttachments
   set attachment = attachments.Item(x)
   description = attachment.Description
   key = attachment.DisplayName
   currentSession.OutputDebugString "Unique key: " & key & " - _
   description: " & description
Next
```

Example (in Perl):

```
# This example assumes that there is at least 1 attachment
# field associated with the record. Otherwise,
# GetAttachmentFields won't return anything interesting and an
# error would be generated

# Get the collection of attachment fields
$attachfields = $entity->GetAttachmentFields();

# Get the first attachment fields
$attachfield1 = $attachfields.Item(0)

# Now get the collection of attachments from the attachments field
$attachments = $attachfield1->GetAttachments();

# Retrieve the number of attachments for the for loop
$numattachments = $attachments->Count();

for ($x = 0 ; $x < $numattachments ; $x++)
 {
 # Retrieve the correct attachment
 $attachment = $attachments.Item($x);
```

```
  # Get the unique attachment key and the attachment description
  # and print it out
$description = $attachment->GetDescription();
$key = $attachment->GetDisplayName();
$session->OutputDebugString("Unique key: ".$key." - description:
        ".$description);
}
```

See Also:

FileName Property
Getting and Setting Attachment Information

DisplayName Property

Description:

Returns the unique key used to identify the attachment. This is a read-only property; it can be viewed but not set.

The unique key is a concatenation of the file name, file size, description, and database ID, each delimited by a newline (“\n”) character. However, the format of this property is subject to change.

VBScript Syntax:

attachment.**DisplayName**

Perl Syntax:

$attachment->**GetDisplayName()**

Identifier	Description
attachment	An Attachment object, representing the attachment of a file to a record.
Data Type	A String containing the unique key.

Example (in VBScript):

```
' This example assumes there is at least 1 attachment field
' and 1 attachment associated with the record.
```

```
set currentSession = GetSession
set attachFields = entity.AttachmentFields
set attachField1 = attachFields.Item(0)

set attachments = attachField1.Attachments
numAttachments = attachments.Count
For x = 0 to numAttachments
   set attachment = attachments.Item(x)
   description = attachment.Description
   key = attachment.DisplayName
   currentSession.OutputDebugString "Unique key: " & key & " - _
   description: " & description
            & description
Next
```

Example (in Perl):

```
# This example assumes that there is at least 1 attachment
# field associated with the record. Otherwise,
# GetAttachmentFields won't return anything interesting and an
# error would be generated

# Get the collection of attachment fields
$attachfields = $entity->GetAttachmentFields();

# Get the first attachment fields
$attachfield1 = $attachfields.Item(0)

# Now get the collection of attachments from the attachments field
$attachments = $attachfield1->GetAttachments();

# Retrieve the number of attachments for the for loop
$numattachments = $attachments->Count();

for ($x = 0 ; $x < $numattachments ; $x++)
 {
 # Retrieve the correct attachment
 $attachment = $attachments.Item($x);

   # Get the unique attachment key and the attachment description
   # and print it out
 $description = $attachment->GetDescription();
 $key = $attachment->GetDisplayName();
```

```
$session->OutputDebugString("Unique key: ".$key." - description:
        ".$description);
}
```

See Also:

Description Property
FileName Property

FileName Property

Description:

Returns the path name of the attached file.

This a read-only property; it can be viewed but not set.

Before the attachment has been committed to the database, this property contains the original path name of the file. However, once the attachment has been committed, the file exists in the database rather than in the file system, so the path information is removed. For example, if you add the file C:projectsmyfilesexample.txt, it will have that full name until the record is committed, whereupon the name will shrink to example.txt.

It is legal in ClearQuest to attach two files with the same name and different path information to the same database. ClearQuest does not rely on the filename alone when locating the file internally.

VBScript Syntax:

attachment.**FileName**

Perl Syntax:

$attachment->**GetFileName()**

Identifier	Description
attachment	An Attachment object, representing the attachment of a file to a record.
Data Type	A String containing the name of the attached file.

Example (in VBScript):

```
' This example assumes there is at least 1 attachment field
' and 1 attachment associated with the record.
set currentSession = GetSession
set attachFields = entity.AttachmentFields
set attachField1 = attachFields.Item(0)

set attachments = attachField1.Attachments
numAttachments = attachments.Count
For x = 0 to numAttachments
   set attachment = attachments.Item(x)
   fileName = attachment.FileName
   fileSize = attachment.FileSize
   currentSession.OutputDebugString "Attached file: " & fileName &
         " - size: " & fileSize
Next
```

Example (in Perl):

```
# This example assumes that there is at least 1 attachment
# field associated with the record. Otherwise,
# GetAttachmentFields
# won't return anything interesting and an error would be
# generated

# Get the collection of attachment fields
$attachfields = $entity->GetAttachmentFields();

# Get the first attachment fields
$attachfield1 = $attachfields.Item(0)

# Now get the collection of attachments from the attachments field
$attachments = $attachfield1->GetAttachments();

# Retrieve the number of attachments for the for loop
$numattachments = $attachments->Count();

for ($x = 0 ; $x < $numattachments ; $x++)
 {
 # Retrieve the correct attachment
 $attachment = $attachments.Item($x);
```

```
# Get the filename and filesize for the attachment and print out
  # the results
$filename = $attachment->GetFileName();
$filesize = $attachment->GetFileSize();
$session->OutputDebugString("Attached file: ".$filename." -
        size: ".$filesize);
}
```

See Also:

FileSize Property
Add Method of the **Attachments Object**
Commit Method of the **Entity Object**
Getting and Setting Attachment Information

FileSize Property

Description:

Returns the size of the attached file in bytes.

This a read-only property; it can be viewed but not set. This method should be called only after the attachment has been committed to the database. If you call it earlier, the return value will be empty.

VBScript Syntax:

attachment.**FileSize**

Perl Syntax:

$attachment->**GetFileSize();**

Identifier	Description
attachment	An Attachment object, representing the attachment of a file to a record.
Data Type	A Long indicating the file's size in bytes.

Example (in VBScript):

```
' This example assumes there is at least 1 attachment field
' and 1 attachment associated with the record.
set attachFields = entity.AttachmentFields
set attachField1 = attachFields.Item(0)

set attachments = attachField1.Attachments
numAttachments = attachments.Count
For x = 0 to numAttachments
   set attachment = attachments.Item(x)
   fileName = attachment.FileName
   fileSize = attachment.FileSize
   OutputDebugString "Attached file: " & fileName & " - size: " _
         & fileSize
Next
```

Example (in Perl):

```
# This example assumes that there is at least 1 attachment
# field associated with the record. Otherwise,
# GetAttachmentFields
# won't return anything interesting and an error would be generated

# Get the collection of attachment fields
$attachfields = $entity->GetAttachmentFields();

# Get the first attachment fields
$attachfield1 = $attachfields.Item(0)

# Now get the collection of attachments from the attachments field
$attachments = $attachfield1->GetAttachments();

# Retrieve the number of attachments for the for loop
$numattachments = $attachments->Count();

for ($x = 0 ; $x < $numattachments ; $x++)
 {
 # Retrieve the correct attachment
 $attachment = $attachments.Item($x);
 # Get the filename and filesize for the attachment and print out
   # the results
 $filename = $attachment->GetFileName();
 $filesize = $attachment->GetFileSize();
```

```
$session->OutputDebugString("Attached file: ".$filename." -
        size: ".$filesize);
}
```

See Also:

FileName Property

Getting and Setting Attachment Information

Attachment Object Methods

The following list summarizes the Attachment Object methods:

Method name	Description
Load Method	Writes this object's contents to the specified file

Load Method

Description:

Writes this object's contents to the specified file.

You can use this method to extract an attached file from the database and save it to your local file system. If a file with the same name already exists at the path you specify in the filename parameter, that file must writeable and its existing contents will be replaced. The extracted file is not a temporary file; it persists after the process using this API has terminated.

VBScript Syntax:

attachment.**Load** *filename*

Perl Syntax:

$attachment->**Load(***filename***);**

Identifier	Description
attachment	An Attachment object, representing the attachment of a file to a record.
filename	A String containing the path name of the file you want to write. This path name can be an absolute or relative path.
Return Type	A Boolean whose value is True if the operation was successful, otherwise False.

Example (in VBScript):

```
' This example assumes there is at least 1 attachment field
' and 1 attachment associated with the record.
set attachFields = entity.AttachmentFields
set attachField1 = attachFields.Item(0)

set attachments = attachField1.Attachments
numAttachments = attachments.Count
For x = 0 to numAttachments
   set attachment = attachments.Item(x)
   fileName = "C:\attach" & x & ".txt"
Next
```

Example (in Perl):

```
# This example assumes that there is at least 1 attachment
# field associated with the record. Otherwise,
# GetAttachmentFields
# won't return anything interesting and an error would be
# generated
# Get the collection of attachment fields
$attachfields = $entity->GetAttachmentFields();

# Get the first attachment fields
$attachfield1 = $attachfields.Item(0)

# Now get the collection of attachments from the attachments field
$attachments = $attachfield1->GetAttachments();

# Retrieve the number of attachments for the for loop
$numattachments = $attachments->Count();

for ($x = 0 ; $x < $numattachments ; $x++)
 {
 # Retrieve the correct attachment
 $attachment = $attachments.Item($x);
 # Select a filename to write to
 $filename = "C:\\attach".$x.".txt";
 # Write the file
 $attachment.Load($filename);
 }
```

See Also:

Add Method of the Attachments object
Attachments Object
Getting and Setting Attachment Information

AttachmentField Object

An AttachmentField object represents one attachment field in a record. A record can have more than one field of type attachment list. Each AttachmentField object represents a single attachment field in the record. An **AttachmentFields Object** represents the set of all the record's attachment type fields.

Note: You cannot modify the properties of this object directly. However, you can modify the attachments associated with this field. (See the **Attachment Object**.)

See Also:

Attachment Object
AttachmentFields Object
Attachments Object
Getting and Setting Attachment Information

AttachmentField Object Properties

The following list summarizes the AttachmentField Object properties:

Property name	Access	Description
Attachments Property	Read-only	Returns this attachment field's collection of attachments.
DisplayNameHeader Property	Read-only	Returns the unique keys of the attachments in this field.
FieldName Property	Read-only	Returns the name of the attachment field.

Attachments Property

Description:

Returns this attachment field's collection of attachments.

This is a read-only property; the value can be viewed but not set. However, you can still add items to (and remove items from) the collection using methods of the Attachments object.

This property always returns an Attachments object, even if there are no attached files associated with the field. If the field has no attached files, the **Count Property** of the Attachments object contains the value zero.

VBScript Syntax:

attachmentField.**Attachments**

Perl Syntax:

$attachmentField->**GetAttachments();**

Identifier	Description
field	An AttachmentField object representing one attachment field of a record.
Return value	An Attachments collection object, which itself contains a set of **Attachment Object**s.

Example (in VBScript):

```
' This example assumes there is at least 1 attachment field
' associated with the record.
set attachFields = entity.AttachmentFields
set attachField1 = attachFields.Item(0)

set attachments = attachField1.Attachments
numAttachments = attachments.Count
For x = 0 to numAttachments
   set attachment = attachments.Item(x)

   'Do something with the attachments
Next
```

Example (in Perl):

```
# This example assumes that there is at least 1 attachment
# field associated with the record. Otherwise,
# GetAttachmentFields won't return anything interesting and an
# error would be generated

# Get the collection of attachment fields
$attachfields = $entity->GetAttachmentFields();

# Get the first attachment fields
$attachfield1 = $attachfields->Item(0)

# Now get the collection of attachments from the attachments field
$attachments = $attachfield1->GetAttachments();

# Retrieve the number of attachments for the for loop
$numattachments = $attachments->Count();

for ($x = 0 ; $x < $numattachments ; $x++)
```

```
{
$attachment = $attachments->Item($x);

# ...do some work with $attachment
}
```

See Also:

DisplayNameHeader Property
FieldName Property
Getting and Setting Attachment Information
Attachment Object

DisplayNameHeader Property

Description:

Returns the unique keys of the attachments in this field.

This is a read-only property; it can be viewed but not set. The unique keys are set using ClearQuest Designer, not the ClearQuest API.

VBScript Syntax:

attachmentField.**DisplayNameHeader**

Perl Syntax:

*$attachmentField->***GetDisplayNameHeader();**

Identifier	Description
field	An AttachmentField object representing one attachment field of a record.
Return value	For VB, a Variant containing an Array whose elements are Strings. Each String contains the **DisplayName Property** of one **Attachment Object** associated with this field. For Perl, a reference to a String Array.

Example (in VBScript):

```
' This example assumes there is at least 1 attachment field
' associated with the record.
set attachFields = entity.AttachmentFields
set attachField1 = attachFields.Item(0)

keys = attachField1.DisplayNameHeader
x = 0
For Each key in keys
   OutputDebugString "Displaying key number " & x & " - " & key
   x = x + 1
Next
```

Example (in Perl):

```
# This example assumes that there is at least 1 attachment
# field associated with the record. Otherwise,
# GetAttachmentFields
# won't return anything interesting and an error would be
# generated

# Get the collection of attachment fields
$attachfields = $entity->GetAttachmentFields();

# Get the first attachment fields
$attachfield1 = $attachfields.Item(0)

# Get the list of unique keys for identifying each attachment.
$keys = attachfield1.GetDisplayNameHeader();

# Iterate through the list of keys and print the key value
$x = 0;
foreach $key (@$keys)
 {
 $session->OutputDebugString("Displaying key number".$x." -
          ".$key);
 $x++;
 }
```

See Also:

Attachments Property
FieldName Property

DisplayName Property of the Attachment object
Attachment Object
Getting and Setting Attachment Information

FieldName Property

Description:

Returns the name of the attachment field.

This is a read-only property; it can be viewed but not set. The field name is set using ClearQuest Designer, not the ClearQuest API.

VBScript Syntax:

attachmentField.**FieldName**

Perl Syntax:

*$attachmentField->***GetFieldName();**

Identifier	Description
field	An AttachmentField object representing one attachment field of a record.
Return value	A String containing the name of the field.

Example (in VBScript):

```
' This example assumes there is at least 1 attachment field
' associated with the record.
set attachFields = entity.AttachmentFields
set attachField1 = attachFields.Item(0)

name = attachField1.FieldName
```

Example (in Perl):

```
# This example assumes that there is at least 1 attachment
# field associated with the record. Otherwise,
# GetAttachmentFields won't return anything interesting and an
# error would be generated
```

```
# Get the collection of attachment fields
$attachfields = $entity->GetAttachmentFields();

# Get the first attachment fields
$attachfield1 = $attachfields.Item(0);

# And retrieve the name of that field
$name = $attachfield1->GetFieldName();
```

See Also:

Attachments Property
DisplayNameHeader Property
Getting and Setting Attachment Information

AttachmentFields Object

An AttachmentFields object represents all of the attachment fields in a record.

AttachmentFields is a collection object similar to the standard Visual Basic collection objects. It is a container for a set of AttachmentField objects. The AttachmentFields object's property and methods tell you how many items are in the collection and let you retrieve individual items. You cannot programmatically change the number of attachment fields that the record type specifies. (The ClearQuest administrator creates these fields using ClearQuest Designer.) However, you can add or remove individual attached files using the methods of the **Attachments Object**.

Every **Entity Object** has exactly one AttachmentFields object. You cannot explicitly create an AttachmentFields object. However, you can retrieve a pre-existing AttachmentFields object from a given Entity object by invoking the Entity's **AttachmentFields Property**.

See Also:

Attachment Object
AttachmentField Object
Attachments Object
Getting and Setting Attachment Information

AttachmentFields Object Properties

The following list summarizes the AttachmentFields Object properties:

Property name	Access	Description
Count Property	Read-only	Returns the number of items in the collection.

Count Property

Description:

Returns the number of items in the collection. This property is read-only.

VBScript Syntax:

collection.**Count**

Perl Syntax:

*$collection->***Count();**

Identifier	Description
collection	An AttachmentFields collection object, representing all of the attachment fields in a record.
Return value	A Long indicating the number of items in the collection object. This collection always contains at least one item.

Example (in VBScript):

```
set attachFields = entity.AttachmentFields
numFields = attachFields.Count
For x = 0 to numFields
   set oneField = attachFields.Item x
   ...
Next
```

Example (in Perl)

```
# Get the list of attachment fields
$attachfields = $entity->GetAttachmentFields()

# Find out how many attachment fields there
# are so the for loop can iterate them
$numfields = $attachfields->Count();

for ($x = 0; $x < $numfields ; $x++)
 {
 # Get each attachment field
 $onefield = $attachfields.Item($x);

 # ...do some work with $onefield
 }
```

See Also:

Item Method
Getting and Setting Attachment Information

AttachmentFields Object Methods

The following list summarizes the AttachmentFields Object methods:

Method name	Description
Item Method	Returns the specified item in the collection.

Item Method

Description:

Returns the specified item in the collection.

The argument to this method can be either a numeric index (itemNum) or a String (name).

VBScript Syntax:

collection.**Item** *itemNum*
collection.**Item** *name*

Perl Syntax:

*$collection->*__Item__(*itemNum);*
*$collection->*__ItemByName__(*name);*

Identifier	Description
collection	An AttachmentFields collection object, representing all of the attachment fields in a record.
itemNum	A Long that serves as an index into the collection. This index is 0-based so the first item in the collection is numbered 0, not 1.
name	A String that serves as a key into the collection. This string corresponds to the **FieldName Property** of the desired AttachmentField.
Return value	The AttachmentField object at the specified location in the collection.

Example (in VBScript):

```
set attachFields = entity.AttachmentFields
numFields = attachFields.Count
For x = 0 to numFields
   set oneField = attachFields.Item x
   ...
Next
```

See Also:

Count Property
Getting and Setting Attachment Information

Attachments Object

The Attachments object represents the collection (container or set) of attachments in one attachment field of a record.

This object is a container for one or more **Attachment Object**s. The Attachments object's property and methods tell you how many items are in the collection and let you retrieve, add and remove individual items.

Every **AttachmentField Object** has exactly one Attachments object. You retrieve it by retrieving the AttachmentField object's **Attachments Property**.

See Also:

Attachment Object
AttachmentField Object
Getting and Setting Attachment Information

Attachments Object Property

The following list summarizes the Attachments Object property:

Property name	Access	Description
Count Property	Read-only	Returns the number of items in the collection.

Count Property

Description:

Returns the number of items in the collection.

This property is read-only.

VBScript Syntax:

collection.**Count**

Perl Syntax:

$collection->**Count();**

Identifier	Description
collection	An Attachments collection object, representing the set of attachments in one field of a record.
Return value	A Long indicating the number of items in the collection object. This method returns zero if the collection contains no items.

Example (in VBScript):

```
' This example assumes there is at least 1 attachment field
' associated with the record.
set attachFields = entity.AttachmentFields
set attachField1 = attachFields.Item(0)

set attachments = attachField1.Attachments
```

```
numAttachments = attachments.Count
For x = 0 to numAttachments
   set attachment = attachments.Item(x)

   ' Do something with the attachments
Next
```

Example (Perl):

```
#This example assumes that there is at least
# one attachment field associated with the record

# For this entity record, get the collection of all
# attachment fields
$attachfields = $entity->GetAttachmentFields();

# Work with the first attachment field
$attachfield1 = $attachfields->Item(0);

# For this attachment field, get the collection of all
# it's attachments
$attachments = $attachfield1->GetAttachments();

# Iterate through the attachments and do some work
$numattachments = $attachments->Count();

for ($x = 0 ; $x < $numattachments ; $x++)
 {
 $attachment = $attachments->Item($x);

 # Perform some function on the attachment
 }
```

See Also:

Item Method
Getting and Setting Attachment Information

Attachments Object Methods

The following list summarizes the Attachments Object methods:

Method name	Description
Add Method	Adds an Attachment object to the collection.
Delete Method	Deletes an attached file from the collection.
Item Method	Returns the specified item in the collection.

Add Method

Description:

Adds an Attachment object to the collection.

This method creates a new Attachment object for the file and adds the object to the end of the collection. You can retrieve items from the collection using the **Item Method**.

VBScript Syntax:

attachments.**Add** *filename*, *description*

Perl Syntax:

$attachments->**Add**(*filename*, *description);*

Identifier	Description
attachments	An Attachments collection object, representing the set of attachments in one field of a record.
filename	A String containing the absolute or relative pathname of the file to be attached to this field.
description	A String that contains arbitrary text describing the nature of the attached file.
Return value	A Boolean that is True if the file was added successfully, otherwise False.

Example (in VBScript):

```
' This example assumes there is at least 1 attachment field
' associated with the record.
set attachFields = entity.AttachmentFields
set attachField1 = attachFields.Item(0)

set attachments = attachField1.Attachments
If Not attachments.Add("c;\attach1.txt", "Defect description") Then
  OutputDebugString "Error adding attachment to record."
End If
```

Example (in Perl):

```
# This example assumes that there is at least
# one attachment field associated with the record

# For this entity record, get the collection of all
# attachment fields
$attachfields = $entity->GetAttachmentFields();

# Work with the first attachment field
$attachfield1 = $attachfields->Item(0);

# For this attachment field, get the collection of all
# it's attachments
$attachments = $attachfield1->GetAttachments();

# Add the designated file and description to the
# attachments field
if (!$attchments.Add("c:\\attach1.txt","attachment description"))
 {
 $session->OutputDebugString("Error adding attachment to record.\n");
 }
```

See Also:

Count Property
Delete Method
Item Method
Getting and Setting Attachment Information

Delete Method

Description:

Deletes an attached file from the collection.

The argument to this method can be either a numeric index (itemNum) or a String (displayName). You can use the **Count Property** and **Item Method** to locate the correct Attachment object before calling this method.

VBScript Syntax:

attachments.**Delete** *itemNum*
attachments.**Delete** *displayName*

Perl Syntax:

*$attachments->***Delete**(*itemNum);*
*$attachments->***DeleteByName**(*displayName);*

Identifier	Description
attachments	An Attachments collection object, representing the set of attachments in one field of a record.
itemNum	A Long that is an index into the collection. This index is 0-based and points to the file that you want to delete.
displayName	A String that serves as a key into the collection. Its value specifies the **DisplayName Property** of the Attachment object you want to delete.
Return value	A Boolean that is True if the file was deleted successfully, otherwise False.

Example (in VBScript):

```
' This example assumes there is at least 1 attachment field
' associated with the record.
set attachFields = entity.AttachmentFields
set attachField1 = attachFields.Item(0)

set attachments = attachField1.Attachments
If Not attachments.Delete(0) Then
```

```
  OutputDebugString "Error deleting the attachment."
End If
```

Example (in Perl):

```
# This example assumes that there is at least
# one attachment field associated with the record

# For this entity record, get the collection of all
# attachment fields
$attachfields = $entity->GetAttachmentFields();

# Work with the first attachment field
$attachfield1 = $attachfields->Item(0);

# For this attachment field, get the collection of all
# it's attachments
$attachments = $attachfield1->GetAttachments();

# Delete the first attachment
if (!$attchmentsDelete(0))
 {
 $session->OutputDebugString("Error deleting attachment from
          record.\n");
 }
```

See Also:

Count Property
Add Method
Item Method
Getting and Setting Attachment Information

Item Method

Description:

Returns the specified item in the collection.

The argument to this method can be either a numeric index (itemNum) or a String (name).

VBScript Syntax:

collection.**Item** *itemNum*
collection.**Item** *name*

Perl Syntax:

*$collection->***Item**(*itemNum);*
*$collection->***ItemByName**(*name);*

Identifier	Description
collection	An Attachments collection object, representing the set of attachments in one field of a record.
itemNum	A Long that serves as an index into the collection. This index is 0-based so the first item in the collection is numbered 0, not 1.
name	A String that serves as a key into the collection. This string corresponds to the **DisplayName Property** of the desired Attachment.
Return value	The Attachment object at the specified location in the collection.

Example (in VBScript):

```
' This example assumes there is at least 1 attachment field
' associated with the record.
set attachFields = entity.AttachmentFields
set attachField1 = attachFields.Item(0)

set attachments = attachField1.Attachments
firstAttachment = attachments.Item(0)
```

Example (in Perl):

```
# This example assumes there is at least 1 attachment field
# associated with the record.
$attachFields = $entity->GetAttachmentFields();
$attachField1 = $attachFields->Item(0);

$attachments = attachField1->GetAttachments();
$firstAttachment = $attachments->Item(0);
```

See Also:

Count Property
Getting and Setting Attachment Information

CHARTMGR Object

The CHARTMGR object provides an interface for creating charts.

You can use this object to write external applications to execute charts defined in the ClearQuest workspace. You can also modify the properties of this object to set the attributes of the chart.

1. Verify that the WORKSPACE object is associated with a Session object.
2. Call the **GetChartMgr Method** of the **WORKSPACE Object**.
3. Execute a query by calling the ResultSet object's **Execute Method**.
4. Specify the data to use for the chart by calling the **SetResultSet Method** and specifying a ResultSet object containing the data your query generated.
5. Specify the chart to use in creating the image and generate the image.

   Note: To generate a JPEG image, call the **MakeJPEG Method**. To generate a Portable Network Graphics (PNG) image, call the **MakePNG Method**.

See Also:

ResultSet Object
WORKSPACE Object

ChartMgr Object Properties

The following list summarizes the ChartMgr Object properties:

Property name	Access	Description
GrayScale Property	Read/Write	Gets or sets the Bool that indicates whether or not the chart should be created as a grayscale image.
Height Property	Read/Write	Sets or gets the height of the image.
Interlaced Property	Read/Write	Sets or gets whether or not PNG images are interlaced.
OptimizeCompression Property	Read/Write	Sets or gets whether or not the image compression is optimized.
Progressive Property	Read/Write	Sets or gets whether or not to create progressive JPEG images.
Quality Property	Read/Write	Sets or gets the quality factor used to generate the image.
Width Property	Read/Write	Sets or gets the width of the image.

GrayScale Property

Description:

Gets or sets the Bool that indicates whether or not the chart should be created as a grayscale image.

This property is set to False by default. You can set it to True if you want to generate grayscale images.

VBScript Syntax:

chartMgr.**GrayScale**
chartMgr.**GrayScale** *isGrayScale*

Perl Syntax:

$chartMgr->**GetGrayScale();**
$chartMgr->**SetGrayScale(*boolean_value*)**

Identifier	Description
chartMgr	The CHARTMGR object associated with the current session.
isGrayScale	True if the image should be created in grayscale, otherwise False.
Return value	True if the image should be rendered as a grayscale image, otherwise False to indicate the image will be rendered in color.

See Also:

MakeJPEG Method
MakePNG Method

Height Property

Description:

Sets or gets the height of the image.

You must set the height and width of the image separately. By default, ClearQuest sets the height of images to 500 pixels.

VBScript Syntax:

chartMgr.**Height**
chartMgr.**Height** *newHeight*

Perl Syntax:

$chartMgr->**GetHeight();**
$chartMgr->**SetHeight(*integer_for_height_in_pixels*);**

Identifier	Description
chartMgr	The CHARTMGR object associated with the current session.
newHeight	An INT indicating the new height of the image in pixels.
Return value	An INT indicating the height of the image in pixels

See Also:

Width Property
MakeJPEG Method
MakePNG Method

Interlaced Property

Description:

Sets or returns whether or not PNG images are interlaced.

This property is used when producing PNG images. By default, this property is set to True.

VBScript Syntax:

chartMgr.**Interlaced**
chartMgr.**Interlaced** *isInterlaced*

Perl Syntax:

$chartMgr->**GetInterlaced();**
$chartMgr->**SetInterlaced(*boolean_value*);**

Identifier	Description
chartMgr	The CHARTMGR object associated with the current session.

Identifier	Description
isInterlaced	A Bool indicating whether or not PNG images should be created in multiple passes.
Return value	True if the MakePNG method will create an interlaced PNG image, otherwise False.

See Also:

Progressive Property
MakePNG Method

OptimizeCompression Property

Description:

Sets or gets whether or not the image compression is optimized.

By default, this property is set to True.

Syntax:

chartMgr.**OptimizeCompression**
chartMgr.**OptimizeCompression** *useCompression*

Perl Syntax:

$chartMgr->**GetOptimizeCompression();**
$chartMgr->**SetOptimizeCompression(*boolean_value*);**

Identifier	Description
chartMgr	The CHARTMGR object associated with the current session.
useCompression	A Bool indicating whether or not the image compression should be optimized
Return value	True if the image compression should be optimized, otherwise False.

See Also:

Quality Property

Progressive Property

Description:

Sets or gets whether or not to create progressive JPEG images.

This property is used when producing JPEG images. By default, this property is set to False.

Syntax:

chartMgr.**Progressive**
chartMgr.**Progressive** *isProgressive*

Perl Syntax:

$chartMgr->**GetProgressive();**
$chartMgr->**SetProgressive(*boolean_value*);**

Identifier	Description
chartMgr	The CHARTMGR object associated with the current session.
isProgressive	A Bool indicating whether or not JPEG images should be created in multiple passes.
Return value	True if the MakeJPEG method will create a progressive JPEG image, otherwise False.

See Also:

Interlaced Property
MakeJPEG Method

Quality Property

Description:

Sets or gets the quality factor used to generate the image.

You use this property to determine how much time should be spent in generating an image. Higher values indicate better compression but also mean that the image takes

more processing time to create. By default, this property is set to 100 for maximum compression.

Syntax:

chartMgr.**Quality**
chartMgr.**Quality** *newValue*

Perl Syntax:

$chartMgr->**GetQuality();**
$chartMgr->**SetQuality(*boolean_value*);**

Identifier	Description
chartMgr	The CHARTMGR object associated with the current session.
newValue	An INT between 1 and 100 indicating the new compression factor of the image.
Return value	An INT between 1 and 100 indicating the compression factor of the image.

See Also:

OptimizeCompression Property

Width Property

Description:

Sets or gets the width of the image.

You must set the height and width of the image separately. By default, ClearQuest sets the width of images to 800 pixels.

VBScript Syntax:

chartMgr.**Width**
chartMgr.**Width** *newHeight*

Perl Syntax:

$chartMgr->**GetWidth();**
$chartMgr->**SetWidth(*integer_for_width_in_pixels*);**

Identifier	Description
chartMgr	The CHARTMGR object associated with the current session.
newWidth	An INT indicating the new width of the image in pixels.
Return value	An INT indicating the new width of the image in pixels.

See Also:

Height Property
MakeJPEG Method
MakePNG Method

ChartMgr Object Methods

The following list summarizes the ChartMgr Object methods:

Method name	Description
MakeJPEG Method	Creates a JPEG image of the chart.
MakePNG Method	Creates a PNG image of the chart.
SetResultSet Method	Sets the result set used to generate the chart.

MakeJPEG Method

Description:

Creates a JPEG image of the chart.

This image takes the data in the current result set and generates a JPEG image using the current settings. If the image was created successfully, this method displays the image in the ClearQuest client.

VBScript Syntax:

chartMgr.**MakeJPEG** *chartName*

Perl Syntax:

$chartMgr->**MakeJPEG**(*chartName*);

Identifier	Description
chartMgr	The CHARTMGR object associated with the current session.
chartName	A String containing the pathname of the chart to use when generating the image.
Return value	True if the image was created successfully, otherwise False.

See Also:

Height Property
OptimizeCompression Property
Progressive Property
Quality Property
Width Property
SetResultSet Method

MakePNG Method

Description:

Creates a PNG image of the chart.

This image takes the data in the current result set and generates a PNG image using the current settings. If the image was created successfully, this method displays the image in the ClearQuest client.

VBScript Syntax:

chartMgr.**MakePNG** *chartName*

Perl Syntax:

$chartMgr->**MakePNG**(*chartName*);

Identifier	Description
chartMgr	The CHARTMGR object associated with the current session.
chartName	A String containing the pathname of the chart to use when generating the image.
Return value	None.

See Also:

Height Property
Interlaced Property
OptimizeCompression Property

Quality Property
Width Property
SetResultSet Method

SetResultSet Method

Description:

Sets the result set used to generate the chart.

You must call this method before calling either MakeJPEG or MakePNG. This method provides the data set from which the specified chart will be generated. You must call the Execute method of the ResultSet object to generate the data before calling this method.

VBScript Syntax:

chartMgr.**SetResultSet** *resultSet*

Perl Syntax:

$chartMgr->**SetResultSet**(*resultSet*);

Identifier	Description
chartMgr	The CHARTMGR object associated with the current chartMgr.
resultSet	The ResultSet object containing the data to use when generating charts.
Return value	None.

See Also:

MakeJPEG Method
MakePNG Method
Execute Method of the ResultSet object
ResultSet Object

Database Object

A Database object stores information about a user database.

Use the Database object to change the properties associated with a database. Using the properties of this object, you can get and set the database name, descriptive information, timeout intervals, and login information. You can also use the methods of this object to adjust the schema revision associated with the database.

Setting a property does not automatically update the corresponding value in the database. To update the values in the database, you must call the **ApplyPropertyChanges Method**. When you call this method, ClearQuest updates the values of any database properties that have changed.

To set the schema revision of a new database, create the database, then call the database object's SetInitialSchemaRev method.

To change the schema revision of an existing database, call the database object's Upgrade method.

To create a new user database by using the Database object, follow these steps:

1. Create the database by calling the **CreateDatabase Method** of the current AdminSession object.
2. Set the initial schema revision by using the **SetInitialSchemaRev Method**.

   Note: As new schema revisions become available, update the database by using the **Upgrade Method**.

The following example show you how to create a database and set its initial schema revision.

Example (in VBScript):

```
set adminSession = CreateObject("ClearQuest.AdminSession")
set db = adminSession.CreateDatabase("newDB")

' Set initial schema to first revision of "mySchema"
set schemas = adminSession.Schemas
set mySchema = schemas.Item("mySchema")
set schemaRevs = mySchema.SchemaRevs
set firstRev = schemaRevs.Item(1)
db.SetInitialSchemaRev(firstRev)

db.ApplyPropertyChanges
```

Example (in Perl):

```
$AdminSession= CQPerlExt::CQAdminSession_Build();
$SessionObj = CQPerlExt::CQSession_Build();

#Create a database
$db = $adminSession->CreateDatabase("newDB");

#From the list of schemas from the schema repository, get the
#"mySchema" schema
$schemas = $adminSession->GetSchemas();
$mySchema = $schemas->ItemByName("mySchema");

#From the list of all the revisions associated with "mySchema",
#get the first revision in the list
$schemaRevs = $mySchema->GetSchemaRevs();
$firstRev = $schemaRevs->Item(1);

#Set initial schema to first revision of "mySchema"
$db->SetInitialSchemaRev($firstRev);

#You do not need to call the ApplyPropertyChanges method after
#a call to the SetInitialSchemaRev or Upgrade methods
```

See Also:

CreateDatabase Method of the AdminSession object
AdminSession Object
Schema Object
SchemaRev Object

Database Object Properties

The following list summarizes the Database Object properties:

Property name	Access	Description
CheckTimeoutInterval Property	Read/Write	Sets or returns the interval at which to check for user timeouts.
ConnectHosts Property	Read/Write	Sets or returns the host name list for the physical location of a database server.
ConnectProtocols Property	Read/Write	Sets or returns the network protocol list for the database server.
DatabaseName Property	Read/Write	Sets or returns the physical name of the database.
DBOLogin Property	Read/Write	Sets or returns the database owner's login name.
DBOPassword Property	Read/Write	Sets or returns the database owner's password.
Description Property	Read/Write	Sets or returns the descriptive comment associated with the database.
Name Property	Read/Write	Sets or returns the logical database name.
ROLogin Property	Read/Write	Sets or returns the login name for users who have read-only access to the database.
ROPassword Property	Read/Write	Sets or returns the password for users who have read-only access to the database.
RWLogin Property	Read/Write	Sets or returns the login name for users who have read/write access to the database.
RWPassword Property	Read/Write	Sets or returns the password for users who have read/write access to the database.
SchemaRev Property	Read-only	Returns the schema revision currently in use by the database.
Server Property	Read/Write	Sets or returns the name of the server on which the database resides.

Property name	Access	Description
SubscribedGroups Property	Read-only	Returns the groups that are explicitly subscribed to this database.
SubscribedUsers Property	Read-only	Returns the users that are explicitly subscribed to this database.
TimeoutInterval Property	Read/Write	Sets or returns the user timeout interval.
Vendor Property	Read/Write	Sets or returns the vendor type of the database.

CheckTimeoutInterval Property

Description:

Sets or returns the interval at which to check for user timeouts.

ClearQuest uses this property to determine how often it should check the status of user connections. When the specified interval is up, ClearQuest checks each user connection for activity. If no activity has been detected recently, ClearQuest checks the TimeoutInterval property to see if the user's connection has timed out.

VBScript Syntax:

database.**CheckTimeoutInterval [= *value*]**

Perl Syntax:

*$database->**Get*CheckTimeoutInterval();**
*$database->**Set*CheckTimeoutInterval(*newValue*);**

Identifier	Description
database	A Database object.
value	A Long value indicating the number of milliseconds between checks.

See Also:

TimeoutInterval Property

ConnectHosts Property

Description:

Sets or returns the host name list for the physical location of the database server.

This property is used only in conjunction with databases who Vendor property is SQL_ANYWHERE. This property corresponds to the SQL Anywhere HOST database server option.

VBScript Syntax:

database.**ConnectHosts [= *value*]**

Perl Syntax:

*$database->**Get*ConnectHosts();**
*$database->**Set*ConnectHosts(string_for_the_host_connection);**

Identifier	Description
database	A Database object.
value	[A VB array of Strings or] a Perl reference to a string array. Each String in the array contains the host name list for a physical location of a database server.

See Also:

ConnectProtocols Property

ConnectProtocols Property

Description:

Sets or returns the network protocol list for the database server.

This property is used only in conjunction with databases who Vendor property is SQL_ANYWHERE. This property corresponds to the SQL Anywhere -x database server option.

VBScript Syntax:

database.**ConnectProtocols [= *value*]**

Perl Syntax:

*$database->***GetConnectProtocols();**
*$database->***SetConnectProtocols(string_for_the_host_connection);**

Identifier	Description
database	A Database object.
value	[A VB array of Strings or] a Perl reference to an array of Strings. Each String in the array contains the name of a network protocol for the database server.

See Also:

ConnectHosts Property

DatabaseName Property

Description:

Sets or returns the physical name of the database.

VBScript Syntax:

database.**DatabaseName *[= value]***

Perl Syntax:

*$database->****Get*****DatabaseName()**;
*$database->****Set*****DatabaseName(string_for_physical_database_name)**;

Identifier	Description
database	A Database object.
value	A string containing the physical name of the database, including any associated path information.

See Also:

Name Property

DBOLogin Property

Description:

Sets or returns the database owner's login name.

The database owner is the same as the database administrator. This property is used primarily in conjunction with SQL Server databases.

VBScript Syntax:

database.**DBOLogin [= *value*]**

Perl Syntax:

*$database->**Get*DBOLogin();**
*$database->**Set*DBOLogin(*string_for_db-owner_login_name*);**

Identifier	Description
database	A Database object.
value	A String containing the database owner's login name.

See Also:

DBOPassword Property

DBOPassword Property

Description:

Sets or returns the database owner's password.

The database owner is the same as the database administrator. This property is used primarily in conjunction with SQL Server databases.

VBScript Syntax:

database.**DBOPassword [= *value*]**

Perl Syntax:

*$database->**Get**DBOPassword();*
*$database->**Set**DBOPassword(**string_for_db-owner_password**);*

Identifier	Description
database	A Database object.
value	A String containing the database owner's password.

See Also:

DBOLogin Property

Description Property

Description:

Sets or returns the descriptive comment associated with the database.

Use this property to store additional information, such as the purpose of the database.

VBScript Syntax:

database.**Description [= *value*]**

Perl Syntax:

*$database->Get**Description();*
*$database->Set**Description(**string_describing_database**);*

Identifier	Description
database	A Database object.
value	A String containing the database's comment text.

See Also:

Name Property

Name Property

Description:

Sets or returns the logical database name.

The logical database name is the name to use when referring to the database from VBScript code or within queries. This property differs from the DatabaseName property, which specifies the name of the database file on the server's local file system.

Note: The local database name must be no longer than 5 characters.

VB Syntax:

database.**Name [= *value*]**

Perl Syntax:

*$database->**Get***Name**();**
*$database->**Set***Name**(*string_for_database_name*);**

Identifier	Description
database	A Database object.
value	A String containing the logical database name.

See Also:

DatabaseName Property

ROLogin Property

Description:

Sets or returns the login name for users who have read-only access to the database.

This property is used only in conjunction with databases whose Vendor property is SQL_SERVER.

The read-only login name and password are for users who need to view the information in the database but who are not allowed to modify the contents of the database.

VB Syntax:

database.**ROLogin [= *value*]**

Perl Syntax:

*$database->**Get*ROLogin();**
*$database->**Set*ROLogin(*string_for_read-only_login_name*);**

Identifier	Description
database	A Database object.
value	A String containing the read-only login name.

See Also:

ROPassword Property
RWLogin Property
Vendor Property
Notation Conventions for Perl
Notation Conventions for VBScript

ROPassword Property

Description:

Sets or returns the password for users who have read-only access to the database.

This property is used only in conjunction with databases whose Vendor property is set to SQL_SERVER.

The read-only login name and password are for users who need to view the information in the database but who are not allowed to modify the contents of the database.

VB Syntax:

database.**ROPassword [= *value*]**

Perl Syntax:

*$database->**Get*ROPassword();**
*$database->**Set*ROPassword(*string_for_read-only_password*);**

Identifier	Description
database	A Database object.
value	A String containing the read-only password.

See Also:

ROLogin Property
RWPassword Property
Vendor Property
Notation Conventions for Perl
Notation Conventions for VBScript

RWLogin Property

Description:

Sets or returns the login name for users who have read/write access to the database.

This property is used in conjunction with SQL Server and SQL Anywhere databases.

The read/write login name and password are for general-purpose users who need to modify and view information in the database.

VB Syntax:

database.**RWLogin [= *value*]**

Perl Syntax:

*$database->**Get*RWLogin();**
*$database->**Set*RWLogin(*string_for_read-write_login_name*);**

Identifier	Description
database	A Database object.
value	A String containing the read/write login name.

See Also:

ROLogin Property
RWPassword Property
Vendor Property

RWPassword Property

Description:

Sets or returns the password for users who have read/write access to the database.

This property is used in conjunction with SQL Server and SQL Anywhere databases.

The read/write login name and password are for general-purpose users who need to modify and view information in the database.

VB Syntax:

database.**RWPassword [= *value*]**

Perl Syntax:

*$database->**Get*RWPassword();**
*$database->**Set*RWPassword(*string_for_read-write-user_password*);**

Identifier	Description
database	A Database object.
value	A String containing the read/write password name.

See Also:

ROPassword Property
RWLogin Property
Vendor Property

SchemaRev Property

Description:

Returns the schema revision currently in use by the database.

This is a read-only property; it can be viewed but not set.

To change the schema revision of an existing database, you must upgrade the database by calling the Upgrade method. If you are creating a new database, you can set its initial schema revision using the SetInitialSchemaRev method.

VB Syntax:

database.**SchemaRev**

Perl Syntax:

*$database->**Get***SchemaRev();**

Identifier	Description
database	A Database object.
Return value	An SchemaRev object corresponding to the schema revision in use by this database.

See Also:

SetInitialSchemaRev Method
Upgrade Method
SchemaRev Object

Server Property

Description:

Sets or returns the name of the server on which the database resides.

VB Syntax:

database.**Server [= *value*]**

Perl Syntax:

*$database->**Get***Server();
*$database->**Set***Server(*string_for_database_server_name*);

Identifier	Description
database	A Database object.
value	A String containing the name of the server.

See Also:

DatabaseName Property

SubscribedGroups Property

Description:

Returns the groups explicitly subscribed to this database.

This is a read-only property; it can be viewed but not set. Each element in the returned collection is a Group object. This property does not return the groups that are implicitly subscribed to all databases.

VB Syntax:

database.**SubscribedGroups**

Perl Syntax:

*$database->**Get***SubscribedGroups();

Identifier	Description
database	A Database object.
Return value	A Groups collection object containing the groups explicitly subscribed to this database.

See Also:

Group Object
Groups Object

SubscribedUsers Property

Description:

Returns the users that are explicitly subscribed to this database.

This is a read-only property; it can be viewed but not set. Each element in the returned collection is an User object. This property does not return the users that are implicitly subscribed to all databases.

VB Syntax:

database.**SubscribedUsers**

Perl Syntax:

*$database->**Get***SubscribedUsers();**

Identifier	Description
database	A Database object.
Return value	A Users collection object containing the users explicitly subscribed to this database.

See Also:

User Object
Users Object

TimeoutInterval Property

Description:

Returns or sets the user timeout interval.

ClearQuest periodically checks user connections and disconnects users who have been idle for too long. If a user has been idle for a period of time greater than the value in this property, ClearQuest disconnects the user's session.

VB Syntax:

database.**TimeoutInterval [= *value*]**

Perl Syntax:

*$database->**Get*TimeoutInterval();**
*$database->**Set*TimeoutInterval(*timeout_inverval*);**

Identifier	Description
database	A Database object.
value	A Long value indicating the number of milliseconds a user may be idle before being disconnected from the database.

See Also:

CheckTimeoutInterval Property

Vendor Property

Description:

Sets or returns the vendor type of the database.

VB Syntax:

database.**Vendor [= *value*]**

Perl Syntax:

*$database->**Get***Vendor();
*$database->**Set***Vendor(***constant_for_a_database_vendor***);**

Identifier	Description
database	A Database object.
value	A Short containing one of the DatabaseVendor enumerated constants.

See Also:

DatabaseVendor Constants
Enumerated Constants

Databases Object Methods

The following list summarizes the Database Object methods:

Method name	Description
ApplyPropertyChanges	Updates the database's writable properties with any recent changes.
SetInitialSchemaRev	Sets the initial schema revision of a new database.
Upgrade	Upgrade this database to the specified schema revision.
UpgradeMasterUserInfo	Upgrade this database's user information.

ApplyPropertyChanges Method

Description:

Updates the writable properties of the user database's with any recent schema changes.

Call this method after you have set the properties of the user database to update the corresponding values in the database. If you do not call this method, any recent changes you made to the database will be lost. You do not have to call this method after a call to either the SetInitialSchemaRev or Upgrade method.

VB Syntax:

database.**ApplyPropertyChanges**

Perl Syntax:

$database->**ApplyPropertyChanges**();

Identifier	Description
database	A Database object.
Return value	None.

See Also:

SetInitialSchemaRev Method
Upgrade Method

SetInitialSchemaRev Method

Description:

Sets the initial schema revision of a new database.

After creating a new database, immediately call this method to set the database's initial schema revision. Calling this method on an existing database has no effect.

VB Syntax:

database.**SetInitialSchemaRev** *schemaRev*

Perl Syntax:

*$database->***SetInitialSchemaRev**(*schemaRev);*

Identifier	Description
database	A Database object.
schemaRev	A SchemaRev object corresponding to the desired schema revision.
Return value	None.

See Also:

Upgrade Method
SchemaRev Property

Upgrade Method

Description:

Upgrade this database to the specified schema revision.

Call this method to update the schema revision of an existing database. Do not use this method to set the initial schema revision of the database; use the SetInitialSchemaRev method instead.

VB Syntax:

database.**Upgrade** *schemaRev*

Perl Syntax:

$database->**Upgrade**(*schemaRev);*

Identifier	Description
database	A Database object.
schemaRev	A SchemaRev object corresponding to the desired schema revision.
Return value	None.

See Also:

SetInitialSchemaRev Method
UpgradeMasterUserInfo Method
SchemaRev Property

UpgradeMasterUserInfo Method

Description:

Upgrade this database's user information.

This method updates the user information in the master database. You should call this function to upgrade the appropriate databases after making changes to the users and groups of the master database.

VB Syntax:

database.**UpgradeMasterUserInfo**

Perl Syntax:

*$database->***UpgradeMasterUserInfo**();

Identifier	Description
database	A Database object.
Return value	None.

See Also:

Upgrade Method

DatabaseDesc Object

The DatabaseDesc object provides information about a particular database.

If you already know which database to log on to, you do not need to obtain a DatabaseDesc object to logon to the database. However, suppose that you want to have a logon dialog that presents to the user a list of the available databases. You can call the Session object's **GetAccessibleDatabases Method**, which returns a list of DatabaseDesc objects.

When you have a DatabaseDesc object, you can

- find the name of a particular database by using the **GetDatabaseName Method**
- find the name of the **database set** of which the database is a member by using the **GetDatabaseSetName Method**
- get a "direct connect" string by using the **GetDatabaseConnectString Method** (ODBC experts can use this string to log on to the database)

You can also use a DatabaseDesc object inside a hook. In this case, you would call the Session object's **GetSessionDatabase Method** to retrieve the DatabaseDesc object that has information about the current database.

See Also:

GetSessionDatabase Method of the Session object
Session Object

DatabaseDesc Object Methods

Method name	Description
GetDatabaseConnectString Method	Returns the "direct connect" string for logging into the database.
GetDatabaseName Method	Returns the name of the database.
GetDatabaseSetName Method	Returns the name of the **database set** of which this database is a member.
GetDescription Method	Returns a string describing the contents of the database.
GetIsMaster Method	Returns a Bool indicating whether this database is a master database.
GetLogin Method	Returns the database login associated with the current user.

GetDatabaseConnectString Method

Returns the "direct connect" string for logging into the database.

This method returns a database-specific “direct connect” string suitable for passing to an ODBC interface. The normal way of logging into a database is by invoking the Session object’s **UserLogon Method**. This method can be useful for experts who want to use DAO or other ODBC methods to read the ClearQuest database.

VBScript Syntax:

dbDesc.**GetDatabaseConnectString**

Perl Syntax:

*$dbDesc->***GetDatabaseConnectString***();*

Parameter	Description
dbDesc	A DatabaseDesc object containing information about one of the installed databases.
Return Value	A String whose value is the “direct connect” string.

Example (in VBScript):

The following example shows you how to logon to the database from a Visual Basic application.

```
set sessionObj = CreateObject("CLEARQUEST.SESSION")
' Login to each database successively.
set databases = sessionObj.GetAccessibleDatabases
For Each db in databases
   dbConnectString = db.GetDatabaseConnectString


Next
```

Example (in Perl):

```
#Start a ClearQuest session
$Session = CQSession::Build();

#Get a list of accessible databases
@databases = $sessionObj->GetAccessibleDatabases("ClearQuest
          2.0", "", "");

#Foreach accessible database, login as joe with password gh36ak3
foreach $db ( @databases ) {
#Get a "direct connect" string that ODBC experts
#can use to logon to the database
   $dbConnectString = $db->GetDatabaseConnectString();
}
```

See Also:

UserLogon Method of the Session object
Session Object
Getting Session and Database Information

GetDatabaseName Method

Returns the name of the database.

You can use the Session object's **GetAccessibleDatabases Method** to obtain a list of DatabaseDesc objects, and then use GetDatabaseName to get the name of each one.

You use the name of the database as an argument to the Session object's **UserLogon Method**.

VBScript Syntax:

dbDesc.**GetDatabaseName**

Perl Syntax:

*$dbDesc->***GetDatabaseName***();*

Parameter	Description
dbDesc	A DatabaseDesc object containing information about one of the installed databases.
Return Value	A String containing the name of the database.

Example (in VBScript):

The following example shows you how to logon to the database from a Visual Basic application.

```
set sessionObj = CreateObject("CLEARQUEST.SESSION")

' Login to each database successively.
set databases = sessionObj.GetAccessibleDatabases
For Each db in databases
   If Not db.GetIsMaster Then
         dbName = db.GetDatabaseName
         'Logon to the database
         sessionObj.UserLogon "tom", "gh36ak3", dbName,
               AD_PRIVATE_SESSION, ""
   End If
  ...
Next
```

Example (in Perl):

```
#Start a ClearQuest session
$Session = CQSession::Build();
```

```
#Get a list of accessible databases
@databases = $sessionObj->GetAccessibleDatabases("ClearQuest
          2.0", "", "");

#Foreach accessible database that is not the master database, #Login as user "tom" with password
"gh36ak3"
foreach $db ( @databases ) {
   if (! $db->GetIsMaster() ) {
         $dbName = $db->GetDatabaseName();
         # Logon to the database
$sessionObj->UserLogon( "tom", "gh36ak3", $dbName, AD_PRIVATE_SESSION, "" );
}
...
}
```

See Also:

GetDatabaseSetName Method
GetAccessibleDatabases Method of the Session object
Session Object
Getting Session and Database Information
Notation Conventions for VBScript
Notation Conventions for Perl

GetDatabaseSetName Method

Returns the name of the **database set** of which this database is a member.

You can use this method to get the database set name of this database. You can pass this name to the Session object's **GetAccessibleDatabases Method** to get a list of the user databases in the database set.

Note: By default, systems have only one database set. You can refer to this default database set using an empty string ("") instead of the name returned by this method.

VBScript Syntax:

dbDesc.**GetDatabaseSetName**

Perl Syntax:

*$dbDesc->***GetDatabaseSetName***();*

Parameter	Description
dbDesc	A DatabaseDesc object containing information about one of the installed databases.
Data Type	A String containing the name of the database set.

Example (in VBScript):

The following example shows you how to logon to the database from a Visual Basic application.

```
set sessionObj = CreateObject("CLEARQUEST.SESSION")

' Login to each database successively.
set databases = sessionObj.GetAccessibleDatabases
For Each db in databases
   If Not db.GetIsMaster Then
         bSetName = db.GetDatabaseSetName
          dbName = db.GetDatabaseName
         ' Logon to the database
         sessionObj.UserLogon "tom", "gh36ak3", dbName,
               AD_PRIVATE_SESSION, dbSetName
   End If
   ...
Next
```

Example (in Perl):

```
#Start a ClearQuest session
$sessionObj = CQPerlExt::CQSession_Build();

#Get a list of accessible database description objects
@databases = $sessionObj->GetAccessibleDatabases("ClearQuest
          2.0", "", "");

#Foreach accessible database that is not the master database,
    login as
#user "tom" with password "gh36ak3"
```

```
foreach $db ( @databases ) {
   if (! $db->GetIsMaster() ) {
         #Get the database set of which this database is a member
         $dbSetName = $db->GetDatabaseSetName();
         #Get the database name from the description object
         $dbName = $db->GetDatabaseName();
         # Logon to the database
$sessionObj->UserLogon( "tom", "gh36ak3", $dbName, AD_PRIVATE_SESSION,
$dbSetName );
}
...
}
```

See Also:

GetDatabaseName Method
GetAccessibleDatabases Method of the **Session Object**
Getting Session and Database Information
Notation Conventions for VBScript
Notation Conventions for Perl

GetDescription Method

Returns a string describing the contents of the database.

The description string is initially set when the database is created in ClearQuest Designer. To modify this string programmatically, you must modify the Description property of the Database object.

VBScript Syntax:

dbDesc.**GetDescription**

Perl Syntax:

*$dbDesc->***GetDescription***();*

Parameter	Description
dbDesc	A DatabaseDesc object containing information about one of the installed databases.
Data Type	A String containing descriptive comments about the database.

Example (in VBScript):

The following example shows you how to logon to the database from a Visual Basic application.

```
set sessionObj = CreateObject("CLEARQUEST.SESSION")
' Login to each database successively.
set databases = sessionObj.GetAccessibleDatabases
For Each db in databases
   dbDescription = db.GetDescription
   ...
Next
```

Example (in Perl):

```
#Start a ClearQuest session
$sessionObj = CQPerlExt::CQSession_Build();

#Get a list of accessible database description objects
@databases = $sessionObj->GetAccessibleDatabases("ClearQuest
        2.0", "", "");

#For each accessible database, get the description
#of the contents of the database
foreach $db ( @databases ) {
   $dbDescription = $db->GetDescription();
...
}
```

See Also:

Description Property of the Database object
Database Object

GetIsMaster Method

Returns a Boolean indicating whether this database is a master database.

A master database is a schema repository for one or more user databases. When manipulating the master database, you should use the methods of the AdminSession object.

VBScript Syntax:

dbDesc.**GetIsMaster**

Perl Syntax:

*$dbDesc->***GetIsMaster***();*

Parameter	Description
dbDesc	A DatabaseDesc object containing information about one of the installed databases.
Data Type	True if this database is a master database, otherwise false.

Example (in VBScript):

The following example shows you how to logon to the database from a Visual Basic application.

```
set sessionObj = CreateObject("CLEARQUEST.SESSION")
' Login to each database successively.
set databases = sessionObj.GetAccessibleDatabases
For Each db in databases
   If db.GetIsMaster Then
        ' Create an AdminSession object and logon to the schema
        ' repository.
        ...
   ElseIf
        'Logon to the database using the regular Session object.
        ...
   End If
Next
```

Example (in Perl):

```
#Start a ClearQuest session
$sessionObj = CQPerlExt::CQSession_Build();

#Get a list of accessible database description objects
@databases = $sessionObj->GetAccessibleDatabases("ClearQuest
          2.0", "", "");

#Foreach accessible database that is the master database
foreach $db ( @databases ) {
   if ( $db->GetIsMaster() ) {
         #Create an AdminSession and logon to the schema repository
         ...
   }
   else {
         #Logon to the database using the regular Session object
         ...
   }
}
```

See Also:

Logon Method of the AdminSession object
AdminSession Object

GetLogin Method

Returns the database login associated with the current user.

VBScript Syntax:

loginvalue dbDesc.**GetLogin**

Perl Syntax:

*$dbDesc->***GetLogin***();*

Parameter	Description
dbDesc	A DatabaseDesc object containing information about one of the installed databases.
Data Type	A String containing the database login associated with the current user.

The database login is not the same as the user's ClearQuest login. The database login refers to the account name ClearQuest uses when initiating transactions with the database. This value is set up in advance by the database administrator.

The user must be logged in to a database for this method to return an appropriate value. For hook code writers, ClearQuest logs the user in to the database automatically. If you are writing a standalone application, you must manually create a Session object and call the UserLogon method before calling this method.

For most users, this method returns the Read/Write login associated with the database. However, if the user associated with the current session is the ClearQuest administrator, this method returns the database-owner login instead. Similarly, if the user has a read-only account, this method returns the read-only login.

If you have access to the schema repository, you can retrieve information about this user database by accessing the properties of the corresponding Database object.

Example (in VBScript):

The following example shows you how to logon to the database from a Visual Basic application.

```
set sessionObj = CreateObject("CLEARQUEST.SESSION")

' Login to each database successively.
set databases = sessionObj.GetAccessibleDatabases
For Each db in databases
   If Not db.GetIsMaster Then
         ' Logon to the database.
         sessionObj.UserLogon "tom", "gh36ak3", dbName,
               AD_PRIVATE_SESSION, dbSetName
         ' Get the database login and password for "tom"
         dbLogin = db.GetLogin
         dbPassword = db.GetPassword
```

```
        ...
    End If
Next
```

Example (in Perl):

```
#Start a ClearQuest session
$sessionObj = CQPerlExt::CQSession_Build();

#Get a list of accessible database description objects
@databases = $sessionObj->GetAccessibleDatabases("ClearQuest
    2.0", "", "");

#Foreach accessible database that is not the master database
foreach $db ( @databases ) {
    if ( ! $db->GetIsMaster() ) {
        #Logon to the database as "tom" with password "gh36ak3"
        $sessionObj->UserLogon( "tom", "gh36ak3", $dbName,
                AD_PRIVATE_SESSION, $dbSetName );

        #Get the database login and password for "tom"
        $dbLogin = $db->GetLogin();
        $dbPassword = $db->GetPassword();
        ...
    }
}
```

See Also:

DBOLogin Property of the Database object
ROLogin Property of the Database object
RWLogin Property of the Database object
UserLogon Method of the Session object
Database Object
Session Object
Notation Conventions for VBScript
Notation Conventions for Perl

DatabaseDescs Object

The DatabaseDescs Object is a collection of **DatabaseDesc Object**s.

You can get the number of items in the collection by accessing the value in the Count Property. Use the Item Method to retrieve items from the collection.

Note: DatabaseDescs Objects and its property and methods are only applicable for usage with Perl script.

See Also:

DatabaseDesc Object

DatabaseDescs Object Property

The following list summarizes the DatabaseDescs Object property:

Property name	Description
Count Property	Returns the number of items in the collection.

Count Property

Description:

Returns the number of items in the collection. This property is read-only.

Perl Syntax:

```
$collection->Count();
```

Identifier	Description
collection	A DatabaseDescs collection object.
Data Type	A Long indicating the number of items in the collection object. This collection always contains at least one item.

See Also:

Item Method
Add Method

DatabaseDescs Object Methods

The following list summarizes the DatabaseDescs Object methods:

Method name	Description
Add Method	Adds an Attachment object to the collection.
Item Method	Returns the specified item in the collection.

Add Method

Description:

Adds a DatabaseDesc object to this DatabaseDescs collection.

The new DatabaseDesc object is added to the end of the collection. You can retrieve items from the collection using the **Item Method**.

Perl Syntax:

```
$databasedescs->Add(databasedesc);
```

Identifier	Description
databasedescs	A DatabaseDescs collection object.
databasedesc	The DatabaseDesc object to add to this collection
Return value	A Boolean that is True if the DatabaseDesc object was added successfully, otherwise False.

See Also:

Count Property
Item Method

Item Method

Description:

Returns the specified item in the DatabaseDescs collection.

The argument to this method can be either a numeric index (itemNum) or a String (name).

Perl Syntax:

```
$databasedescs->Item(itemNum);
$databasedescs->ItemByName(name);
```

Identifier	Description
databasedescs	A DatabaseDescs collection object.
itemNum	A Long that serves as an index into the collection. This index is 0-based so the first item in the collection is numbered 0, not 1.
name	A String that serves as a key into the collection. This string corresponds to the **GetDatabaseName Method** of the desired DatabaseDescs.
Return value	The DatabaseDesc object at the specified location in the collection.

See Also:

Count Property
Add Method

Databases Object

A Databases object is a collection object for Database objects.

You can get the number of items in the collection by accessing the value in the **Count Property**. Use the **Item Method** to retrieve items from the collection.

See Also:

Database Object

Databases Object Property

The following list summarizes the Databases Object property:

Property name	Access	Description
Count Property	Read-only	Returns the number of items in the collection.

Count Property

Returns the number of items in the collection.

This is a read-only property; it can be viewed but not set.

VBScript Syntax:

db_collection.**Count**

Perl Syntax:

$db_collection->**Count();**

Identifier	Description
collection	A Databases collection object, representing the set of databases associated with the current master database.
Data Type	A Long indicating the number of items in the collection object. This method returns zero if the collection contains no items.

See Also:

Item Method

Databases Object Method

The following list summarizes the Databases Object method:

Method name	Description
Item Method	Returns the item at the specified index in the collection.

Item Method

Returns the specified item in the collection.

The argument to this method can be either a numeric index (*itemNum*) or a String (*name*).

VBScript Syntax:

db_collection.**Item** *itemNum*
collection.**Item** *name*

Perl Syntax:

*$db_collection->*__Item__(*itemNum);*
*$db_collection->*__ItemByName__(*name);*

Identifier	Description
db_collection	A Databases collection object, representing the set of databases associated with the current master database.
itemNum	A Long that serves as an index into the collection. This index is 0-based so the first item in the collection is numbered 0, not 1.
name	A String that serves as a key into the collection. This string corresponds to the unique key of the desired Database object.
Data Type	The Database object at the specified location in the collection.

See Also:

Count Property

Entity Object

An Entity object represents a record in the database.

Entity objects are some of the most important objects in ClearQuest. They represent the data records the user creates, modifies, and views using ClearQuest. ClearQuest uses a single Entity object to store the data from a single database record. All of the data associated with that record is stored in the Entity object. When you want to view a field of a record, you use the methods of Entity to request the information.

The structure of an Entity object is derived from a corresponding **EntityDef Object** (record type). The EntityDef object contains metadata that defines the generic properties for a single type of Entity object. EntityDef objects can be state-based or stateless.

Accessing the fields of a record

Entity objects contain all of the data associated with the fields of a record. When you need to know something about a field, you always start with the Entity object. In some cases, you can call methods of Entity to get the information you need. However, you can also use the Entity object to acquire a **FieldInfo Object**, which contains additional information about the field.

To acquire a FieldInfo object, call the **GetFieldValue Method**.

To get the value stored in the FieldInfo object, call the **GetValue Method** of the FieldInfo object.

To acquire a collection of FieldInfo objects, one for each field in the record, call the **GetAllFieldValues Method**. (Note that GetAllFieldValues does not return the values in attachment fields.)

To get a list of the names of all fields, call the **GetFieldNames Method**.

To get the type of data stored in the field, call the **GetFieldType Method**.

To find out the field's behavior for the current action (mandatory, optional, or read-only), call the **GetFieldRequiredness Method**.

Although you would normally use a FieldInfo object to access a field, there are situations where you must use methods of Entity.

To set the value of a field, call the **SetFieldValue Method**.

To compare the new value with the old value of a field (if you previously updated the contents of a field), get the old value by calling the **GetFieldOriginalValue Method**.

Note: Although you can get the behavior of a field using either an Entity object or FieldInfo object, you can only use the **SetFieldRequiredness ForCurrentAction Method** of Entity to set the field's behavior.

To modify fields that contain choice lists, use the methods of the *Entity Object*on page 141.

Task	Entity object method to call
To retrieve the list of permissible values in the field	**GetFieldChoiceList Method**
To get a constant indicating whether or not you can add additional items to the choice list.	**GetFieldChoiceType Method**
To add items to a choice list that can be modified	**AddFieldValue Method**
To delete items from a choice list that can be modified	**DeleteFieldValue Method**

As you update the fields of a record, the Entity object gives you several ways to keep track of all the modified fields. Because hooks can be written to modify other fields, calling the SetFieldValue method might result in more than one field being changed. For example, suppose you call SetFieldValue for Field X, and a field hook in Field X changes the value of Field Y.

Note: You should be careful to avoid creating an infinite loop (hooks that call each other).

- To discover which fields were updated in the most recent call to SetFieldValue, call the **GetFieldsUpdatedThisSetValue Method**.
- To discover which fields have been updated since the beginning of the current action, call the **GetFieldsUpdatedThisAction Method**.
- To track changes during a specific period of code, surround calls to SetFieldValue with the **BeginNewFieldUpdateGroup Method** and **GetFieldsUpdatedThisGroup Method**.

Committing entity objects to the database

Committing an entity object to the database is a two step process:

1. Validate the record you changed.
2. Commit the change.

Note: In the context of a hook, you do not have to commit modifications to the current record. However, if you are writing an external application and want to retain the changes you made to a record, you must commit those changes to the database yourself.

To validate a record, call the **Validate Method** of the corresponding Entity object. This method runs the schema's validation scripts and returns a string containing any validation errors. If this string is not empty, you can use the **GetInvalidFieldValues Method** to return a list of fields that contain invalid data. After fixing the values in these fields, you must call Validate again. If the Validate method returns an empty string, there are no more errors.

After you validate the record, and the validation succeeds, you commit your changes to the database by calling the **Commit Method** of the corresponding Entity object. When you call the Commit method, ClearQuest writes the changes to the database and calls the action's commit hook. If the commit succeeds, ClearQuest launches the action's notification hook.

Note: For information about the order in which hooks fire, see "Execution order of field and action hooks" in the "Using hooks to customize your workflow" chapter of Administering ClearQuest.

If you decide that you do not want to commit your changes to the database, you can revert those changes by calling the Revert method of the Entity object. Reverting a set of changes returns the record to the state it was in before you called **EditEntity Method**. If you revert the changes made to an Entity object created by the **BuildEntity Method**, the record is discarded altogether.

Note: ClearQuest does not recycle the visible IDs associated with records. If you revert a record that was made editable by the BuildEntity method, the record is discarded but its visible ID is not so that future records cannot use that ID.

Working with duplicates

A duplicate record is one whose contents are essentially the same as another record. For example, two different users might file defect reports for the same problem, not knowing the other had filed a similar report. Rather than consolidate the defect information and delete one of the records, ClearQuest allows you to link the records. In your code, you might want to know if there are any records related to the current one so that you can notify the user that additional information is available.

Finding duplicate records and the original record

You can use the methods of Entity to find the duplicates of a record or find the records of which the current record is a duplicate. To determine if a record has one or more duplicates, call the **HasDuplicates Method** of Entity. To determine if the current record is itself a duplicate, call the **IsDuplicate Method**.

Finding duplicate objects and the original object

To get the duplicates of an object, you can use either the GetAllDuplicates method or the GetDuplicates method. These methods follow the links associated with the Entity object and return a list of the duplicates associated with it. The GetAllDuplicates method returns not only the duplicates of the object, but also any duplicates of duplicates, and so on. The GetDuplicates method returns only the immediate duplicates of the Entity object.

To discover whether the current Entity object is the parent of the duplicates, call the **IsOriginal Method**. (You can also call the **GetOriginalID Method** to return the object's visible ID instead of the object itself.)

To find out which Entity object is the parent of a group of duplicates, call the **IsOriginal Method** of each object until one of them returns True.

Entities and Hooks

Inside a VBScript hook, ClearQuest supplies an implicit Entity object representing the current data record. If your VBScript hook calls a method of Entity without supplying a leading identifier, ClearQuest automatically uses this implicit Entity object. In addition, ClearQuest hooks define an explicit "entity" variable to use if you want to specify the object to which you are referring. The entity variable name is identical to the record type name. If you are accessing the API from outside of a hook, or if you are accessing an

Entity object other than the implicit one, you must specify the other Entity object explicitly. (Also, if you are using Perl, you must always supply an explicit variable, and its name is "entity". See **GetSession Method** for details.)

The following examples show two ways to call the same method in a VBScript hook. In the second example, the value, `defect`, represents the current *entity* (record type) object.

```
fieldvalue = GetFieldValue("fieldname").GetValue()
```

or

```
fieldvalue = defect.GetFieldValue("fieldname").GetValue()
```

The Session object provides two methods to get an entity: **BuildEntity Method** (to build a new record) or **GetEntity Method** (for an existing record). When you submit a new record, BuildEntity automatically gets the entity. To get an existing record, you pass the GetEntity method the unique identifier of the record and the record type name.

You identify Entity objects using the display name of the corresponding record type. For stateless record types, you identify individual records using the contents of the unique key field of the record type. For state-based record types, you identify records using the record's visible ID. ClearQuest assigns each new record a visible ID string composed of the logical database name and a unique, sequential number. For example, the tenth record in the database "BUGID" can have the visible ID "BUGID00000010".

The following VBScript example is from a hook that accesses two Entity objects: the implicit object, and a duplicate object. The duplicate object corresponds to the record whose ID is "BUGID00000031".

```
set sessionObj = GetSession
' Call a method of the implicit Entity object.
set fieldvalue = GetFieldValue("fieldname")
' VBScript assumes the current entity implicitly.
' The fieldname must be valid or ClearQuest returns an error.
value = fieldvalue.GetValue()
' Call the same method for the duplicate object, by explicitly
acquiring
' the other entity, which is of the defect record type.
set otherEntity = sessionObj.GetEntity("defect", "BUGID00000031")
set fieldvalue2 = otherEntity.GetFieldValue("fieldname")
value = fieldvalue2.GetValue()
```

As demonstrated in the preceding example, to access an Entity object other than the implicit one from a VBScript hook, you must first acquire that Entity object. From outside of a hook, you must always acquire the Entity object you are going to work with.

Note: To learn more about acquiring existing Entity objects, see **Working with Queries** or the methods of the current **Session Object**.

See Also:

BuildEntity Method of the **Session Object**
EditEntity Method of the **Session Object**
GetEntity Method of the **Session Object**
GetEntityByDbId Method of the Session Object
EntityDef Object
QueryDef Object
ResultSet Object

Entity Object Properties

The following list summarizes the Entity Object properties:

Property name	Access	Description
AttachmentFields Property	Read-only	Returns the AttachmentFields collection object containing this Entity object's attachment fields.
HistoryFields Property	Read-only	Returns the HistoryFields collection object containing this Entity object's history fields.

AttachmentFields Property

Description:

Returns the AttachmentFields collection object containing this Entity object's attachment fields.

The AttachmentFields property is read-only; you cannot modify this field programmatically. However, once you retrieve an individual AttachmentField object, you can update its Attachments collection. In other words, within a field you can add or remove individual Attachment objects, but you cannot modify the field itself (or the collection of fields).

For an overview of attachments, see **Attachment Objects**.

VBScript Syntax:

```
[entity.]AttachmentFields
```

Perl Syntax:

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

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.
Return Value	An **AttachmentFields Object** that contains all of the **AttachmentField Object**s currently associated with this Entity object.

Example (VBScript):

```
set fields = entity.AttachmentFields
For Each fieldObj in fields
  ' Do something with each AttachmentField object

  ...
Next
```

Example (in Perl):

```
@fields = $entity->GetAttachmentFields();
Foreach $fieldobj (@fields)
  {
  ' Do something with each AttachmentField object
  # ...
```

See Also:

Attachment Object
AttachmentField Object
AttachmentFields Object
Attachments Object
Getting and Setting Attachment Information

HistoryFields Property

Description:

Returns the HistoryFieldscollection object containing this Entity object's history fields.

This property is read-only; you cannot modify this field programmatically. For an overview of history objects, see **History Object**.

VBScript Syntax:

```
[entity.]HistoryFields
```

Perl Syntax:

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

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.
Return Value	A **HistoryFields Object** that contains all the individual HistoryField objects currently associated with this Entity object.

Example (in VBScript):

```
set fields = entity.HistoryFields
For Each fieldObj in fields
  ' Look at the contents of the HistoryField object

  ...
Next
```

Example (in Perl):

```
@fields = $entity->GetHistoryFields();

foreach $fieldobj (@fields)
 {
 # Look at the contents of the HistoryField object
 # ...
 }
```

See Also:

Histories Object
History Object

HistoryField Object
HistoryFields Object

Entity Object Methods

The following list summarizes the Entity Object methods:

Method name	Description
AddFieldValue Method	Adds the specified value to the list of values in the named field.
BeginNewFieldUpdateGroup Method	Marks the beginning of a series of SetFieldValue calls.
Commit Method	Updates the database with the changes made to the Entity object.
DeleteFieldValue Method	Removes the specified value from the field's list of values.
FireNamedHook Method	Executes a named hook of this record's **EntityDef Object**.
GetActionName Method	Returns the name of the action associated with the current Entity object.
GetActionType Method	Returns the type of the action associated with the current Entity object.
GetAllDuplicates Method	Returns links to all of the duplicates of this Entity, including duplicates of duplicates.
GetAllFieldValues Method	Returns an array of FieldInfo objects corresponding to all of the Entity object's fields.
GetDbId Method	Returns the Entity object's database ID number.
GetDefaultActionName Method	Returns the default action associated with the current state.
GetDisplayName Method	Returns the unique key associated with the Entity.
GetDuplicates Method	Returns links to the immediate duplicates of this object.
GetEntityDefName Method	Returns the name of the EntityDef object that serves as a template for this object.
GetFieldChoiceList Method	Returns the list of permissible values for the specified field.

Method name	Description
GetFieldChoiceType Method	Returns the type of the given choice-list field.
GetFieldMaxLength Method	Returns the maximum number of characters allowed for the specified string field.
GetFieldNames Method	Returns the names of the fields in the Entity object.
GetFieldOriginalValue Method	Returns the FieldInfo containing the value that the specified field will revert to, if the action is cancelled.
GetFieldRequiredness Method	Identifies the **behavior** of the specified field.
GetFieldsUpdatedThisAction Method	Returns a FieldInfo object for each field that was modified by the most recent action.
GetFieldsUpdatedThisGroup Method	Returns a FieldInfo object for each field that was modified since the most recent call to **BeginNewFieldUpdateGroup Method**.
GetFieldsUpdatedThisSetValue Method	Returns a FieldInfo object for each of the Entity's fields that was modified by the most recent SetFieldValue call.
GetFieldType Method	Identifies the type of data that can be stored in the specified field.
GetFieldValue Method	Returns the FieldInfo object for the specified field.
GetInvalidFieldValues Method	Returns an array of FieldInfo objects corresponding to all the Entity's invalid fields.
GetLegalActionDefNames Method	Returns the names of the actions that can be used on this Entity object.
GetOriginal Method	Returns the Entity object that is marked as the **original** of this duplicate object.
GetOriginalID Method	Returns the visible ID of this object's original Entity object.
GetSession Method	Returns the current **Session Object**.
GetType Method	Returns the type (state-based or stateless) of the Entity.
HasDuplicates Method	Reports whether this object is the original of one or more duplicates.

Method name	Description
InvalidateFieldChoiceList Method	Use with **SetFieldChoiceList Method** to refresh values in a choice list.
IsDuplicate Method	Indicates whether this Entity object has been marked as a **duplicate** of another Entity object.
IsEditable Method	Returns True if the Entity object can be modified at this time.
IsOriginal Method	Returns True if this Entity has duplicates but is not itself a duplicate.
LookupStateName Method	Returns the name of the Entity object's current state.
Revert Method	Discards any changes made to the Entity object.
SetFieldChoiceList Method	Use with **InvalidateFieldChoiceList Method** to reset choice list values.
SetFieldRequirednessForCurrentAction Method	Sets the behavior of a field for the duration of the current action.
SetFieldValue Method	Places the specified value in the named field.
Validate Method	Validates the Entity object and reports any errors.

AddFieldValue Method

Description:

Adds the specified value to the list of values in the named field.

This method is similar to the **SetFieldValue Method**, except that it adds an item to a list of values, instead of providing the sole value. This method is intended for fields that can accept a list of values. If a field does not already contain a value, you can still use this method to set the value of a field that takes a single value.

To determine whether a field contains a valid value, obtain the **FieldInfo Object** for that field and call the **ValidityChangedThisSetValue Method** of the FieldInfo object to validate the field.

You can call this method only if the Entity object is editable. To make an existing Entity object editable, call the **EditEntity Method** of the Session object.

VBScript Syntax:

```
[entity.]AddFieldValue field_name, new_value
```

Perl Syntax:

```
$entity->AddFieldValue(field_name, new_value);
```

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.
field_name	A String containing a valid field name of this Entity object.
new_value	For VB, a variant containing the new value to add to the field. For Perl, a string containing the new value.
Return Value	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.

Example (in VBScript):

```
AddFieldValue "field1", "option 1"
AddFieldValue "field1", "option 2"
AddFieldValue "field1", "option 3"
```

Example (in Perl):

```
$entity->AddFieldValue("field1", "option 1");
$entity->AddFieldValue("field1", "option 2");
$entity->AddFieldValue("field1", "option 3");
```

See Also:

DeleteFieldValue Method
GetFieldValue Method
SetFieldValue Method
ValidityChangedThisSetValue Method in the **FieldInfo Object**
ValueChangedThisSetValue Method in the **FieldInfo Object**
EditEntity Method of the **Session Object**

BeginNewFieldUpdateGroup Method

Description:

Marks the beginning of a series of SetFieldValue calls.

You can use this method to mark the beginning of a group of calls to **SetFieldValue Method** You can later call the **GetFieldsUpdatedThisGroup Method** to track which fields were updated. This technique is useful for web-based systems where you might need to track any changes to the fields in a form. For example, if the user moves to another web page, you can call the GetFieldsUpdatedThisGroup method to save the current state of the form and restore it when the user returns to that page.

VBScript Syntax:

[*entity.*]**BeginNewFieldUpdateGroup**

Perl Syntax:

$*entity*->**BeginNewFieldUpdateGroup**(*);*

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.

Example (in VBScript):

```
BeginNewFieldUpdateGroup
SetFieldValue "field1", "1"
SetFieldValue "field2", "submitted"
SetFieldValue "field3", "done"
updatedFields = GetFieldsUpdatedThisGroup

' Iterate over all the fields that changed
For Each field In updatedFields
  ...
Next
```

See Also:

GetFieldsUpdatedThisAction Method
GetFieldsUpdatedThisGroup Method
GetFieldsUpdatedThisSetValue Method
SetFieldValue Method
ValidityChangedThisSetValue Method of the **FieldInfo Object**

Commit Method

Description:

Updates the database with the changes made to the Entity object.

This method commits any changes to the database. Before calling this method, you must validate any changes you made to the Entity object by calling the **Validate Method**. The application can call the Commit method only if the Validate method returns an empty string. After calling this method, the Entity object is no longer editable.

You can call this method only if the Entity object is editable. To make an existing Entity object editable, call the **EditEntity Method** of the Session object.

VBScript Syntax:

```
[entity.]Commit
```

Perl Syntax:

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

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.

Example (in VBScript):

```
' Modify the record and then commit the changes.
set sessionObj = GetSession
```

```
set entityObj = sessionObj.GetEntity("defect", "BUGID00000042")
sessionObj.EditEntity entityObj, "modify"

... ' modify the Entity object

entityObj.Validate
entityObj.Commit

' The Entity object is no longer editable
```

Example (in Perl):

```
# Modify the record and then commit the changes.

$sessionObj = $entity->GetSession();
$entityObj = $sessionobj->GetEntity("defect",BUGID00000042");
$sessionObj->EditEntity($entityobj,"modify");

# ... Modify the entity object

$entityObj->Validate();
$entityObj->Commit();

# The entity object is no longer editable.
```

See Also:

IsEditable Method
Revert Method
Validate Method
BuildEntity Method of the **Session Object**
EditEntity Method of the **Session Object**
Updating Duplicate Records to Match the Parent Record

DeleteFieldValue Method

Description:

Removes the specified value from the field's list of values.

This method is intended only for those fields that can support a list of values. However, it is legal to use this method for a field that takes a single value. (In that case, the field's

only value must be the same as old_value; the method then sets the field's value to the empty value.)

You can call this method only if the Entity object is editable. To make an existing Entity object editable, call the **EditEntity Method** of the Session object.

VBScript Syntax:

```
[entity.]DeleteFieldValue field_name, old_value
```

Perl Syntax:

```
$entity->DeleteFieldValue(field_name, value);
```

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 containing a valid field name of this Entity object.
old_value	A Variant containing the value to remove from the field's list of values.
value	A string containing the value to remove from the field's list of values.
Return Value	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.

Example (in VBScript):

```
AddFieldValue "field1", "option 1"
AddFieldValue "field1", "option 2"
AddFieldValue "field1", "option 3"
DeleteFieldValue "field1", "option 2"
DeleteFieldValue "field1", "option 3"
```

Example (in Perl):

```
$entity->AddFieldValue("field1", "option 1");
$entity->AddFieldValue("field1", "option 2");
$entity->AddFieldValue("field1", "option 3");
```

```
$entity->DeleteFieldValue("field1", "option 2");
$entity->DeleteFieldValue("field1", "option 3");
```

See Also:

AddFieldValue Method
GetFieldValue Method
SetFieldValue Method
ValidityChangedThisSetValue Method in the **FieldInfo Object**
ValueChangedThisSetValue Method in the **FieldInfo Object**
EditEntity Method of the **Session Object**

FireNamedHook Method

Description:

Executes a named hook of this record's **EntityDef Object**.

You can use this method to execute a record hook at runtime. Record hooks are routines you define and are specific to a particular record type. You can use record hooks in conjunction with form controls or you can call them from other hooks. You define record hooks using ClearQuest Designer. The syntax for record hooks is as follows:

```
Function EntityDefName_HookName(parameters)
  ' parameter as Variant
  ' EntityDefName_HookName as Variant

  ' Hook program body
End Function
```

You cannot use this method to execute a field or action hook of a record. You also cannot execute a global hook, except indirectly from the record hook.

You can call this method on an Entity object regardless of whether or not it is editable. However, if your hook attempts to modify the Entity object, either your code or the hook code must first call **EditEntity Method** to make the Entity object editable.

If your hook accepts any parameters, put all of the parameters in a single Variant and specify that Variant in parameters. The hook must be able to interpret the parameters passed into it. Upon return, the hook can similarly return a Variant with any appropriate return values.

VBScript Syntax:

```
[entity.]FireNamedHook hookName, parameters
```

Perl Syntax:

```
$entity->FireNamedHook(hookName, parameters);
```

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).
hookName	A String containing the name of the hook to execute.
parameter(s)	[A VB Variant or] a Perl string containing the parameters you want to pass to the hook.
Return Value	A String indicating the status of calling the hook. If the hook executed successfully, this method returns an empty string (""), otherwise the returned string contains a description of the error.

Example (in VBScript):

```
' Execute the hook "MyHook" with the specified parameters
Dim params(1)
params(0) = "option 1"
params(1) = "option 2"

returnValue = entity.FireNamedHook("MyHook", params)
```

Example (in Perl):

```
# Execute the hook "MyHook" with the specified parameters

@params[0] = "option 1";
@params[1] = "option 2";

$returnValue = $entity->FireNamedHook("MyHook",@params);
```

See Also:

EditEntity Method of the **Session Object**
GetHookDefNames Method of the **EntityDef Object**

GetActionName Method

Description:

Returns the name of the current action associated with the current entity. Used in base action hooks.

VBScript Syntax:

entity.**GetActionName**

Perl Syntax:

$*entity*->**GetActionName**(*);*

Identifier	Description
entity	An Entity object corresponding to a record in a schema.
Return Value	A String whose value provides the name of **ActionType Constants _GETACTIONNAME**.

See Also:

GetActionType Method
ActionType Constants

GetActionType Method

Description:

Returns the type of the current action associated with the current entity. Used in base action hooks.

VBScript Syntax:

entitydef.**GetActionType**

Perl Syntax:

$*entity*->**GetActionType**(*);*

Identifier	Description
entity	An Entity object corresponding to a record in a schema.
Return Value	A String whose value is the **ActionType Constants _GETACTIONNAME**.

See Also:

GetActionName Method
ActionType Constants

GetAllDuplicates Method

Description:

Returns links to all of the duplicates of this Entity, including duplicates of duplicates.

This method returns all duplicates, including duplicates of duplicates. To obtain only the immediate duplicates of an object, call the **GetDuplicates Method** instead.

VBScript Syntax:

```
[entity.]GetAllDuplicates
```

Perl Syntax:

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

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	For VB, a Variant containing an Array of **Link Object**s is returned. If this object has no duplicates, the return value is an Empty Variant. For Perl, a **Links Object** collection is returned.

Example (in VBScript):

In the following example, *entity1* is the original object. The objects *entity2* and *entity3* are duplicates of *entity1*. In addition, the object *entity4* is a duplicate of *entity3*. Given the following statement:

```
linkObjs = entity1.GetAllDuplicates
```

The *linkObjs* variable would be an array of 3 **Link Object**s:

- a link between entity1 and entity2
- a link between entity1 and entity3
- a link between entity3 and entity4

Example (in Perl):

```
$linkobjs = $entity1->GetAllDuplicates();
```

See Also:

GetDuplicates Method
GetOriginal Method
GetOriginalID Method
HasDuplicates Method
IsDuplicate Method
IsOriginal Method
MarkEntityAsDuplicate Method of the **Session Object**
UnmarkEntityAsDuplicate Method of the **Session Object**
Link Object
Updating Duplicate Records to Match the Parent Record

GetAllFieldValues Method

Description:

Returns an array of FieldInfo objects corresponding to all of the Entity object's fields. The FieldInfo objects are arranged in no particular order.

VBScript Syntax:

[*entity.*]**GetAllFieldValue**

Perl Syntax:

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

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	For VB, a Variant containing an Array of **FieldInfo Object**s is returned, one for each field in the Entity object. For Perl, a **FieldInfos Object** collection is returned.

Example (in VBScript):

```
' Iterate through the fields and examine the field names and
values
fieldObjs = GetAllFieldValues
For Each field In fieldObjs
  fieldValue = field.GetValue
  fieldName = field.GetName
  ...
Next
```

Example (in Perl):

```
# Iterate through the fields and examine the field names and
# values

$count = $fieldobjs->count();

for ($i = 0; $i < $count; $i++)
  {
  $field = $fieldobjs -> Item($i);
   $fieldvalue = $field->GetValue();
 $fieldname = $field->GetName();
 # ... other field commands
 }
```

See Also:

GetFieldValue Method
GetInvalidFieldValues Method
FieldInfo Object

GetDbId Method

Description:

Returns the Entity object's database ID number.

The return value is a database ID. This value is used internally by the database to keep track of records. Do not confuse this value with the defect ID number returned by the **GetDisplayName Method**.

VBScript Syntax:

[*entity.*]**GetDbId**

Perl Syntax:

$*entity*->**GetDbId**(*);*

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	A Long containing the Entity object's database ID.

Example (in VBScript):

```
dbID = entity.GetDbId
```

Example (in Perl):

```
$dbid = $entity->GetDbId();
```

See Also:

GetDisplayName Method
Extracting Data About an EntityDef (Record Type)

GetDefaultActionName Method

Description:

Returns the default action name associated with the current state.

This method allows you to get the default action for the current record.

Whereas this method returns the default action name associated with the current state, **GetActionDestStateName Method** returns the destination state name associated with the current action.

VBScript Syntax:

```
[entity.]GetDefaultActionName
```

Perl Syntax:

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

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	A String that returns the default action name associated with the current state.

Example (in VBScript):

```
DefaultActionName = entity.GetDefaultActionName
```

Example (in Perl):

```
$defaultactionname = $entity->GetDefaultActionName();
```

See Also:

GetActionDestStateName Method

GetDisplayName Method

Description:

Returns the unique key associated with the Entity.

For state-based record types, the unique key is the record's visible ID, which has the format SITEnnnnnn (for example, 'PASNY00012332'), where SITE is an indication of the installation site and nnnnnn is the defect (bug) number.

For stateless record types, the unique key is formed from the values of the unique key fields defined by the administrator. If there is just a single unique key field, its value will be the unique key. If there are multiple fields forming the unique key, their values will be concatenated in the order specified by the administrator. For state-based record types, calling this method is equivalent to getting the value of the "id" system field using a **FieldInfo Object**.

The unique key should not be confused with the database ID, which is invisible to the user. The database ID is retrieved by the **GetDbId Method**.

VBScript Syntax:

```
[entity.]GetDisplayName
```

Perl Syntax:

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

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	A String containing the record type's unique key.

Example (in VBScript):

```
' Get the record ID using 2 different techniques and compare the
' results
displayName = GetDisplayName
idName = GetFieldValue("id").GetValue
If idName <> displayName Then
  ' Error, these id numbers should match
End If
```

Example (in Perl):

```
# Get the record ID using 2 different techniques and compare the #
results

$displayname = $entity->GetDisplayName();
$idname = $entity->GetFieldValue("id")->GetValue;

if ($idname ne $displayname)
 {
 # error, these id numbers should match
 }
```

See Also:

GetDbId Method
GetFieldValue Method
GetValue Method of the FieldInfo Object
Updating Duplicate Records to Match the Parent Record

GetDuplicates Method

Description:

Returns links to the immediate duplicates of this object.

This method returns only immediate duplicates; it does not return duplicates of duplicates. To return all of the duplicates for a given Entity object, including duplicates of duplicates, call the **GetAllDuplicates Method**.

VBScript Syntax:

[*entity.*]**GetDuplicates**

Perl Syntax:

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

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	For VB, a Variant containing an Array of **Link Object**s is returned. Each Link object points to a duplicate of this object. If this object has no duplicates, the return value is an Empty Variant. For Perl, a **Links Object** collection is returned.

Example (in VBScript):

In the following example, *entity1* is the original object. The objects *entity2* and *entity3* are duplicates of *entity1*. In addition, the object *entity4* is a duplicate of *entity3*. Given the following statement:

```
linkObjs = entity1.GetDuplicates
```

The *linkObjs* variable would be an array of 2 **Link Object**s:

- a link between entity1 and entity2
- a link between entity1 and entity3

Example (in Perl):

```
$linkobjs = $entity1->GetAllDuplicates();
```

See Also:

GetAllDuplicates Method
GetOriginal Method
GetOriginalID Method
HasDuplicates Method
IsDuplicate Method
IsOriginal Method
MarkEntityAsDuplicate Method of the **Session Object**

UnmarkEntityAsDuplicate Method of the **Session Object**
Updating Duplicate Records to Match the Parent Record

GetEntityDefName Method

Description:

Returns the name of the EntityDef object that is the template for this object.

To get the corresponding EntityDef object, call the Session object's **GetEntityDef Method**.

Before using the methods of EntityDef object, you should look at the methods of Entity to see if one of them returns the information you need. Some of the more common information available in an EntityDef object can also be obtained directly from methods of Entity.

VBScript Syntax:

[*entity.*]**GetEntityDefName**

Perl Syntax:

$*entity*->**GetEntityDefName();**

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	A String containing the name of the EntityDef object upon which this object is based.

Example (in VBScript):

```
set sessionObj = GetSession

' Get the EntityDef of the record using GetEntityDefName
entityDefName = GetEntityDefName
set entityDefObj = sessionObj.GetEntityDef(entityDefName)
```

Example (in Perl):

```
$sessionobj = $entity->GetSession();

# Get the EntityDef of the record using GetEntityDefName

$entitydefname = $entity->GetEntityDefName();
$entitydefobj = $sessionobj->GetEntityDef($entitydefname);
```

See Also:

GetEntityDef Method
EntityDef Object

GetFieldChoiceList Method

Description:

Returns the list of permissible values for the specified field.

The administrator specifies whether the legal values for a given field are restricted to the contents of the choice list. If there is a restriction, specifying a value not in the choice list causes a validation error. If there is no restriction, you may specify values not in the choice list. (Note that any values you specify must still be validated.)

If this method returns an Empty Variant, it does not imply that all values are permitted; it just means that the administrator has not provided any hints about the values permitted in the field.

If the administrator chose to use a hook to determine the values of the choice list, ClearQuest will have already executed the hook and cached the resulting values in a **HookChoices object**. You can use that object to retrieve the values.

You can use the **GetFieldNames Method** to obtain a list of valid names for the field_name parameter.

Note: When calling this method from an external Visual Basic program, this method throws an exception if entity is not editable.

VBScript Syntax:

[*entity.*]**GetFieldChoiceList** *field_name*

Perl Syntax:

```
$entity->GetFieldChoiceList(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 entity.
Return Value	For VB, a Variant containing an Array is returned. Each element of the array contains an acceptable value for the specified field. If a list of choices was not provided with the field, the returned Variant is Empty. For Perl, a reference to a string array is returned.

Example (in VBScript):

```
fieldValue = GetFieldValue("field1").GetValue

' Check to see if the field's current value is in the choice list
fieldChoiceList = GetFieldChoiceList("field1")
For Each fieldChoice in fieldChoiceList
  If fieldValue = fieldChoice Then
      ' This is a valid choice
  End If
Next
```

Example (in Perl):

```
# If the field must have a value from a closed choice list, assign
# the first the value in the list to the field by default.

$choicetype = $entity->GetFieldChoiceType("field1");
if ($choicetype eq $CQPerlExt::CQ_CLOSED_CHOICE)
 {
 # Set the field to the first item in the choice list.
 $fieldchoicelist = $entity->GetFieldChoiceList("field1);
 $entity->SetFieldValue("field1",$fieldchoicelist->Item (0));
 }
```

See Also:

GetFieldChoiceType Method
GetFieldNames Method
HookChoices object

GetFieldChoiceType Method

Description:

Returns the type of the given choice-list field.

The return value is either CLOSED_CHOICE or OPEN_CHOICE. If the return value is CLOSED_CHOICE, the valid values for the field are limited to those specified in the choice list. If the return value is OPEN_CHOICE, the user may select an item from the choice list or type in a new value.

VBScript Syntax:

```
[entity.]GetFieldChoiceType field_name
```

Perl Syntax:

```
$entity->GetFieldChoiceType(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 entity.
Return Value	A Long indicating the type of the field.

Example (in VBScript):

```
' If the field must have a value from a closed choice list, assign
' the first the value in the list to the field by default.
choiceType = GetFieldChoiceType("field1")
If choiceType = AD_CLOSED_CHOICE Then
  ' Set the field to the first item in the choice list.
```

```
  fieldChoiceList = GetFieldChoiceList("field1")
  SetFieldValue "field1", fieldChoiceList(0)
End If
```

Example (in Perl):

```
# If the field must have a value from a closed choice list, assign
# the first the value in the list to the field by default.

$choicetype = $entity->GetFieldChoiceType("field1");
if ($choicetype eq $CQPerlExt::CQ_CLOSED_CHOICE)
 {
 # Set the field to the first item in the choice list.
 @fieldchoicelist = $entity->GetFieldChoiceList("field1");
 $entity->SetFieldValue("field1",@fieldchoicelist[0]);
 }
```

See Also:

GetFieldChoiceList Method
GetFieldNames Method
HookChoices object
Notation Conventions for Perl
Notation Conventions for VBScript

GetFieldMaxLength Method

Description:

Returns the maximum number of characters allowed for the specified string field.

This method is relevant only for fields whose type is SHORT_STRING.

VBScript Syntax:

[*entity.*]**GetFieldMaxLength** *field_name*

Perl Syntax:

$*entity*->**GetFieldMaxLength(***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 entity.The field must contain a fixed-length string.
Return Value	A Long indicating the maximum number of characters the field can store.

Example (in VBScript):

```
' Check the maximum length of a string field.
fieldType = GetFieldType("field1")
If fieldType = AD_SHORT_STRING Then
  maxLength = GetFieldMaxLength("field1")
End If
```

Example (in Perl):

```
# Check the maximum length of a string field.
$fieldtype = $entity->GetFieldType("field1");
if ($fieldtype eq $CQPerlExt::CQ_SHORT_STRING)
 {
 $maxlength = $entity->GetFieldMaxLength("field1");
 }
```

See Also:

GetFieldType Method
EventType Constants
Notation Conventions for VBScript
Notation Conventions for Perl

GetFieldNames Method

Description:

Returns the names of the fields in the Entity object.

The list of names is returned in no particular order and there is always at least one field. You must examine each entry in the array until you find the name of the field you are looking for.

VBScript Syntax:

[*entity.*]**GetFieldNames**

Perl Syntax:

$*entity->***GetFieldNames;**

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	For VB, a Variant containing an Array whose elements are Strings is returned. Each String contains the name of one field. For Perl, a reference to a string array is returned.

Example (in VBScript):

```
set sessionObj = GetSession

' Iterate through the fields and output
' the field name, type, and value
fieldNameList = GetFieldNames
For Each fieldName in fieldNameList
  set fieldInfoObj = GetFieldValue(fieldName)
  fieldType = fieldInfoObj.GetType
  fieldValue = fieldInfoObj.GetValue

  sessionObj.OutputDebugString "Field name: " & fieldName & _
     ", type="  & fieldType & ", value=" & fieldValue
Next
```

Example (in Perl):

```
$sessionobj = $entity->GetSession();

# Iterate through the fields and output
# the field name, type, and value
```

```
$fieldnamelist = $entity->GetFieldNames();

$count = $fieldnamelist->count();

for ($i = 0; $i < $count; $i++)
  {
  $fieldname = $fieldnamelist -> Item($i);
 $fieldinfoobj = $entity->GetFieldValue($fieldname);
 $fieldtype = $fieldinfoobj->GetType();
 $fieldvalue = $fieldinfoobj->GetValue();

 $sessionobj->OutputDebugString("Field name: ".$fieldname.",
type=".$fieldtype.", value=".$fieldvalue);
 }
```

See Also:

GetFieldChoiceList Method
GetFieldDefNames Method
GetFieldRequiredness Method
GetFieldType Method
GetFieldValue Method
Showing Changes to an Entity (Record)

GetFieldOriginalValue Method

Description:

Returns the FieldInfo containing the value that the specified field will revert to, if the action is cancelled.

When you initiate an action, ClearQuest caches the original values of the record's fields in case the action is cancelled. You can use this method to return the original value of a field that you have modified. You can get the original value of a field only while the record is editable. The record's notification hook is the last opportunity to get the original value before a new value takes effect.

VBScript Syntax:

[*entity.*]**GetFieldOriginalValue** *field_name*

Perl Syntax:

```
$entity->GetFieldOriginalValue(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 containing a valid field name of this Entity object.
Return Value	A FieldInfo object that contains the original value for the specified field.

Example (in VBScript):

```
' Iterate through the fields and report which ones have changed.
fieldNameList = GetFieldNames
For Each fieldName in fieldNameList
  originalValue = GetFieldOriginalValue(fieldName).GetValue
  currentValue = GetFieldValue(fieldName).GetValue
  If currentValue <> originalValue Then
     ' Report a change in the field value
     OutputDebugString "The value in field " & fieldName & " has
changed."
  End If
Next
```

See Also:

GetFieldValue Method
FieldInfo Object
Showing Changes to an Entity (Record)

GetFieldRequiredness Method

Description:

Identifies the **behavior** of the specified field.

A field can be mandatory, optional, or read-only. If entity is not an editable Entity object, this method always returns the value READONLY. If the Entity object is

editable, because an action has been initiated, the return value can be READONLY, MANDATORY, or OPTIONAL.

This method never returns the value USE_HOOK. If the behavior of the field is determined by a permission hook, ClearQuest will have already executed that hook and cached the resulting value. This method then returns the cached value.

Note: Because hooks operate with administrator privileges, they can always modify the contents of a field, regardless of its current behavior setting.

You can use the **GetFieldNames Method** to obtain a list of valid names for the field_name parameter.

VBScript Syntax:

[*entity.*]**GetFieldRequiredness** *field_name*

Perl Syntax:

$*entity->***GetFieldRequiredness(***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 entity.
Return Value	A Long that identifies the behavior of the named field. The value corresponds to one of the **Behavior Constants**.

Example (in VBScript):

```
' Change all mandatory fields to optional
fieldNameList = GetFieldNames
For Each fieldName in fieldNameList
  fieldReq = GetFieldRequiredness(fieldName)
     if fieldReq = AD_MANDATORY Then
        SetFieldRequirednessForCurrentAction fieldName,
            AD_OPTIONAL
     End If
Next
```

Example (in Perl):

```
# Change all mandatory fields to optional

@fieldnamelist = $entity->GetFieldNames();

foreach $fieldname (@fieldnamelist)
 {
 $fieldreq = $entity->GetFieldRequiredness($fieldname);
 if ($fieldreq = $CQPerlExt::CQ_MANDATORY)
  {
     $entity->SetFieldRequirednessForCurrentAction($fieldname,
         $CQPerlExt::CQ_OPTIONAL);
  }
 }
```

See Also:

GetFieldNames Method
GetRequiredness Method of the **FieldInfo Object**
Notation Conventions for Perl
Notation Conventions for VBScript

GetFieldsUpdatedThisAction Method

Description:

Returns a FieldInfo object for each field that was modified by the most recent action.

This method reports the fields that changed during the current action, that is, all fields that changed after the call to BuildEntity or EditEntity returned. Fields that were implicitly changed during the action's startup phase are not reported; fields that were modified by hooks during the initialization of the action are also not reported. This method does report fields that were changed by hooks after the initialization phase of the action; see the ClearQuest Designer documentation for the timing and execution order of hooks.

As an example, if the user initiates a CHANGE_STATE action, the value in the record's "state" field changes but is not reported by this method. Similarly, if the action-initialization hook of the action modifies a field, that change is not reported.

However, changes that occurred during a field value-changed hook or a validation hook are reported because they occur after the action is completely initialized.

VBScript Syntax:

*[entity.]***GetFieldsUpdatedThisAction** *field_name*

Perl Syntax:

$*entity*->**GetFieldsUpdatedThisAction();**

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	For VB, a Variant containing an Array of **FieldInfo Object**s is returned. Each FieldInfo object corresponds to a field of the Entity object whose value was changed since the most recent action was initiated. If no fields were updated, this method returns an Empty Variant. For Perl, a **FieldInfos Object** collection is returned.

Example (in VBScript):

```
set sessionObj = GetSession

' Report any fields that changed during the recent action
fieldList = GetFieldsUpdatedThisAction
For Each field in fieldList
     ' Report the field to the user
     sessionObj.OutputDebugString "Field " & field.GetName & "
        changed."
Next
```

Example (in Perl):

```
$sessionobj = $entity->GetSession();
# Report any fields that changed during the recent action
@fieldlist = $entity->GetFieldsUpdatedThisAction();

foreach $field (@fieldlist)
 {
 # Report the field to the user
 $sessionobj->OutputDebugString("Field ".$field->GetName."
      changed." )
 }
```

See Also:

BeginNewFieldUpdateGroup Method
GetFieldsUpdatedThisAction Method
GetFieldsUpdatedThisSetValue Method
SetFieldValue Method
ValidityChangedThisSetValue Method of the **FieldInfo Object**

GetFieldsUpdatedThisGroup Method

Description:

Returns a **FieldInfo Object** for each field that was modified since the most recent call to **BeginNewFieldUpdateGroup Method**.

Use this method to mark the end of a group of calls to **SetFieldValue Method** (You must have previously called **BeginNewFieldUpdateGroup Method** to mark the beginning of the group.) This technique is useful for web-based systems where you might need to track any changes to the fields in a form. For example, if the user moves to another web page, you can call this method to save the current state of the form and restore it when the user returns to that page.

VBScript Syntax:

[*entity.*]**GetFieldsUpdatedThisGroup**

Perl Syntax:

$*entity->***GetFieldsUpdatedThisGroup();**

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	For VB, a Variant containing an Array of **FieldInfo Object**s is returned. Each FieldInfo object corresponds to a field whose value changed since the most recent call to BeginNewFieldUpdateGroup. If no fields were updated, this method returns an Empty Variant. For Perl, a **FieldInfos Object** collection is returned.

Example (in VBScript):

```
BeginNewFieldUpdateGroup
SetFieldValue "field1", "1"
SetFieldValue "field2", "submitted"
SetFieldValue "field3", "done"
updatedFields = GetFieldsUpdatedThisGroup

' Iterate over all the fields that changed
For Each field In updatedFields
  ...
Next
```

Example (in Perl):

```
$entity->BeginNewFieldUpdateGroup()
$entity->SetFieldValue("field1", "1" );
$entity->SetFieldValue("field2", "submitted");
$entity->SetFieldValue("field3", "done");
$updatedFields = $entity->GetFieldsUpdatedThisGroup ();

# Iterate over all the fields that changed
$count = $updatedFields->count();

for ($i = 0; $i < $count; $i++)
  {
  $field = $updatedFields -> Item($i);
 # do other tasks...
 }
```

See Also:

BeginNewFieldUpdateGroup Method
GetFieldsUpdatedThisAction Method
GetFieldsUpdatedThisSetValue Method
SetFieldValue Method
ValidityChangedThisSetValue Method of the **FieldInfo Object**

GetFieldsUpdatedThisSetValue Method

Description:

Returns a FieldInfo object for each of the Entity's fields that was modified by the most recent SetFieldValue call.

This method usually returns a single FieldInfo object for the field that was modified by **SetFieldValue Method**. However, this method can return multiple FieldInfo objects if other fields are dependent on the field that was changed. In such a case, hook code could automatically modify the value of any dependent fields, causing them to be modified as well and thus reported by this method.

VBScript Syntax:

[*entity.*]**GetFieldsUpdatedThisSetValue**

Perl Syntax:

$*entity->***GetFieldsUpdatedThisSetValue();**

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	For VB, a Variant containing an Array of **FieldInfo Object**s is returned, one for each field in the Entity object whose value was changed by the most recent invocation of SetFieldValue. If no fields were modified, this method returns an Empty Variant. For Perl, a **FieldInfos Object** collection is returned.

Example (in VBScript):

```
SetFieldValue "field1" "100"
modifiedFields = GetFieldsUpdatedThisSetValue
numFields = UBound(modifiedFields) + 1
If numFields > 1 Then
  OutputDebugString "Changing field1 resulted in changes to " _
     & numFields & " other fields"
End If
```

Example (in Perl):

```
$entity->SetFieldValue("field1", "100");

@modifiedfields = $entity->GetFieldsUpdatedThisSetValue();
$numfields = $#modifiedfields + 1;

if ($numfields > 1)
 {
 $session->OutputDebugString("Changing field1 resulted in changes
   to ".$numfields." other fields");
 }
```

See Also:

BeginNewFieldUpdateGroup Method
GetFieldsUpdatedThisAction Method
GetFieldsUpdatedThisGroup Method
SetFieldValue Method
ValidityChangedThisSetValue Method in the **FieldInfo Object**
FieldInfo Object

GetFieldType Method

Description:

Identifies the type of data that can be stored in the specified field.

The **EntityDef Object** controls what type of data can be stored in each field of an Entity object. Fields can store strings, numbers, timestamps, references, and so on. (See **FieldType Constants** for the complete list.)

You cannot change the type of a field using the API. The field type is determined by the corresponding information in the EntityDef object and must be set by the administrator using ClearQuest Designer.

You can use the **GetFieldNames Method** to obtain a list of valid names for the field_name parameter.

VBScript Syntax:

[*entity.*]**GetFieldType** *field_name*

Perl Syntax:

$*entity*->**GetFieldType(***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 entity.
Return Value	A Long that identifies what type of data can be stored in the named field. The value corresponds to one of the **FieldType Constants**.

Example (in VBScript):

```
set sessionObj = GetSession

' Iterate through the fields and output
' the field name and type.
fieldNameList = GetFieldNames
For Each fieldName in fieldNameList
  fieldType = GetFieldType(fieldName)
  sessionObj.OutputDebugString "Field name: " & fieldName & _
        ", type=" & fieldType
Next
```

Example (in Perl):

```
$sessionobj = $entity->GetSession();

# Iterate through the fields and output
```

```
# the field name and type.

$fieldnamelist = $entity->GetFieldNames();

$count = $fieldnamelist->count();

for ($i = 0; $i < $count; $i++)
  {
  $field = $fieldnamelist -> Item($i);
 $fieldtype = $entity->GetFieldType($fieldname);
 $sessionobj->OutputDebugString("Field name: ".$fieldname. ",
         type=".$fieldtype);
 }
```

See Also:

GetFieldNames Method
GetType Method of the **FieldInfo Object**

GetFieldValue Method

Description:

Returns the FieldInfo object for the specified field.

This method returns a FieldInfo object from which you can obtain information about the field. This method does not return the actual value stored in the field. To retrieve the actual value (or values), call this method first and then call the FieldInfo object's **GetValue Method** or **GetValueAsList Method**.

VBScript Syntax:

[*entity.*]**GetFieldValue** *field_name*

Perl Syntax:

$*entity*->**GetFieldValue(***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 containing a valid field name of this Entity object.
Return Value	The FieldInfo object corresponding to the specified field.

Example (in VBScript):

```
set sessionObj = GetSession

' Iterate through the fields and output
' the field name and type.
fieldNameList = GetFieldNames
For Each fieldName in fieldNameList
  fieldValue = GetFieldValue(fieldName).GetValue
  sessionObj.OutputDebugString "Field name: " & fieldName & _
          ", value=" & fieldValue
Next
```

Example (in Perl):

```
$sessionobj = $entity->GetSession();

# Iterate through the fields and output
# the field name and type.

$fieldnamelist = $entity->GetFieldNames();

$count = $fieldnamelist->count();

for ($i = 0; $i < $count; $i++)
  {
  $field = $fieldnamelist -> Item($i);
 $fieldvalue = $entity->GetFieldValue($fieldname);
 $sessionobj->OutputDebugString("Field name: ".$fieldname. ",
          value=".$fieldvalue);
 }
```

See Also:

AddFieldValue Method
DeleteFieldValue Method
GetAllFieldValues Method
SetFieldValue Method
GetValue Method of the **FieldInfo Object**
GetValueAsList Method of the **FieldInfo Object**

GetInvalidFieldValues Method

Description:

Returns an array of FieldInfo objects corresponding to all the Entity's invalid fields.

The FieldInfo objects are arranged in no particular order. Use this method before committing a record to determine which fields contain invalid values, so that you can fix them.

VBScript Syntax:

[*entity.*]**GetInvalidFieldValues**

Perl Syntax:

$*entity->***GetInvalidFieldValues();**

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	For VB, a Variant containing an Array of **FieldInfo Object**s is returned. Each FieldInfo object corresponds to a field of the Entity object that contains an invalid value. If all of the fields are valid, this method returns an Empty Variant. For Perl, a **FieldInfos Object** collection is returned.

See Also:

GetAllFieldValues Method
GetFieldValue Method

Validate Method
ValidityChangedThisSetValue Method of the **FieldInfo Object**

GetLegalActionDefNames Method

Description:

Returns the names of the actions that can be used on this Entity object.

This method is similar to the **GetActionDefNames Method** of EntityDef; however, the list returned by this method contains only those actions that can be performed on the Entity object in its current state. You can use this method before calling the Session object's **EditEntity Method** to determine which actions you can legally perform on the record.

VBScript Syntax:

[*entity.*]**GetLegalActionDefNames**

Perl Syntax:

$*entity*->**GetLegalActionDefNames();**

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	For VB, a Variant containing an Array of Strings is returned. Each String contains the name of a legal action. If no actions can be performed on the Entity object, the return value is an Empty variant. For Perl, a reference to a string array is returned.

Example (in VBScript):

```
set sessionObj = GetSession

entityDefName = GetEntityDefName
set entityDefObj = sessionObj.GetEntityDef(entityDefName)

' Search for a legal action with which to modify the record
```

```
actionDefList = GetLegalActionDefNames
For Each actionDef in actionDefList
  actionDefType = entityDefObj.GetActionDefType(actionDef)
  if actionDefType = AD_MODIFY Then
      sessionObj.EditEntity(entity, actionDef)
      Exit For
  End If
Next
```

Example (in Perl):

```
$sessionobj = $entity->GetSession();

$entitydefname = $entity->GetEntityDefName();

$entitydefobj = $sessionobj->GetEntityDef($entitydefname);

# Search for a legal action with which to modify the record
$actiondeflist = $entity->GetLegalActionDefNames();

$count = $actiondeflist->count();

for ($i = 0; $i < $count; $i++)
  {
  $field = $actiondeflist -> Item($i);
 $actiondeftype = $entitydefobj->GetActionDefType($actiondef);
 if ($actiondedftype eq $CQPerlExt::CQ_MODIFY)
  {
  $sessionobj->EditEntity($entity,$actiondef);
  }
 }
```

See Also:

GetActionDefNames Method
EditEntity Method of the **Session Object**
Notation Conventions for Perl
Notation Conventions for VBScript

GetOriginal Method

Description:

Returns the Entity object that is marked as the parent of this duplicate object.

Use this method to get the Entity object that is the immediate parent of this object.

The returned object may itself be a duplicate of another Entity object. To find the true original, call the **IsDuplicate Method** of the returned object. If IsDuplicate returns True, call that object's **GetOriginal Method** to get the next Entity object in the chain. Continue calling the IsDuplicate and GetOriginal methods until IsDuplicate returns False, at which point you have the true original.

Note: It is an error to call this method for an Entity object that is not a duplicate. You should always call the IsDuplicate method first to verify that the object is a duplicate.

VBScript Syntax:

[*entity.*]**GetOriginal**

Perl Syntax:

$*entity*->**GetOriginal();**

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	The Entity object of which entity is a duplicate.

Example (in VBScript):

```
' Display a dialog box indicating which record is
' the original of this record
If entity.IsDuplicate Then
  ' Get the ID of this record
  duplicateID = entity.GetDisplayName

  ' Get the ID of the original record
  set originalObj = entity.GetOriginal
```

```
  originalID = originalObj.GetDisplayName
  OutputDebugString "The parent of record " & duplicateID & _
            " is record " & originalID
End If
```

Example (in Perl):

```
# Display a dialog box indicating which record is
# the original of this record

if ($entity->IsDuplicate())
 {
 # Get the ID of this record

 $duplicateID = $entity->GetDisplayName ();

 # Get the ID of the original record
 $originalObj =$entity->GetOriginal();
 $originalID = $originalObj->GetDisplayName();
 $session->OutputDebugString("The parent of record
         ".$duplicateID." is record ".$originalID);
 }
```

See Also:

GetAllDuplicates Method
GetDuplicates Method
GetOriginalID Method
HasDuplicates Method
IsDuplicate Method
IsOriginal Method
MarkEntityAsDuplicate Method of the **Session Object**
UnmarkEntityAsDuplicate Method of the **Session Object**
Updating Duplicate Records to Match the Parent Record

GetOriginalID Method

Description:

Returns the visible ID of this object's original Entity object.

Use this method to get the visible ID of the Entity object that is the immediate original of this object. The returned ID may correspond to an object that is a duplicate of another

Entity object. See the **GetOriginal Method** for information on how to track a string of duplicate records back to the source.

The returned ID is a string containing the defect number the user sees on the form and is of the format SITEnnnnnnn (for example, "PASNY00012343"). Do not confuse this ID with the invisible database ID, which is used internally by the database to keep track of records.

Note: It is an error to call this method for an Entity object that is not a duplicate. You should always call the IsDuplicate method first to verify that the object is a duplicate.

VBScript Syntax:

[*entity.*]**GetOriginalID**

Perl Syntax:

$*entity->***GetOriginalID();**

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	A String containing the ID of this object's original Entity.

Example (in VBScript):

```
' Display a dialog box indicating which record is
' the original of this record
If entity.IsDuplicate Then
  ' Get the ID of this record
  duplicateID = entity.GetDisplayName

  ' Get the ID of the original record
  originalID = entity.GetOriginalID
  OutputDebugString "The parent of record " & duplicateID & _
            " is record " & originalID
End If
```

Example (in Perl):

```
# Display a dialog box indicating which record is
# the original of this record

if ($entity->IsDuplicate())
 {
 # Get the ID of this record
 $duplicateID =$entity->GetDisplayName();

 # Get the ID of the original record
 $originalObj = $entity->GetOriginal();
 $originalID = $originalObj->GetDisplayName();
 $session->OutputDebugString("The parent of record
      ".$duplicateID." is record ".$originalID);
 }
```

See Also:

GetAllDuplicates Method
GetDuplicates Method
GetOriginal Method
HasDuplicates Method
IsDuplicate Method
IsOriginal Method
MarkEntityAsDuplicate Method of the **Session Object**
UnmarkEntityAsDuplicate Method of the **Session Object**
Updating Duplicate Records to Match the Parent Record

GetSession Method

Description:

Returns the current **Session Object**.

This method instantiates a new Session object using the current session information. This method is intended for use in hook code only and should not be called from any other context.

If you are creating a standalone application, you cannot call this method to obtain a Session object. You must create your own Session object and pass it to any standalone application methods that need it.

You can use this method to obtain the Session object associated with the current user. See the description of the Session object for more information on how to use this object.

VBScript Syntax:

`[`*`entity.`*`]`**`GetSession`**

Perl Syntax:

`$`*`entity->`*`GetSession();`

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). For Perl hooks, see **Getting a Session Object**.
Return Value	The **Session Object** representing the current database-access session.

Example (in VBScript):

```
set sessionObj = GetSession
```

Example (in Perl):

```
$sessionobj = $entity->GetSession();
```

See Also:

UserLogon Method of the **Session Object**
Updating Duplicate Records to Match the Parent Record

GetType Method

Description:

Returns the type (state-based or stateless) of the Entity.

You cannot change the type of an Entity object using the API. The type of a record is determined by the corresponding record type and must be set by the administrator using ClearQuest Designer.

VBScript Syntax:

[*entity.*]**GetType**

Perl Syntax:

$*entity*->**GetType();**

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	A Long whose value is an **EntityType Constants**: REQ_ENTITY for a state-based Entity object or AUX_ENTITY for a stateless Entity object.

Example (in VBScript):

```
recordType = GetType
If recordType = AD_REQ_ENTITY Then
  OutputDebugString "This record is a state-based record."
Else
  OutputDebugString "This record is a stateless record."
End If
```

Example (in Perl):

```
$recordtype = $entity->GetType();

if ($recordtype eq $CQPerlExt::CQ_REQ_ENTITY)
 {
```

```
 $session->OutputDebugString("This record is a state-based
      record.");
 }
else
 {
 $session->OutputDebugString("This record is a stateless
record.");
 }
```

See Also:

EntityType Constants
Notation Conventions for Perl
Notation Conventions for VBScript

HasDuplicates Method

Reports whether this object is the original of one or more duplicates.

An Entity can have more than one duplicate. Furthermore, an Entity can have duplicates and also be a duplicate itself. See the **IsDuplicate Method** and **IsOriginal Method** for details.

VBScript Syntax:

[*entity.*]**HasDuplicates**

Perl Syntax:

$*entity*->**HasDuplicates();**

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	A Boolean whose value is True if the Entity has any duplicates, otherwise False.

Example (in VBScript):

```
originalID = GetDisplayName
If HasDuplicates Then
  duplicateLinkList = GetDuplicates

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

Example (in Perl):

```
$originalID = $entity->GetDisplayName();
if ($entity->HasDuplicates())
 {
 @duplicateLinkList = $entity->GetDuplicates();

 # Output the IDs of the parent/child records

 foreach $duplicatelink (@duplicatelinklist)
  {
  $duplicateObj = $duplicateLink->GetChildEntity();
  $duplicateID = $duplicateObj->GetDisplayName();
  $session->OutputDebugString("Parent ID:".$originalID." child
      Id:"$duplicateID);
  }
 }
```

See Also:

GetAllDuplicates Method
GetDuplicates Method
GetOriginal Method
GetOriginalID Method
IsDuplicate Method
IsOriginal Method
MarkEntityAsDuplicate Method of the **Session Object**

UnmarkEntityAsDuplicate Method of the **Session Object**
Updating Duplicate Records to Match the Parent Record

InvalidateFieldChoiceList Method

Description:

Erases the values in a (dynamic) choice list, which can then be reset with **SetFieldChoiceList Method**.

Makes the “cached” choice list for the field invalid so that when GetFieldChoiceList is called next time, the ClearQuest Form either gets a choice list from the database or a hook program.

VBScript Syntax:

[*entity.*]**InvalidateFieldChoiceList *field_name***

Perl Syntax:

$*entity->*__InvalidateFieldChoiceList (*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	None.

Example:

```
void InvalidateFieldChoiceList(fieldname)
```

See Also:

SetFieldChoiceList Method

IsDuplicate Method

Description:

Indicates whether this Entity object has been marked as a **duplicate** of another Entity object.

A duplicate object reflects the changes made to the **original** object. When an Entity object is marked as a duplicate, any changes that occur to the original object are reflected in the duplicate as well. ClearQuest maintains a link between the original object and each one of its duplicates to update these changes.

Attempting to modify an object that is marked as a duplicate will result in an error; you must modify the original object instead. To locate the original object, you can use the **GetOriginal Method** of the duplicate.

VBScript Syntax:

[*entity.*]**IsDuplicate**

Perl Syntax:

$*entity*->**IsDuplicate();**

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	A Boolean whose value is True if this Entity object has been marked as a duplicate of another Entity object, otherwise False.

Example (in VBScript):

```
'Display a dialog box indicating which record is
'the original of this record
If entity.IsDuplicate Then
  ' Get the ID of this record
  duplicateID = entity.GetDisplayName

  ' Get the ID of the original record
```

```
  set originalObj = entity.GetOriginal
  originalID = originalObj.GetDisplayName
  OutputDebugString "The parent of record " & duplicateID & _
            " is record " & originalID
End If
```

Example (in Perl):

```
# Display a dialog box indicating which record is
# the original of this record

if ($entity->IsDuplicate)
 {
 # Get the ID of this record

 $duplicateID = $entity->GetDisplayName();

 # Get the ID of the original record
 $originalObj = $entity->GetOriginal();
 $originalID = $originalObj->GetDisplayName();
 $session->OutputDebugString("The parent of record
      ".$duplicateID. " is record ".$originalID);
 }
```

See Also:

GetAllDuplicates Method
GetDuplicates Method
GetOriginal Method
HasDuplicates Method
IsOriginal Method
MarkEntityAsDuplicate Method of the **Session Object**
UnmarkEntityAsDuplicate Method of the **Session Object**
Updating Duplicate Records to Match the Parent Record

IsEditable Method

Description:

Returns True if the Entity object can be modified at this time.

To edit an Entity object, you must either create a new object using the **BuildEntity Method** or open an existing object for editing with the **EditEntity Method**. An Entity object remains editable until you either commit your changes with the **Commit Method** or revert the Entity object with the **Revert Method**.

VBScript Syntax:

[*entity.*]**IsEditable**

Perl Syntax:

$*entity->*`IsEditable();`

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	A Boolean whose value is True if the Entity is currently editable, otherwise False.

Example (in VBScript):

```
set sessionObj = GetSession

entityToEdit = sessionObj.GetEntity("BUGID00000042")
sessionObj.EditEntity(entityToEdit, "modify")

' Verify that the entity object was opened for editing.
If Not entityToEdit.IsEditable Then
  OutputDebugString "Error - the entity object could not be
      edited."
End If
```

Example (in Perl):

```
$sessionObj = $entity->GetSession();

$entityToEdit = $sessionObj->GetEntity("BUGID00000042");
$sessionObj->EditEntity($entityToEdit, "modify");

# Verify that the entity object was opened for editing.
if (!$entityToEdit->IsEditable())
 {
 $session->OutputDebugString("Error - the entity object could not
be edited.");
 }
```

See Also:

Commit Method
Revert Method
BuildEntity Method of the **Session Object**
EditEntity Method of the **Session Object**

IsOriginal Method

Description:

Returns True if this Entity has duplicates but is not itself a duplicate.

This method reports whether an Entity object is a true original, that is, one that is not itself a duplicate. If this method returns True, then the **IsDuplicate Method** must return False and the **HasDuplicates Method** must return True. An Entity object must have at least one duplicate to be considered an original.

VBScript Syntax:

```
[entity.]IsOriginal
```

Perl Syntax:

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

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	A Boolean whose value is True if this object has duplicates but is not itself marked as a duplicate of any other Entity object.

Example (in VBScript):

```
'Display a dialog box indicating the IDs of the
' the duplicates of this record
If entity.IsOriginal Then
  ' Get the ID of this record
  originalID = entity.GetDisplayName
```

```
  ' Display the IDs of its duplicates
  duplicateLinkList = entity.GetDuplicates
  For Each duplicateLink In duplicateLinkList
     duplicateObj = duplicateLink.GetChildEntity
     duplicateID = duplicateObj.GetDisplayName
     OutputDebugString "Parent ID:" & originalID & _
           " child Id:" & duplicateID
  Next
End If
```

Example (in Perl):

```
#Display a dialog box indicating the IDs of the
# the duplicates of this record
if ($entity->IsOriginal())
 {
 # Get the ID of this record
 $originalID = $entity->GetDisplayName();

 # Display the IDs of its duplicates
 @duplicateLinkList = $entity->GetDuplicates();

 foreach $duplicateLink (@duplicateLinkList)
  {
  $duplicateObj = $duplicateLink->GetChildEntity();
  $duplicateID = $duplicateObj->GetDisplayName();
  $session->OutputDebugString("Parent ID: ".$originalID." child
Id:".$duplicateID);
  }
 }
```

See Also:

GetAllDuplicates Method
GetDuplicates Method
GetOriginal Method
GetOriginalID Method
HasDuplicates Method
IsDuplicate Method
MarkEntityAsDuplicate Method of the **Session Object**
UnmarkEntityAsDuplicate Method of the **Session Object**
Updating Duplicate Records to Match the Parent Record

LookupStateName Method

Description:

Returns the name of the Entity object's current state.

If the Entity object is not editable, this method simply returns the current state of the record. If the Entity object is editable and the current action involves a change of state, this method returns the new state of the record.

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

VBScript Syntax:

[*entity.*]**LookupStateName**

Perl Syntax:

$*entity*->**LookupStateName();**

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	A String containing the name of the Entity object's current state. If this Entity object is stateless, this method returns an empty String ('').

Example (in VBScript):

```
currentState = LookupStateName
```

Example (in Perl):

```
# Find the entity's current state name
$currentstate = $entity->LookupStateName();
```

See Also:

GetFieldValue Method
EditEntity Method of the **Session Object**

Revert Method

Description:

Discards any changes made to the Entity object.

Use this method to exit the transaction that allowed the record to be edited. You should call this method if you tried to change a record and the Validate method returned an error string.

You can call this method only if the Entity object is editable. To make an existing Entity object editable, call the **EditEntity Method** of the Session object. If you call this method on a newly created Entity object, one that was created with the BuildEntity method, this method cancels the submission of the record.

This method reverts the Entity's fields to the values that were stored in the database. After reverting, the Entity is no longer editable, so you must call the EditEntity method again to make new modifications.

VBScript Syntax:

```
[entity.]Revert
```

Perl Syntax:

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

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.

Example (in VBScript):

```
set sessionObj = GetSession
entityToEdit = sessionObj.GetEntity("defect", "BUGID00000042")
sessionObj.EditEntity(entityToEdit, "modify")

' Revert the changes to the record
entityToEdity.Revert

Example (in Perl):
# Get the current session
$sessionobj = $entity->GetSession();

# Select an entity to modify
$entityobj = $session->GetEntity("defect","BUGID00000042");

# Take the modify action on the entity object
$sessionobj.EditEntity($entityobj,"modify");

# ...make modifications to the entity object

# Revert the changes

$entityobj->Revert();

# At this point, the entity object is no longer modifiable
```

See Also:

Commit Method
IsEditable Method
Validate Method
EditEntity Method of the **Session Object**
Extracting Data About an EntityDef (Record Type)

SetFieldChoiceList Method

Description:

Resets a dynamic choice list. Can be use with **InvalidateFieldChoiceList Method** to empty any values already stored.

Use this function to force the ClearQuest client to fetch the new choice list values. You can set the values with this function or by other means (for example, a hook script).

You can design your schema so that ClearQuest recalculates a choice list every time a user interacts with it (no cached values), or only the first time (cached values). If you want to refresh cached values, call **InvalidateFieldChoiceList Method** to empty any cached values, then call SetFieldChoiceList to reinitialize the values. (The first time the choice list appears, there is no need to call **InvalidateFieldChoiceList Method** because no values pre-exist in cache memory.)

Use these two methods in a Value-Changed Field hook. For example, if the end-user selects a new item from the list of projects, the record type changes, and the form needs a refreshed dependent choice list.

VBScript Syntax:

```
[entity.]SetFieldChoiceList fieldName, choiceList
```

Perl Syntax:

```
$entity->SetFieldChoiceList(fieldName, choiceList);
```

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).
fieldName	A String that identifies a valid field name of an entity.
choiceList	[A VB variant containing a string or] a Perl string array.
Return Value	None.

VBScript Example:

```
SetFieldChoiceList(fieldname, VARIANT choiceList)
```

Note: Sets a list of acceptable values for the field. The parameter choiceList is of the type of Variant and must contain an array of strings.
In the current implementation, you cannot pass a reference to a variant.

Perl Example:

```
SetFieldChoiceList($fieldname, @choiceList)
```

Sets a list of acceptable values for the field. The parameter choiceList is of the type of array and must contain an array of strings.

See Also:

InvalidateFieldChoiceList Method

SetFieldRequirednessForCurrentAction Method

Description:

Sets the behavior of a field for the duration of the current action.

Use this method to set the field behavior to mandatory, optional, or read-only. Once the action has been committed, the behavior of the field reverts to read-only.

You can call this method only if the Entity object is editable. To make an existing Entity object editable, call the **EditEntity Method** of the Session object.

VBScript Syntax:

```
[entity.]SetFieldRequirednessForCurrentAction field_name, newValue
```

Perl Syntax:

```
$entity->SetFieldRequirednessForCurrentAction(field_name, newValue);
```

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 entity.

Identifier	Description
newValue	A Long identifying the new behavior type of the field. This value corresponds to one of the constants of the Behavior enumerated type. (It is illegal to use the USE_HOOK constant.)
Return Value	None.

Example (in VBScript):

```
' Change all mandatory fields to optional
fieldNameList = GetFieldNames
For Each fieldName in fieldNameList
  fieldReq = GetFieldRequiredness(fieldName)
  if fieldReq = AD_MANDATORY Then
     SetFieldRequirednessForCurrentAction fieldName, AD_OPTIONAL
  End If
Next
```

Example (in Perl):

```
# Change all MANDATORY fields to OPTIONAL

# Retrieve the collection of fields
@fieldnamelist = $entity->GetFieldNames()

foreach $fieldname (@fieldnamelist)
 {
 # Find out if the selected field is mandatory
 $fieldreq = $entity->GetFieldRequiredness($fieldname);
 if ($fieldreq $CQPerlExt::CQ_MANDATORY)
  {
  # Since it is, make it optional

$entity->SetFieldRequirenessForCurrentAction($fieldname,$CQPerlEx
t::CQ_OPTIONAL);
  }
 }
```

See Also:

GetFieldRequiredness Method
EditEntity Method of the **Session Object**

Behavior Constants
Notation Conventions for Perl
Notation Conventions for VBScript

SetFieldValue Method

Description:

Places the specified value in the named field.

If the field can be changed, this method sets its new value, regardless of whether that value is valid, and returns the empty String. To determine whether a field contains a valid value, obtain the **FieldInfo Object** for that field and call the **ValidityChangedThisSetValue Method** of the FieldInfo object to validate the field.

If the field cannot be changed, the returned String indicates why the field cannot be changed. Typical values include "no such field", "record is not being edited", and "field is read-only".

If the field can have multiple values instead of just one, use the **AddFieldValue Method** to add each new value. It is still legal to use SetFieldValue; however, using SetFieldValue on a field that already contains a list of values replaces the entire list with the single new value.

You can call this method only if the Entity object is editable. To make an existing Entity object editable, call the **EditEntity Method** of the Session object.

VBScript Syntax:

```
[entity.]SetFieldValue field_name, new_value
```

Perl Syntax:

```
$entity->SetFieldValue(field_name, new_value);
```

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 containing a valid field name of this Entity object.
new_value	For VB, a Variant containing the new setting for the field. For Perl, a string containing the new value.
Return Value	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.

Example (in VBScript):

```
' Set two field values, but only check errors for
' the second field.
entity.SetFieldValue "field1", "new value"
returnVal = SetFieldValue("field2", "100")
```

Example (in Perl):

```
# Set two field values for the entity
# Perform error checking on the second field
$entity->SetFieldValue("field1","new value");
$returnval = $entity->SetFieldValue("field2","100");
```

See Also:

BuildEntity Method of the **Session Object**
EditEntity Method of the **Session Object**

AddFieldValue Method
GetFieldValue Method
ValidityChangedThisSetValue Method
ValueChangedThisSetValue Method
FieldInfo Object
Updating Duplicate Records to Match the Parent Record
Extracting Data About an EntityDef (Record Type)

Validate Method

Description:

Validates the Entity object and reports any errors.

Before an Entity can be committed, it must be validated (even if no fields have been changed). If you are changing the contents of a record programmatically, you should make sure that your code provides valid data.

You should not attempt to parse and interpret the returned String programmatically, because the error text may change in future releases. If you want to try to correct the value in an invalid field, you can use the **GetInvalidFieldValues Method** to get the **FieldInfo Object** for that field.

You can call this method only if the Entity object is editable. To make an existing Entity object editable, call the **EditEntity Method** of the Session object.

VBScript Syntax:

```
[entity.]Validate
```

Perl Syntax:

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

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	If the Entity object is valid, this method returns the empty String ('). If any validation errors are detected, the String contains an explanation of the problem, suitable for presenting to the user.

Example (in VBScript):

```
set sessionObj = GetSession

set entityObj = sessionObj.GetEntity("defect", "BUGID00000042")
sessionObj.EditEntity entityObj, "modify" ...
```

```
' modify the Entity object
entityObj.Validate
entityObj.Commit
' The Entity object is no longer editable
```

Example (in Perl):

```
# Get the current session
$sessionobj = $entity->GetSession();

# Select an entity to modify
$entityobj = $session->GetEntity("defect","BUGID00000042");

# Take the modify action on the entity object
$sessionobj.EditEntity($entityobj,"modify");

# ...make modifications to the entity object

# Validate and commit the changes
$entityobj.Validate();
$entityobj.Commit();

# At this point, the entity object is no longer modifiable
```

See Also:

Commit Method
GetInvalidFieldValues Method
Revert Method
FieldInfo Object
Updating Duplicate Records to Match the Parent Record

EntityDef Object

An EntityDef object represents one of the **record type** in a schema.

In a schema, a record type specifies the metadata for one kind of record. The record type metadata defines the generic structure of that record. Metadata does not include the user data itself. Record type metadata includes the number of fields, the names of the fields, which data type each field must contain, the names of permitted actions, the names of permitted states, and so on.

An EntityDef object is the runtime representation of a record type. An EntityDef object contains information ClearQuest uses to create corresponding Entity objects at runtime. EntityDef objects can be either state-based or stateless. A state-based EntityDef object contains information about the states in which a corresponding Entity object can be placed. A stateless EntityDef object does not have any state information, but does specify which field of the Entity object is used as the unique key.

You cannot create or modify EntityDef objects at runtime. To create a new EntityDef object, you must define a corresponding record type using ClearQuest Designer. You can use an EntityDef object to obtain information about the corresponding record type. For example, you can use the **GetFieldDefNames Method**, **GetActionDefNames Method**, and **GetStateDefNames Method** to obtain the names of the record type's fields, actions, and states, respectively. You can also use the **GetFieldDefType Method** or **GetActionDefType Method** to obtain the type of a particular field or action.

You can use methods of the current **Session Object** to discover the available EntityDef objects.

Note: If you need to create a new data record, see the Session object's **BuildEntity Method**.

See Also:

Entity Object
Session Object
Extracting Data About an EntityDef (Record Type)

EntityDef Object Methods

The following list summarizes the EntityDef Object methods:

Method name	Description
DoesTransitionExist Method	Returns the list of transitions that exist between two states.
GetActionDefNames Method	Returns the action names defined in the EntityDef object.
GetActionDefType Method	Identifies the type of the specified action.
GetActionDestStateName Method	Returns the name of the destination state of a given action def.
GetFieldDefNames Method	Returns the field names defined in the EntityDef object.
GetFieldDefType Method	Identifies the type of data that can be stored in the specified field.
GetFieldReferenceEntityDef Method	Returns the type of record referenced by the specified field.
GetHookDefNames Method	Returns the list of named hooks associated with records of this type.
GetLocalFieldPathNames Method	Returns the path names of local fields.
GetName Method	Returns the name of the EntityDef object's corresponding record type.
GetStateDefNames Method	Returns the state names defined in the EntityDef object.
GetType Method	Returns the type (state-based or stateless) of the EntityDef.
IsActionDefName Method	Identifies whether the EntityDef object contains an action with the specified name.
IsFamily Method	Returns true of a given entitydef defines a family.
IsFieldDefName Method	Identifies whether the EntityDef object contains a field with the specified name.

Method name	Description
IsStateDefName Method	Identifies whether the EntityDef object contains a state with the specified name.
IsSystemOwnedFieldDefName Method	Returns a Bool indicating whether the specified field is owned by the system.

DoesTransitionExist Method

Description:

Returns the list of transitions that exist between two states.

The list of transitions is returned in no particular order. You must examine each entry in the array until you find the name of the action you are looking for.

VBScript Syntax:

```
[entitydef.]DoesTransitionExist sourceState, destState
```

Perl Syntax:

```
$entitydef->DoesTransitionExist(sourceState, destState);
```

Identifier	Description
entitydef	An EntityDef object corresponding to a record type in a schema.
sourceState	A String containing the name of the state that is the source of the transition.
destState	A String containing the name of the state that is the destination of the transition.
Return Value	For VB, if at least one transition between the two states exists, this method returns a Variant containing a list of strings. Each string corresponds to the name of an action. If no transitions exist, this method returns an EMPTY variant. For Perl, if at least one transition between the two states exists, this method returns a reference to a String Array.

Example (in VBScript):

```
set sessionObj  = GetSession
set entityDefObj = sessionObj.GetEntityDef(GetEntityDefName())

transitions = entityDefObj.DoesTransitionExist("open", "resoved")
If transitions <> Empty Then
  ' Simply initiate an action using the first entry.
  sessionObj.EditEntity entity, transitions(0)

  ...
End If
```

Example (in Perl):

```
$sessionObj = $entity->GetSession();
$entityDefObj =
$sessionObj->GetEntityDef($entity->GetEntityDefName());

@transitions = $entityDefObj->DoesTransitionExist("open",
      "resoved");

if (@transitions)
 {
 # Simply initiate an action using the first entry.
 $sessionObj->EditEntity($entity, @transitions[0]);
 }
```

See Also:

GetActionDefNames Method
IsActionDefName Method

GetActionDefNames Method

Description:

Returns the action names defined in the EntityDef object.

The list of actions is returned in no particular order. You must examine each entry in the array until you find the name of the action you are looking for.

Like the other parts of an EntityDef object, the administrator sets the defined actions using ClearQuest Designer. They cannot be set directly from the API.

VBScript Syntax:

[*entitydef.*]**GetActionDefNames**

Perl Syntax:

$*entitydef->***GetActionDefNames();**

Identifier	Description
entitydef	An EntityDef object corresponding to a record type in a schema.
Return Value	For VB, a Variant containing an Array whose elements are Strings. Each String names one action. If the EntityDef object has no actions, the return value is an Empty Variant. For Perl, a reference to a String Array is returned.

Example (in VBScript):

```
set sessionObj  = GetSession
set entityDefObj = sessionObj.GetEntityDef(GetEntityDefName())

sessionObj.OutputDebugString "Action names for " &
entityDefObj.GetName()

nameList = entityDefObj.GetActionDefNames()
For Each actionName in nameList
  sessionObj.OutputDebugString actionName
Next
```

Example (in Perl):

```
$sessionObj = $entity->GetSession();
$sessionObj->GetEntityDef($entity->GetEntityDefName());

$sessionObj.OutputDebugString("Action names for
"$entityDefObj->GetName());

$nameList = $entityDefObj->GetActionDefNames());
$count = $nameList->count();
```

```
for ($i = 0; $i < $count; $i++)
  {
  $field = $nameList -> Item($i);
 $sessionObj->OutputDebugString($actionName);
 }
```

See Also:

GetActionDefType Method
IsActionDefName Method
ActionType Constants
Extracting Data About an EntityDef (Record Type)

GetActionDefType Method

Description:

Identifies the type of the specified action.

You can use the **GetActionDefNames Method** to obtain the list of valid values for the *action_def_name* parameter.

The record type controls what types of actions are permitted for a given record. (See the **ActionType Constants** for the complete list.)

Like the other parts of an EntityDef object, the administrator sets the defined actions using ClearQuest Designer. They cannot be set directly from the API.

VBScript Syntax:

[*entitydef.*]**GetActionDefType** *action_def_name*

Perl Syntax:

$*entitydef*->**GetActionDefType(***action_def_name***);**

Identifier	Description
entitydef	An EntityDef object corresponding to a record type in a schema.

Identifier	Description
action_def_name	A String that identifies a valid action name of entitydef.
Return Value	A Long that specifies the type of the action specified in action_def_name.The value corresponds to one of the **ActionType Constants**.

Example (in VBScript):

```
set sessionObj  = GetSession
set entityDefObj = sessionObj.GetEntityDef(GetEntityDefName())

sessionObj.OutputDebugString "Modify action names for " & _
        entityDefObj.GetName()

' List the action names whose type is "modify"
nameList = entityDefObj.GetActionDefNames()
For Each actionName in nameList
  actionType = entityDefObj.GetActionDefType(actionName)
  if actionType = AD_MODIFY Then
     sessionObj.OutputDebugString actionName
  End If
Next
```

See Also:

GetActionDefNames Method
IsActionDefName Method
Extracting Data About an EntityDef (Record Type)
Notation Conventions for VBScript
Notation Conventions for Perl

GetActionDestStateName Method

Description:

Returns the destination state name associated with the current action.

Use this call to allow an external application to navigate the state transition matrix.

Whereas **GetDefaultActionName Method** returns the default action name associated with the current state, this method returns the destination state name associated with the current action.

VBScript Syntax:

```
entitydef.GetActionDestStateName actionDefName
```

Perl Syntax:

```
$entitydef->GetActionDestStateName(actionDefName);
```

Identifier	Description
entitydef	An EntityDef object corresponding to a record type in a schema.
actionDefName	A String that identifies a valid action name.
Return Value	A String that specifies the destination state of a given action def.

See Also:

GetDefaultActionName Method

GetFieldDefNames Method

Description:

Returns the field names defined in the EntityDef object.

The list of fields is returned in no particular order. You must examine each entry in the array until you find the name of the field you are looking for.

Like the other parts of an EntityDef object, the administrator sets the defined fields using ClearQuest Designer. They cannot be set directly from the API.

VBScript Syntax:

```
entitydef.GetFieldDefNames
```

Perl Syntax:

```
$entitydef->GetFieldDefNames();
```

Identifier	Description
entitydef	An EntityDef object corresponding to a record type in a schema.
Return Value	For VB, a Variant containing an Array whose elements are Strings is returned. Each String contains the name of one field. If the EntityDef object has no fields, the return value is an Empty Variant. For Perl, a reference to a String Array is returned.

Example (in VBScript):

```
set sessionObj  = GetSession
set entityDefObj = sessionObj.GetEntityDef(GetEntityDefName())

sessionObj.OutputDebugString "Field names for " &
      entityDefObj.GetName()

' List the field names in the record
nameList = entityDefObj.GetFieldDefNames()
For Each fieldName in nameList
  sessionObj.OutputDebugString fieldName
Next
```

Example (in Perl):

```
$sessionObj = $entity->GetSession();
$entityDefObj =
     $sessionObj->GetEntityDef($entity->GetEntityDefName());

$sessionObj.OutputDebugString("Field names for
"$entityDefObj->GetName());

$nameList = $entityDefObj->GetFieldDefNames());
$count = $nameList->count();

for ($i = 0; $i < $count; $i++)
  {
  $field = $nameList -> Item($i);
 $sessionObj->OutputDebugString($fieldName);
 }
```

See Also:

GetFieldDefType Method
IsFieldDefName Method
Extracting Data About an EntityDef (Record Type)

GetFieldDefType Method

Description:

Identifies the type of data that can be stored in the specified field.

You can use the **GetFieldDefNames Method** to obtain a list of valid field names.

The record type controls what type of data can be stored in each field of a corresponding data record. Fields can store strings, numbers, timestamps, references, and so on. (See the **FieldType Constants** for the complete list.)

Like the other parts of an EntityDef object, the administrator sets the defined fields using ClearQuest Designer. They cannot be set directly from the API.

VBScript Syntax:

entitydef.**GetFieldDefType** *field_def_name*

Perl Syntax:

$*entitydef*->**GetFieldDefType(***field_def_name***);**

Identifier	Description
entitydef	An EntityDef object corresponding to a record type in a schema.
field_def_name	A String that identifies a valid field name of entitydef.
Return Value	A Long that specifies what type of data can be stored in the named field. The value corresponds to one of the **FieldType Constants**.

Example (in VBScript):

```
set sessionObj  = GetSession
set entityDefObj = sessionObj.GetEntityDef(GetEntityDefName())
```

```
sessionObj.OutputDebugString "Integer fields of " & _
            entityDefObj.GetName()

' List the field names in the record that contain integers
nameList = entityDefObj.GetFieldDefNames()
For Each fieldName in nameList
  fieldType = entityDefObj.GetFieldDefType(fieldName)
  if fieldType = AD_INT Then
     sessionObj.OutputDebugString fieldName
  End If
Next
```

Example (in Perl):

```
$sessionObj = $entity->GetSession();
$entityDefObj =
     $sessionObj->GetEntityDef($entity->GetEntityDefName());

$sessionObj->OutputDebugString("Integer fields of
".$entityDefObj.GetName());

# List the field names in the record that contain integers
$nameList = $entityDefObj->GetFieldDefNames();

$count = $nameList->count();

for ($i = 0; $i < $count; $i++)
  {
  $field = $nameList -> Item($i);
 $fieldType = $entityDefObj->GetFieldDefType($fieldName);
 if ($fieldType eq $CQPerlExt::CQ_INT)
  {
  $sessionObj->OutputDebugString($fieldName);
  }
 }
```

See Also:

GetFieldDefNames Method
IsFieldDefName Method
Extracting Data About an EntityDef (Record Type)
Notation Conventions for VBScript
Notation Conventions for Perl

GetFieldReferenceEntityDef Method

Description:

Returns the type of record referenced by the specified field.

The specified field must contain a reference to other records. The type of the specified field must be one of the following: REFERENCE, REFERENCE_LIST,JOURNAL, or ATTACHMENT_LIST.

VBScript Syntax:

entitydef.**GetFieldReferenceEntityDef** *field_name*

Perl Syntax:

$*entitydef*->**GetFieldReferenceEntityDef(***field_name***);**

Identifier	Description
entitydef	An EntityDef object corresponding to a record type in a schema.
field_name	A String that identifies a valid field name of entitydef.
Return Value	An EntityDef object corresponding to the type of record referenced by the specified field.

Example (in VBScript):

```
set sessionObj  = GetSession
set entityDefObj = sessionObj.GetEntityDef(GetEntityDefName())

' List the type of rence fields
nameList = entityDefObj.GetFieldDefNames()
For Each fieldName in nameList
  fieldType = entityDefObj.GetFieldDefType(fieldName)
  if fieldType = AD_REFERENCE Then
     set refEDefObj =
entityDefObj.GetFieldReferenceEntityDef(fieldName)
     sessionObj.OutputDebugString refEDefObj.GetName()
  End If
Next
```

Example (in Perl):

```
$sessionObj = $entity->GetSession();
$entityDefObj =
$sessionObj->GetEntityDef($entity->GetEntityDefName());

# List the type of rence fields
$nameList = $entityDefObj->GetFieldDefNames();
$count = $nameList->count();

for ($i = 0; $i < $count; $i++)
  {
  $field = $nameList -> Item($i);
 $fieldType = $entityDefObj->GetFieldDefType($fieldName);
 if ($fieldType eq $CQPerlExt::CQ_REFERENCE)
  {
  $refEDefObj =
$entityDefObj->GetFieldReferenceEntityDef($fieldName);
  $sessionObj->OutputDebugString($refEDefObj->GetName());
  }
 }
```

See Also:

GetFieldDefType Method
Notation Conventions for VBScript
Notation Conventions for Perl

GetHookDefNames Method

Description:

Returns the list of named hooks associated with records of this type.

This method returns the list of Named hooks. Named hooks (also referred to as record hooks in the ClearQuest Designer user interface) are special functions used by ClearQuest form controls to implement specific tasks.

VBScript Syntax:

entitydef.**GetHookDefNames** *field_def_name*

Perl Syntax:

```
$entitydef->GetHookDefNames(field_def_name);
```

Identifier	Description
entitydef	An EntityDef object corresponding to a record type in a schema.
Return Value	For VB, a Variant containing a list of strings. Each string corresponds to the name of a hook associated with this record type. If no named hooks are associated with this record type, this method returns an EMPTY variant. For Perl, a reference to a String Array is returned.

Example (in VBScript):

```
set sessionObj  = GetSession
set entityDefObj = sessionObj.GetEntityDef(GetEntityDefName())

sessionObj.OutputDebugString "Hooks of " & entityDefObj.GetName()

' List the record type's hooks
nameList = entityDefObj.GetHookDefNames()
For Each hookName in nameList
  sessionObj.OutputDebugString hookName
Next
```

Example (in Perl):

```
$sessionObj = $entity->GetSession();
$entityDefObj =
$sessionObj->GetEntityDef($entity->GetEntityDefName());

$sessionObj->OutputDebugString("Hooks of
      ".$entityDefObj->GetName());

# List the record type's hooks
$nameList = $entityDefObj->GetHookDefNames();
$count = $nameList->count();

for ($i = 0; $i < $count; $i++)
  {
  $field = $nameList -> Item($i);
 $sessionObj->OutputDebugString($hookName);
```

```
}
```

See Also:

GetActionDefNames Method
GetFieldDefNames Method

GetLocalFieldPathNames Method

Description:

Returns the path names of local fields.

Each string in the returned variant contains the path name of a single field.

VBScript Syntax:

entitydef.**GetLocalFieldPathNames** *visible_only*

Perl Syntax:

$*entitydef*->**GetLocalFieldPathNames(***visible_only***);**

Identifier	Description
entitydef	An EntityDef object corresponding to a record type in a schema.
visible_only	A Bool, which if true restricts the list of fields to only those that are visible.
Return Value	For VB, a Variant containing a list of strings is returned. For Perl, a reference to a String Array is returned.

Example (in VBScript):

```
set sessionObj  = GetSession
set entityDefObj = sessionObj.GetEntityDef(GetEntityDefName())

pathNames = entityDefObj.GetLocalFieldPathNames(False)
For Each name in pathNames
  sessionObj.OutputDebugString "Path name: " & name
Next
```

Example (in Perl):

```
$sessionobj = $entity->GetSession();

$entitydefobj =
$sessionobj->GetEntityDef($entity->GetEntityDefName());

$pathnames = $entitydefobj->GetLocalFieldPathNames(0);

$count = $pathnames->count();

for ($i = 0; $i < $count; $i++)
  {
  $field = $pathnames -> Item($i);
 $sessionobj->OutputDebugString("Path name: ".$name);
 }
```

See Also:

GetFieldDefNames Method
IsFieldDefName Method

GetName Method

Description:

Returns the name of the EntityDef object's corresponding record type.

Like the other parts of an EntityDef object, the name of an EntityDef object is determined by the corresponding record type, whose name is set by the administrator using ClearQuest Designer. The name cannot be set directly from the API.

VBScript Syntax:

entitydef.**GetName**

Perl Syntax:

$*entitydef*->**GetName();**

Identifier	Description
entitydef	An EntityDef object corresponding to a record type in a schema.
Return Value	A String whose value is the name of the EntityDef object's corresponding record type.

Example (in VBScript):

```
set sessionObj  = GetSession
set entityDefObj = sessionObj.GetEntityDef(GetEntityDefName())
sessionObj.OutputDebugString "Name of record type: " & _
        entityDefObj.GetName()
```

Example (in Perl):

```
$sessionobj = $entity->GetSession();
$entitydefobj =
$sessionobj->GetEntityDef($entity->GetEntityDefName());

$sessionobj.OutputDebugString("Name of record type:
      ".$entitydefobj->GetName());
```

See Also:

GetType Method

Extracting Data About an EntityDef (Record Type)

GetStateDefNames Method

Description:

Returns the state names defined in the EntityDef object.

Like the other parts of an EntityDef object, the administrator sets the defined states using ClearQuest Designer. They cannot be set directly from the API.

VBScript Syntax:

entitydef.**GetStateDefNames**

Perl Syntax:

```
$entitydef->GetStateDefNames();
```

Identifier	Description
entitydef	An EntityDef object corresponding to a record type in a schema.
Return Value	For VB, a Variant containing an Array whose elements are Strings. Each String contains the name of one state. If the EntityDef object has no states, the return value is an Empty variant. For Perl, a reference to a String Array is returned.

Example (in VBScript):

```
set sessionObj  = GetSession
set entityDefObj = sessionObj.GetEntityDef(GetEntityDefName())

If entityDefObj.GetType = AD_REQ_ENTITY Then
  sessionObj.OutputDebugString "States of record type: " & _
            entityDefObj.GetName()

  ' List the possible states of the record
  nameList = entityDefObj.GetStateDefNames()
  For Each stateName in nameList
     sessionObj.OutputDebugString stateName
  Next
End If
```

Example (in Perl):

```
$sessionObj = $entity->GetSession();
$entityDefObj =
$sessionObj->GetEntityDef($entity->GetEntityDefName());

if ($entityDefObj->GetType eq $CQPerlExt::CQ_REQ_ENTITY)
 {
 $sessionObj->OutputDebugString("States of record type:
      ".$entityDefObj->GetName());

 # List the possible states of the record
 $nameList = $entityDefObj->GetStateDefNames();

 $count = $nameList->count();
```

```
for ($i = 0; $i < $count; $i++)
  {
  $field = $nameList -> Item($i);
   $sessionObj->OutputDebugString($stateName)
   }
 }
```

See Also:

GetType Method
IsStateDefName Method
Extracting Data About an EntityDef (Record Type)
Notation Conventions for VBScript
Notation Conventions for Perl

GetType Method

Description:

Returns the type (state-based or stateless) of the EntityDef.

Like the other parts of an EntityDef object, the type of an EntityDef object is determined by the corresponding record type, whose type is set by the administrator using ClearQuest Designer. The type cannot be set directly from the API.

VBScript Syntax:

entitydef.**GetType**

Perl Syntax:

$*entitydef*->**GetType();**

Identifier	Description
entitydef	An EntityDef object corresponding to a record type in a schema.
Return Value	A Long whose value is an **EntityType Constants**: REQ_ENTITY for a state-based EntityDef object or AUX_ENTITY for a stateless EntityDef object.

Example (in VBScript):

```
set sessionObj  = GetSession
set entityDefObj = sessionObj.GetEntityDef(GetEntityDefName())

If entityDefObj.GetType = AD_REQ_ENTITY Then
  sessionObj.OutputDebugString "States of record type: " & _
         entityDefObj.GetName()

  ' List the possible states of the record
  nameList = entityDefObj.GetStateDefNames()
  For Each stateName in nameList
     sessionObj.OutputDebugString stateName
  Next
End If
```

Example (in Perl):

```
$sessionObj = $entity->GetSession();
$entityDefObj =
$sessionObj->GetEntityDef($entity->GetEntityDefName());

if ($entityDefObj->GetType() eq $CQPerlExt::CQ_REQ_ENTITY)
 {
 $sessionObj->OutputDebugString("States of record type:
      ".$ entityDefObj->GetName());

 # List the possible states of the record
 $nameList = $entityDefObj->GetStateDefNames();
$count = $nameList->count();

for ($i = 0; $i < $count; $i++)
  {
  $field = $nameList -> Item($i);
   $sessionobj->OutputDebugString($statename);
   }
 }
```

See Also:

GetName Method

Extracting Data About an EntityDef (Record Type)

Notation Conventions for VBScript

Notation Conventions for Perl

IsActionDefName Method

Description:

Identifies whether the EntityDef object contains an action with the specified name.

VBScript Syntax:

entitydef.**IsActionDefName** *name*

Perl Syntax:

$*entitydef*->**IsActionDefName(***name***);**

Identifier	Description
entitydef	An EntityDef object corresponding to a record type in a schema.
name	A String containing the name of the action to verify.
Return Value	True if name is the name of an actual action in the EntityDef object; otherwise False.

Example (in VBScript):

```
set sessionObj  = GetSession
set entityDefObj = sessionObj.GetEntityDef(GetEntityDefName())

If entityDefObj.IsActionDefName("open") Then
  sessionObj.OutputDebugString "The record type supports the open
action"
End If
```

Example (in Perl):

```
$sessionObj = $entity->GetSession();
$entityDefObj =
$sessionObj->GetEntityDef($entity->GetEntityDefName());

if ($entityDefObj->IsActionDefName("open"))
 {
 $sessionObj->OutputDebugString("The record type supports the
open action");
 }
```

See Also:

GetActionDefNames Method
GetActionDefType Method

IsFamily Method

Description:

Returns the boolean value of True if this entitydef defines a family.

Use this call to determine whether a given entitydef is an entitydef or an entitydef family. The IsFamily method fetches a flag marked on the EntityDef object.

VBScript Syntax:

```
entitydef.IsFamily entitydef
```

Perl Syntax:

```
$entitydef->IsFamily(entitydef);
```

Identifier	Description
entitydef	An EntityDef object corresponding to a record type in a schema.
Return Value	A Boolean. True signifies the entitydef does define a family record type.

See Also:

GetEntityDef Method
GetEntityDefFamilyNames Method

IsFieldDefName Method

Description:

Identifies whether the EntityDef object contains a field with the specified name.

VBScript Syntax:

entitydef.**IsFieldDefName** *name*

Perl Syntax:

$*entitydef*->**IsFieldDefName(***name***);**

Identifier	Description
entitydef	An EntityDef object corresponding to a record type in a schema.
name	A String containing the name of the field to verify.
Return Value	True if name is the name of an actual field in the EntityDef object; otherwise False.

See Also:

GetFieldDefNames Method
GetFieldDefType Method

IsStateDefName Method

Description:

Identifies whether the EntityDef object contains a state with the specified name.

VBScript Syntax:

entitydef.**IsStateDefName** *name*

Perl Syntax:

$*entitydef*->**IsStateDefName(***name***);**

Identifier	Description
entitydef	An EntityDef object corresponding to a record type in a schema.

Identifier	Description
name	A String containing the name of the state to verify.
Return Value	True if name is the name of an actual state in the EntityDef object; otherwise False.

See Also:

GetStateDefNames Method
Extracting Data About an EntityDef (Record Type)

IsSystemOwnedFieldDefName Method

Description:

Returns a Bool indicating whether the specified field is owned by the system. System-owned fields are used internally by ClearQuest to maintain information about the database. You should never modify system fields directly as it could corrupt the database.

VBScript Syntax:

entitydef.**IsSystemOwnedFieldDefName** *field_name*

Perl Syntax:

$*entitydef*->**IsSystemOwnedFieldDefName(***field_name***);**

Identifier	Description
entitydef	An EntityDef object corresponding to a record type in a schema.
field_name	A String that identifies a valid field name of entitydef.
Return Value	True if the field is owned by the system, otherwise False.

See Also:

GetFieldDefNames Method

EntityDefs Object

The EntityDefs object (EntityDefs) is a collection object that contains a collection of EntityDef objects.

You can get the number of items in the collection by accessing the value in the Count Property. Use the Item Method to retrieve items from the collection.

See Also:

EntityDef Object

EntityDefs Object Property

The following list summarizes the EntityDefs Object properties:

Property name	Access	Description
Count Property	Read-only	Returns the number of items in the collection.

Count Property

Description:

Returns the number of items in the collection. This property is read-only.

VBScript Syntax:

collection.**Count**

Perl Syntax:

$collection->**Count();**

Identifier	Description
collection	An EntityDefs collection object, representing the set of EntityDefs available for fetching as a collection.
Data Type	A Long that specifies the number of items in the collection object. This method returns zero if the collection contains no items.

See Also:

Item Method
EntityDef Object

EntityDefs Object Method

The following list summarizes the EntityDefs Object method:

Method name	Description
Item Method	Returns the specified item in the collection.

Item Method

Description:

Returns the specified item in the collection. The argument to this method can be either a numeric index (`itemNum`) or a String (`name`).

Syntax:

```
collection.Item itemNum
collection.Item name
```

Identifier	Description
collection	An EntityDefs collection object representing the set of EntityDefs in a schema.
itemNum	A Long that serves as an index into the collection. This index is 0-based so the first item in the collection is numbered 0, not 1.
name	A String that serves as a key into the collection. This string corresponds to the unique key returned by the **DisplayName Property** of the desired EntityDefs.
Return value	The EntityDef object at the specified location in the collection.

See Also:

Count Property

EventObject Object

An EventObject contains information that is passed to the named hook of an Entity object.

This object is not accessible through the normal object model and you should not create this object directly. The properties of this object are for informational purposes and are read-only.

See Also

Entity Object

EventObject Object Properties

The following list summarizes the EventObject Object properties:

Property name	Access	Description
EventType Property	Read-only	Returns the type of event that caused the hook invocation.
ItemName Property	Read-only	Returns the name of the form item that caused the hook to be invoked.
ObjectItem Property	Read-only	Returns the entity object associated with the current selection.
StringItem Property	Read-only	Returns the name of the menu item hook.
CheckState Property		Reserved for future use.
EditText Property		Reserved for future use.

EventType Property

Description:

Returns the type of event that caused the hook invocation. This is a read-only property; it can be viewed but not set.

VBScript Syntax:

eventObject.**EventType**

Perl Syntax:

$*eventObject*->**EventType();**

Identifier	Description
eventObject	An instance of EventObject.
Data Type	A Long whose value is one of the EventType enumerated constants.

See Also:

EntityType Constants

ItemName Property

Description:

Returns the name of the form item that caused the hook to be invoked. This is a read-only property; it can be viewed but not set.

VBScript Syntax:

eventObject.**ItemName**

Perl Syntax:

$*eventObject*->**ItemName();**

Identifier	Description
eventObject	An instance of EventObject.
Data Type	A String containing the name of the form item from which the hook was invoked.

See Also:

ObjectItem Property

ObjectItem Property

Description:

Returns the entity object associated with the current selection. This is a read-only property; it can be viewed but not set.

The Entity object contained in this property may not be the same object that invoked the current hook. This property is set only when the EventType property contains the value ITEM_SELECTION or ITEM_DLBCLICK.

VBScript Syntax:

eventObject.**ObjectItem**

Identifier	Description
eventObject	An instance of EventObject.
Return value	The Entity object associated with the current selection.

See Also:

EventType Property
Entity Object

StringItem Property

Description:

Returns the name of the menu item hook. This is a read-only property; it can be viewed but not set.

VBScript Syntax:

eventObject.**StringItem**

Perl Syntax:

$*eventObject*->**StringItem();**

Identifier	Description
eventObject	An instance of EventObject.
Data Type	A String whose value indicates the menu item hook name when the EventType property contains the value CONTEXTMENU_ITEM_CONDITION; otherwise, this property contains an empty value.

See Also:

EventType Property
Notation Conventions for VBScript
Notation Conventions for Perl

FieldInfo Object

A FieldInfo object contains static information about one field of a user data record.

The FieldInfo object contains the information about one field of an Entity object. You can use the methods of FieldInfo to obtain the following information:

- the name of the field
- what type of data the field must contain
- whether a value is required in the field
- whether the field contains a value, and whether the value is valid
- what the error message is for an invalid value
- what the value stored in the field is
- whether the value or validity of the field has changed

A FieldInfo object is an informational object. All of its methods are for getting, rather than setting, values. To change the value stored in a field, use the **SetFieldValue Method** of Entity.

A FieldInfo object is a "snapshot" of the corresponding field in the database. If you change the value of that field with a call to SetFieldValue, the existing FieldInfo object does not reflect the change. To obtain an updated value for the field, you must get a new FieldInfo object.

To get an instance of FieldInfo, call the **GetFieldValue Method** of Entity, passing the name of the field as an argument. Other methods of Entity allow you to return one or more instances of FieldInfo that satisfy certain conditions. For for more details, see the methods of the **Entity Object**.

As a convenience, Entity contains a few methods that act as wrappers for FieldInfo methods. For example, the **GetFieldType Method** of Entity is equivalent to the **GetType Method** of FieldInfo. However, Entity also has some methods that have no FieldInfo counterparts, such as the **GetFieldOriginalValue Method** and the **GetFieldChoiceList Method**.

See Also:

GetFieldsUpdatedThisAction Method
GetFieldsUpdatedThisGroup Method
GetFieldsUpdatedThisSetValue Method
Entity Object

Extracting Data About a Field in a Record
Showing Changes to an Entity (Record)

FieldInfo Object Methods

The following list summarizes the FieldInfo Object methods:

Method name	Description
GetMessageText Method	Returns a String explaining why the value stored in the field is invalid.
GetName Method	Returns the name of the field.
GetRequiredness Method	Identifies the **behavior** of the specified field.
GetType Method	Identifies the type of data that can be stored in this field.
GetValidationStatus Method	Identifies whether the field's value is valid.
GetValue Method	Returns the field's value as a single String.
GetValueAsList Method	Returns an Array of Strings containing the values stored in the field.
GetValueStatus Method	Identifies whether the field currently has a value.
ValidityChangedThisAction Method	Returns True if the field's validity was changed by the most recent action.
ValidityChangedThisGroup Method	Returns True if the field's validity was changed by the most recent group of SetFieldValue calls.
ValidityChangedThisSetValue Method	Returns True if the field's validity was changed by the most recent **SetFieldValue Method** call.
ValueChangedThisAction Method	Returns True if this field's value was modified by the most recent action.
ValueChangedThisGroup Method	Returns True if the field's value was modified by the most recent group of SetFieldValue calls.
ValueChangedThisSetValue Method	Returns True if the field's value was modified by the most recent **SetFieldValue Method** call.

GetMessageText Method

Description:

Returns a String explaining why the value stored in the field is invalid.

Call this method only when the **GetValidationStatus Method** returns KNOWN_INVALID, otherwise the results are undefined; an exception might be thrown or an inaccurate message might be generated.

VBScript Syntax:

fieldInfo.**GetMessageText**

Perl Syntax:

$*fieldInfo*->**GetMessageText();**

Identifier	Description
fieldInfo	A FieldInfo object, which contains information about one field of a user data record.
Return value	A String that explains why this field's value is invalid.

See Also:

GetValidationStatus Method
Extracting Data About a Field in a Record
Notation Conventions for VBScript
Notation Conventions for Perl

GetName Method

Description:

Returns the name of the field. Field names are used by various methods to identify a specific field in an Entity object.

VBScript Syntax:

fieldInfo.**GetName**

Perl Syntax:

$*fieldInfo*->**GetName();**

Identifier	Description
fieldInfo	A FieldInfo object, which contains information about one field of a user data record.
Return value	A String containing the name of the field.

See Also:

GetFieldNames Method of the **Entity Object**
Extracting Data About a Field in a Record

GetRequiredness Method

Description:

Identifies the behavior of the specified field.

A field can be mandatory, optional, or read-only. If the Entity does not have an action running at the time this method is called, the return value will always be READONLY. If an action is running, the return value can be READONLY, MANDATORY, or OPTIONAL.

This method never returns the value USE_HOOK. If the behavior of the field is determined by a permission hook, ClearQuest will have already executed that hook and cached the resulting value. This method then returns the cached value.

VBScript Syntax:

fieldInfo.**GetRequiredness**

Perl Syntax:

$*fieldInfo*->**GetRequiredness();**

Identifier	Description
fieldInfo	A FieldInfo object, which contains information about one field of a user data record.
Return value	A Long that identifies the behavior of this field. The value corresponds to one of the Behavior enumeration constants (with the exception of the USE_HOOK constant).

See Also:

GetFieldRequiredness Method of the **Entity Object**
Notation Conventions for VBScript
Notation Conventions for Perl

GetType Method

Description:

Identifies the type of data that can be stored in this field.

Fields can store strings, numbers, timestamps, references, and several other types. (See **FieldType Constants** for the complete list.)

VBScript Syntax:

fieldInfo.**GetType**

Perl Syntax:

$*fieldInfo*->**GetType();**

Identifier	Description
fieldInfo	A FieldInfo object, which contains information about one field of a user data record.
Return value	A Long that specifies what type of data can be stored in this field. The value corresponds to one of the **FieldType Constants**.

See Also:

GetFieldType Method of the **Entity Object**

GetValidationStatus Method

Description:

Identifies whether the field's value is valid.

The value in the field can be valid, invalid, or its status can be unknown. If field validation has not yet been performed (for example, if this method is invoked inside a hook), this method returns NEEDS_VALIDATION, whether or not the field has a value.

VBScript Syntax:

fieldInfo.**GetValidationStatus**

Perl Syntax:

$*fieldInfo*->**GetValidationStatus();**

Identifier	Description
fieldInfo	A FieldInfo object, which contains information about one field of a user data record.
Return value	A Long that identifies the validation status of this field. The value corresponds to one of the **FieldValidationStatus Constants**.

See Also:

GetMessageText Method
Extracting Data About a Field in a Record
Notation Conventions for VBScript
Notation Conventions for Perl

GetValue Method

Description:

Returns the field's value as a single String.

This method returns a single String. If a field contains a list of values, the String contains a concatenation of the values, separated by newline characters. For a field that returns multiple values, you can use the **GetValueAsList Method** to get a separate String for each value.

VBScript Syntax:

fieldInfo.**GetValue**

Perl Syntax:

$*fieldInfo*->**GetValue();**

Identifier	Description
fieldInfo	A FieldInfo object, which contains information about one field of a user data record.
Return value	A String that contains the value or values stored in the field.

See Also:

GetValueAsList Method
GetFieldValue Method of the Entity Object
Extracting Data About a Field in a Record
Showing Changes to an Entity (Record)

GetValueAsList Method

Description:

Returns an Array of Strings containing the values stored in the field.

It is legal to use this method for a scalar field (that is, one that contains a single value). In that case, this method returns only one element in the Array (unless the field is empty in which case an Empty Variant is returned).

To determine if a field can contain multiple values, call the **GetType Method** on the corresponding FieldInfo object. If the type of the field is REFERENCE_LIST, ATTACHMENT_LIST, or JOURNAL, the field can contain multiple values.

Note: Fields whose type is either ATTACHMENT_LIST or JOURNAL cannot be modified programmatically.

VBScript Syntax:

```
fieldInfo.GetValueAsList
```

Perl Syntax:

```
$fieldInfo->GetValueAsList();
```

Identifier	Description
fieldInfo	A FieldInfo object, which contains information about one field of a user data record.
Return value	For VB, a Variant containing an Array whose elements are Strings. Each String contains the a single value stored in the field. If the field contains no values, this method returns an Empty Variant. For Perl, a reference to a String Array is returned.

See Also:

GetValue Method
AddFieldValue Method of the **Entity Object**
GetFieldValue Method of the **Entity Object**
FieldType Constants
Notation Conventions for VBScript
Notation Conventions for Perl

GetValueStatus Method

Description:

Identifies whether the field currently has a value.

VBScript Syntax:

```
fieldInfo.GetValueStatus
```

Perl Syntax:

```
$fieldInfo->GetValueStatus();
```

Identifier	Description
fieldInfo	A FieldInfo object, which contains information about one field of a user data record.
Return value	A Long that identifies the status of this field. The value corresponds to one of the ValueStatus enumeration constants.

See Also:

GetValue Method
GetFieldValue Method of the **Entity Object**
SetFieldValue Method of the **Entity Object**
Extracting Data About a Field in a Record
Showing Changes to an Entity (Record)

ValidityChangedThisAction Method

Description:

Returns True if the field's validity was changed by the most recent action.

This method considers only those changes that were made after the action was initiated. If the field was implicitly changed during action startup and not afterwards, this method returns False. For example, if a CHANGE_STATE action moves the record from, say, "assigned" to "resolved", the field "resolution info" might become mandatory. The

validity will therefore be "invalid" until you fill it in. However, this validity change will not be reflected by ValidityChangedThisAction.

This mechanism only detects actions for the Entity object to which this field belongs. It ignores actions on other Entity objects.

VBScript Syntax:

fieldInfo.**ValidityChangedThisAction**

Perl Syntax:

$*fieldInfo->***ValidityChangedThisAction();**

Identifier	Description
fieldInfo	A FieldInfo object, which contains information about one field of a user data record.
Return value	A Boolean that is True if the field's validity changed since the most recent action was initiated, otherwise False.

See Also:

ValidityChangedThisGroup Method
ValidityChangedThisSetValue Method
ValueChangedThisAction Method
GetFieldsUpdatedThisAction Method of the **Entity Object**
Extracting Data About a Field in a Record

ValidityChangedThisGroup Method

Description:

Returns True if the field's validity was changed by the most recent group of **SetFieldValue Method** calls.

This method tells you whether the validity of the field changed. In some cases, the validity can change even if this field's value did not. For example, its validity might be dependent upon another field's value.

The grouping mechanism detects BeginNewFieldUpdateGroup and SetFieldValue calls only for the Entity object to which this field belongs. It ignores calls that apply to other Entity objects.

You can instead use the **ValidityChangedThisSetValue Method** if you only care about the most recent SetFieldValue call.

VBScript Syntax:

fieldInfo.**ValidityChangedThisGroup**

Perl Syntax:

$*fieldInfo*->**ValidityChangedThisGroup();**

Identifier	Description
fieldInfo	A FieldInfo object, which contains information about one field of a user data record.
Return value	A Boolean that is True if the field's validity changed since the most recent call to the **BeginNewFieldUpdateGroup Method**, otherwise False.

See Also:

ValidityChangedThisAction Method
ValidityChangedThisSetValue Method
ValueChangedThisGroup Method
BeginNewFieldUpdateGroup Method of the Entity Object
GetFieldsUpdatedThisGroup Method of the **Entity Object**
FieldValidationStatus Constants
Extracting Data About a Field in a Record

ValidityChangedThisSetValue Method

Description:

Returns True if the field's validity was changed by the most recent SetFieldValue call.

This method tells you whether the validity of the field changed. (In some cases, the validity can change even if this field's value did not. For example, its validity might be dependent upon another field's value.)

This mechanism detects SetFieldValue calls only for the Entity object to which this field belongs. It ignores SetFieldValue calls that apply to other Entity objects.

VBScript Syntax:

fieldInfo.**ValidityChangedThisSetValue**

Perl Syntax:

$*fieldInfo->***ValidityChangedThisSetValue();**

Identifier	Description
fieldInfo	A FieldInfo object, which contains information about one field of a user data record.
Return value	A Boolean that is True if the field's validity was changed by the most recent call to SetFieldValue, otherwise False.

See Also:

ValidityChangedThisAction Method
ValidityChangedThisGroup Method
GetFieldsUpdatedThisSetValue Method of the **Entity Object**
SetFieldValue Method of the **Entity Object**
Extracting Data About a Field in a Record

ValueChangedThisAction Method

Description:

Returns True if this field's value was modified by the most recent action.

This method considers changes that were made after the action was initialized, that is, after the BuildEntity or EditEntity method returned. This method returns False if the field was implicitly changed only during the action's startup phase.

This mechanism detects actions that take place only on the Entity object to which this field belongs. It ignores actions on other Entity objects.

VBScript Syntax:

fieldInfo.**ValueChangedThisAction**

Perl Syntax:

$*fieldInfo*->**ValueChangedThisAction();**

Identifier	Description
fieldInfo	A FieldInfo object, which contains information about one field of a user data record.
Return value	A Boolean that is True if the field's value was changed since the most recent action was initiated, otherwise False.

See Also:

ValueChangedThisGroup Method
ValueChangedThisSetValue Method
BuildEntity Method of the **Session Object**
EditEntity Method of the **Session Object**
GetFieldsUpdatedThisAction Method of the **Entity Object**
Extracting Data About a Field in a Record

ValueChangedThisGroup Method

Description:

Returns True if the field's value was modified by the most recent group of SetFieldValue calls.

This mechanism detects BeginNewFieldUpdateGroup and SetFieldValue calls only for the Entity object to which this field belongs.

You can use the **ValueChangedThisSetValue Method** if you only care about the most recent SetFieldValue call.

VBScript Syntax:

fieldInfo.**ValueChangedThisGroup**

Perl Syntax:

$*fieldInfo*->**ValueChangedThisGroup();**

Identifier	Description
fieldInfo	A FieldInfo object, which contains information about one field of a user data record.
Return value	A Boolean that is True if the field's value was changed since the most recent invocation of BeginNewFieldUpdateGroup, otherwise False.

See Also:

ValueChangedThisAction Method
ValueChangedThisSetValue Method
BeginNewFieldUpdateGroup Method of the **Entity Object**
GetFieldsUpdatedThisGroup Method of the **Entity Object**
SetFieldValue Method of the **Entity Object**
Extracting Data About a Field in a Record

ValueChangedThisSetValue Method

Description:

Returns True if the field's value was modified by the most recent SetFieldValue call.

This method usually returns True only if this field was directly modified by a call to SetFieldValue. However, this method can also return true if the field was modified indirectly as a result of a hook.

This mechanism detects SetFieldValue calls only for the Entity object to which this field belongs. It ignores SetFieldValue calls that apply to other Entity objects.

VBScript Syntax:

fieldInfo.**ValueChangedThisSetValue**

Perl Syntax:

```
$fieldInfo->ValueChangedThisSetValue();
```

Identifier	Description
fieldInfo	A FieldInfo object, which contains information about one field of a user data record.
Return value	A Boolean that is True if the field's value was changed by the most recent call to SetFieldValue, otherwise False.

See Also:

GetFieldsUpdatedThisSetValue Method of the **Entity Object**
SetFieldValue Method of the **Entity Object**
FieldType Constants of the **Enumerated Constants**
Extracting Data About a Field in a Record

FieldInfos Object

The FieldInfos Object is a collection of **FieldInfo Object**s.

You can get the number of items in the collection by accessing the value in the Count Property. Use the Item Method to retrieve items from the collection.

Note: FieldInfos Objects and its property and methods are only applicable for usage with Perl script.

See Also:

FieldInfo Object

FieldInfos Object Property

The following list summarizes the FieldInfos Object property:

Property name	Description
Count Property	Returns the number of items in the collection.

Count Property

Description:

Returns the number of items in the collection. This property is read-only.

Perl Syntax:

```
$collection->Count();
```

Identifier	Description
collection	A FieldInfos collection object.
Data Type	A Long indicating the number of items in the collection object. This collection always contains at least one item.

See Also:

Item Method
Add Method

FieldInfos Object Methods

The following list summarizes the FieldInfos Object methods:

Method name	Description
Add Method	Adds an Attachment object to the collection.
Item Method	Returns the specified item in the collection.

Add Method

Description:

Adds a FieldInfo object to this FieldInfos collection.

This method adds a new FieldInfo object to the end of the collection. You can retrieve items from the collection using the **Item Method**.

Perl Syntax:

```
$fieldinfos->Add(fieldinfo);
```

Identifier	Description
fieldinfos	A FieldInfos collection object, representing the set of FieldInfos in one field of a record.
fieldinfo	The FieldInfo object to add to this collection.
Return value	A Boolean that is True if the FieldInfo object was added successfully, otherwise False.

See Also:

Count Property
Item Method

Item Method

Description:

Returns the specified item in the FieldInfos collection.

The argument to this method can be either a numeric index (itemNum) or a String (name).

Perl Syntax:

```
$fieldinfos->Item(itemNum);
$fieldinfos->ItemByName(name);
```

Identifier	Description
fieldinfos	A FieldInfos collection object, representing the set of FieldInfos in one field of a record.
itemNum	A Long that serves as an index into the collection. This index is 0-based so the first item in the collection is numbered 0, not 1.
name	A String that serves as a key into the collection. This string corresponds to the **GetName Method** of the desired FieldInfos.
Return value	The FieldInfo object at the specified location in the collection.

See Also:

Count Property
Add Method

Group Object

A Group object contains information about a single group of users.

Groups allow you to administer users as one or more groups, which is more convenient than administering each user separately. Use the Group object to get or modify the properties of a group, including the group's name and the databases to which it is subscribed. You can also add users to the group.

Changes you make to groups are immediately reflected in the schema repository (master database) but not the associated user databases. To update the user databases, use the user administration tools in ClearQuest Designer. See **Updating User Database Information**.

See Also:

Database Object
User Object

Group Object Properties

The following list summarizes the Group Object properties:

Property name	Access	Description
Active Property	Read/Write	Indicates whether or not the group is active.
SubscribedDatabases Property	Read-only	Returns the collection of databases to which this group is subscribed.
Name Property	Read/Write	Sets or returns the name of the group.
Users Property	Read-only	Returns the collection of users belonging to this group.

Active Property

Description:

Indicates whether or not the group is active.

This property can be returned or set.

Members of an inactive group are not allowed to access any databases using the group's attributes. Access to a database is permitted If the user belongs to another group that has access or if the user's account is specifically subscribed to the database.

VBScript Syntax:

```
group.Active [= value]
```

Perl Syntax:

```
$group->GetActive();
$group->SetActive(boolean_value);
```

Identifier	Description
group	A Group object, representing the set of groups associated with the current master database.
value	A Bool indicating whether or not the group is active.

See Also:

Active Property of the **User Object**

SubscribedDatabases Property

Description:

Returns the collection of databases to which this group is subscribed. This is a read-only property; it can be viewed but not set.

Each element in the returned collection is a Database object.

VBScript Syntax:

group.***Subscribed*****Databases**

Perl Syntax:

*$group->GetSubscribed***Databases();**

Identifier	Description
group	A Group object, representing the set of groups associated with the current master database.
Return value	A Databases collection object containing the databases to which this group is subscribed.

See Also:

SubscribeDatabase Method
UnsubscribeAllDatabases Method

SubscribedDatabases Property of the User object
Database Object
Databases Object
User Object

Name Property

Description:

Sets or returns the name of the group.

VBScript Syntax:

group.**Name [= *value*]**

Perl Syntax:

```
$group->GetGroups();
$group->SetGroups(string_of_group_name);
```

Identifier	Description
group	A Group object, representing the set of groups associated with the current master database.
value	A String containing the name of the group.

See Also:

Active Property

Users Property

Description:

Returns the collection of users belonging to this group.

This is a read-only property; it can be viewed but not set.

Each element in the returned collection is an User object. To add users to a group, use the AddUser method.

VBScript Syntax:

```
group.Users
```

Perl Syntax:

```
$group->GetUsers();
```

Identifier	Description
group	A Group object, representing the set of groups associated with the current master database.
Return value	An Users collection object containing the users belonging to this group.

See Also:

AddUser Method
User Object
Users Object

Group Object Methods

The following list summarizes the Group Object methods:

Method name	Description
AddUser Method	Adds a user to this group.
SubscribeDatabase Method	Subscribes this group to the specified database.
UnsubscribeAllDatabases Method	Unsubscribes the group from all databases.
UnsubscribeDatabase Method	Unsubscribes the group from the specified database.

AddUser Method

Description:

Adds a user to this group.

VBScript Syntax:

```
group.AddUser user
```

Perl Syntax:

```
$group->AddUser(user);
```

Identifier	Description
group	A Group object.
user	The User object corresponding to the user account.
Return value	None.

See Also:

User Object

SubscribeDatabase Method

Description:

Subscribes this group to the specified database.

Use this method to subscribe the group to additional databases. To unsubscribe the group, use the UnsubscribeDatabase method. To get a list of the databases to which the group belongs, get the collection of Database objects in the Databases property.

VBScript Syntax:

```
group.SubscribeDatabase database
```

Perl Syntax:

```
$group->SubscribeDatabase(database);
```

Identifier	Description
group	A Group object.
database	The Database object to which the group will be subscribed.
Return value	None.

See Also:

UnsubscribeAllDatabases Method
UnsubscribeDatabase Method
SubscribedDatabases Property

UnsubscribeAllDatabases Method

Description:

Unsubscribes the group from all databases.

Calling this method disassociates the group from all user databases in the master database. The group is still active.

VBScript Syntax:

```
group.UnsubscribeAllDatabases
```

Perl Syntax:

```
$group->UnsubscribeAllDatabases();
```

Identifier	Description
group	A Group object.
Return value	None.

See Also:

SubscribeDatabase Method
UnsubscribeDatabase Method
Active Property
SubscribedDatabases Property

UnsubscribeDatabase Method

Description:

Unsubscribes the group from the specified database.

Use this method to unsubscribe the group from a specific database. The group must be subscribed to the specified database before calling this method. To get a list of the databases to which the group belongs, get the collection of Database objects in the Databases property.

VBScript Syntax:

```
group.UnsubscribeDatabase database
```

Perl Syntax:

```
$group->UnsubscribeDatabase(database);
```

Identifier	Description
group	A Group object.
database	The Database object from which the group will be unsubscribed.
Return value	None.

See Also:

SubscribeDatabase Method
UnsubscribeAllDatabases Method
SubscribedDatabases Property

Groups Object

A Groups Object is a collection object for Group objects.

Groups allow you to administer users as one or more groups, which is more convenient than administering each user separately. Use the Group object to get or modify the properties of a group, including the group's name and the databases to which it is subscribed. You can also add users to the group.

Changes you make to groups are immediately reflected in the schema repository (master database) but not the associated user databases. To update the user databases, use the user administration tools in ClearQuest Designer.

See Also:

Group Object

Groups Object Property

The following list summarizes the Groups Object property:

Property name	Access	Description
Count Property	Read-only	Returns the number of items in the collection.

Count Property

Returns the number of items in the collection.

This is a read-only property; it can be viewed but not set.

VBScript Syntax:

collection.**Count**

Perl Syntax:

$collection->**Count();**

Identifier	Description
collection	A Groups collection object, representing the set of groups associated with the current master database.
Return value	A Long indicating the number of items in the collection object. This method returns zero if the collection contains no items.

See Also:

Item Method

Groups Object Method

The following list summarizes the Groups Object method:

Method name	Description
Item Method	Returns the item at the specified index in the collection.

Item Method

Returns the specified item in the collection.

VBScript Syntax:

```
collection.Item itemNum
collection.Item name
```

Perl Syntax:

```
$collection->Item(itemNum);
$collection->ItemByName(name);
```

Identifier	Description
collection	A Groups collection object, representing the set of groups associated with the current master database.
itemNum	A Long that serves as an index into the collection. This index is 0-based so the first item in the collection is numbered 0, not 1.
name	A String that serves as a key into the collection. This string corresponds to the unique key of the desired Group object.
Return value	The Group object at the specified location in the collection.

The argument to this method can be either a numeric index (*itemNum*) or a String (*name*).

See Also:

Count Property

Histories Object

In ClearQuest a defect (bug report) has history information associated with it. Each record has a history field, and this field can have multiple history entries. Each history entry is a line of text describing the modification. All history objects are read-only, because the history entries for a data record are created automatically by ClearQuest.

The Histories object is a container object that stores one or more History objects. A Histories object is always associated with a single HistoryField object.

See Also:

History Object
HistoryField Object
HistoryFields Object

Histories Object Property

The following list summarizes the Histories Object property:

Property name	Access	Description
Count Property	Read-only	Returns the number of items in the collection.

Count Property

Description:

Returns the number of items in the collection. This property is read-only.

VBScript Syntax:

collection.**Count**

Perl Syntax:

$collection->**Count();**

Identifier	Description
collection	A Histories collection object, representing the set of history entries in one history field of a record.
Data Type	A Long indicating the number of items in the collection object. This method returns zero if the collection contains no items.

See Also:

Item Method

Histories Object Method

The following list summarizes the Histories Object method:

Method name	Description
Item Method	Returns the specified item in the collection.

Item Method

Description:

Returns the specified item in the collection.

The argument to this method can be either a numeric index (*itemNum*) or a String (*name*).

VBScript Syntax:

```
collection.Item itemNum
collection.Item name
```

Perl Syntax:

```
$collection->Item(itemNum);
$collection->ItemByName(name);
```

Identifier	Description
collection	A Histories collection object, representing the set of history entries in one history field of a record.
itemNum	A Long that serves as an index into the collection. This index is 0-based so the first item in the collection is numbered 0, not 1.
name	A String that serves as a key into the collection. This string corresponds to the value of the desired History object.
Return value	The History object at the specified location in the collection.

See Also:

Count Property

History Object

In ClearQuest a defect (bug report) has history information associated with it. Each record has a history field, and this field can have multiple history entries. Each history entry is a line of text describing the modification. All history objects are read-only, because the history entries for a data record are created automatically by ClearQuest.

A History object contains a string that describes the modifications to the record.

The History object encapsulates the String that is displayed for one entry in a history field of a data record. The History object has only one property: the **Value Property**.

See Also:

Histories Object
HistoryField Object
HistoryFields Object

History Object Property

The following list summarizes the History Object properties:

Property name	Access	Description
Value Property	Read-only	Returns the String that contains information about one modification to a record, as displayed on one line of a history field.

Value Property

Description:

Returns the String that contains information about one modification to a record, as displayed on one line of a history field.

This a read-only property; it can be viewed but not set.

VBScript Syntax:

history.**Value**

Perl Syntax:

$history->**GetValue();**

Identifier	Description
History	A History object, representing one modification to a record.
Data Type	A String containing the history information. The String consists of several fields separated from each other by whitespace. In the current implementation, these fields consist of a timestamp, the user's name, the action name, the old state, and the new state.

See Also:

Histories Object
HistoryField Object
HistoryFields Object

HistoryField Object

In ClearQuest a defect (bug report) has history information associated with it. Each record has a history field, and this field can have multiple history entries. Each history entry is a line of text describing the modification. All history objects are read-only, because the history entries for a data record are created automatically by ClearQuest.

The HistoryField object represents a single history field in a record. A record can have multiple HistoryField objects, each of which includes a single Histories object.

The HistoryField object has one property: the Histories property. This property contains the set of History objects that describe the changes to the record.

See Also:

HistoryFields Property of the **Entity Object**
History Object
Histories Object
HistoryFields Object

HistoryField Object Properties

The following list summarizes the HistoryField Object properties:

Property name	Access	Description
DisplayNameHeader Property	Read-only	Returns the unique keys of the history items in this field.
FieldName Property	Read-only	Returns the name of the history field.
Histories Property	Read-only	Returns this history field's collection of History objects.

DisplayNameHeader Property

Description:

Returns the unique keys of the history items in this field.

This is a read-only property; it can be viewed but not set. The unique keys are set using ClearQuest Designer, not the ClearQuest API.

VBScript Syntax:

historyField.**DisplayNameHeader**

Perl Syntax:

$historyField->**GetDisplayNameHeader();**

Identifier	Description
field	A HistoryField object, representing one field of a record.
Data Type	For VB, a Variant containing an Array whose elements are Strings is returned. Each String contains the unique key of the corresponding item in the field's collection of Histories objects. For Perl, a reference to a String Array is returned.

See Also:

FieldName Property

FieldName Property

Description:

Returns the name of the history field.

This a read-only property; it can be viewed but not set. The field name is set using ClearQuest Designer, not the ClearQuest API.

VBScript Syntax:

historyField.**FieldName**

Perl Syntax:

$historyField->**GetFieldName();**

Identifier	Description
field	A HistoryField object, representing one field of a record.
Data Type	A String that contains the name of the field.

See Also:

DisplayNameHeader Property

Histories Property

Description:

Returns this history field's collection of History objects. This a read-only property; the value can be viewed but not set.

VBScript Syntax:

historyField.**Histories**

Perl Syntax:

```
$historyField->GetHistories();
```

Identifier	Description
historyField	A HistoryField object, representing one history field of a record.
Data Type	A Histories collection object, which itself contains a set of **History Object** objects.

See Also:

Histories Object

HistoryFields Object

In ClearQuest a defect (bug report) has history information associated with it. Each record has a history field, and this field can have multiple history entries. Each history entry is a line of text describing the modification. All history objects are read-only, because the history entries for a data record are created automatically by ClearQuest.

The HistoryFields object is the container object for all of the other objects. It represents all of the history fields associated with a record. There can be only one HistoryFields object associated with a record. This object contains one or more HistoryField objects.

The HistoryFields object's property and methods tell you how many items are in the collection and let you retrieve individual items. You cannot add, remove, or modify the items.

Every **Entity Object** has exactly one HistoryFields object. You cannot create a new HistoryFields object. However, you can retrieve the pre-existing HistoryFields object from a given Entity object by invoking Entity's **HistoryFields Property**.

See Also:

History Object
Histories Object
HistoryField Object

HistoryFields Object Properties

The following list summarizes the HistoryFields Object properties:

Property name	Access	Description
Count Property	Read-only	Returns the number of items in the collection.

Count Property

Description:

Returns the number of items in the collection. This property is read-only.

VBScript Syntax:

collection.**Count**

Perl Syntax:

$collection->**Count();**

Identifier	Description
collection	A HistoryFields collection object, representing all of the history fields of a record.
Data Type	A Long indicating the number of items in the collection object. This collection always contains at least one item.

See Also:

Item Method

HistoryFields Object Method

The following list summarizes the HistoryFields Object method:

Method name	Description
Item Method	Returns the specified item in the collection.

Item Method

Description:

Returns the specified item in the collection.

The argument to this method can be either a numeric index (*itemNum*) or a String (*name*).

VBScript Syntax:

```
collection.Item itemNum
collection.Item name
```

Perl Syntax:

```
$collection->Item(itemNum);
$collection->ItemByName(name);
```

Identifier	Description
collection	A HistoryFields collection object, representing all of the history fields of a record.
itemNum	A Long that serves as an index into the collection. This index is 0-based so the first item in the collection is numbered 0, not 1.
name	A String that serves as a key into the collection. This string corresponds to the field name of the desired HistoryField.
Return value	The HistoryField object at the specified location in the collection.

See Also:

HistoryField Object

HookChoices object

A HookChoices object represents the list of choices presented by a CHOICE_LIST hook.

The HookChoices object is a special object that is invisible except inside a CHOICE_LIST hook. This object has only one method, the **AddItem Method**, which you can use to add new items to the list.

The HookChoices object is stored in a variable called choices and you can only access it by that name.

Note: For Perl, use a Perl array to return a choice list. See the Examples of Hooks and Scripts.

See Also:

FieldInfo Object
Examples of Hooks and Scripts

HookChoices Object Methods

The following list summarizes the HookChoices Object methods:

Method name	Description
AddItem Method	Adds a new item to the list of choices created by a CHOICE_LIST hook.
Sort Method	Sorts the entries in the choice list.

AddItem Method

Description:

Adds a new item to the list of choices created by a CHOICE_LIST hook.

The pre-existing HookChoices object is stored in a variable called choices that is visible only within a CHOICE_LIST hook. In the syntax section of this method, choices is a variable name that must be typed literally; it is not a placeholder for an arbitrary name or expression. This is the only way to access a HookChoices object.

A CHOICE_LIST hook should call this method repeatedly to build up a list of choices for the user. The object contains no items when you first access it. Add items in the order in which you want them to appear, because the list is not automatically sorted.

There is no corresponding RemoveItem method. Duplicate items are not automatically removed, but empty values are.

VBScript Syntax:

choices.**AddItem** *newChoice*

Identifier	Description
choices	A special HookChoices object; see the remarks below.

Identifier	Description
newChoice	A String containing the new text to be added to the list of choices displayed to the user.
Return value	None.

See Also:

HookChoices object
Examples of Hooks and Scripts

Sort Method

Description:

Sorts the entries in the choice list.

The pre-existing HookChoices object is stored in a variable called choices that is visible only within a CHOICE_LIST hook. In the syntax section of this method, choices is a variable name that must be typed literally; it is not a placeholder for an arbitrary name or expression. This is the only way to access a HookChoices object.

VBScript Syntax:

choices.**Sort** [*sortAscending*]

Identifier	Description
choices	A special HookChoices object; see the remarks below.
sortAscending	An optional flag to indicate the sorting direction. The default value for this flag is true, which sorts the entries in ascending order. Specify False to sort the entries in descending order.
Return value	None.

See Also:

AddItem Method

Link Object

A Link object connects two Entity objects.

Links are the edges in the tree of duplicates. Links point both to the original record (the "parent") and to the duplicate record (the "child"). Both records must be state-based (as opposed to stateless). However, the parent and child do not need not be based on the same record type.

The methods of link allow you to retrieve:

- the parent and child record objects that are linked together.
- the ID strings for the parent and child.
- the EntityDef that is the template for the parent or child.
- the names of these EntityDefs

To create a Link object, use the **MarkEntityAsDuplicate Method** of the Entity object that is to become the duplicate. To delete the object, use the **UnmarkEntityAsDuplicate Method**.

See Also:

GetAllDuplicates Method of the **Entity Object**
GetDuplicates Method of the **Entity Object**
HasDuplicates Method of the **Entity Object**
IsDuplicate Method of the **Entity Object**

Link Object Methods

The following list summarizes the Link Object methods:

Method name	Description
GetChildEntity Method	Returns the Entity object that is the child (duplicate) in a pair of linked Entity objects.
GetChildEntityDef Method	Returns the **EntityDef Object** that is the template for the child (duplicate) in a pair of linked Entity objects.
GetChildEntityDefName Method	Returns the name of the EntityDef object that is the template for the child (duplicate) Entity object.
GetChildEntityID Method	Returns the ID String of the Entity object that is the child (duplicate) in a pair of linked Entity objects.
GetParentEntity Method	Returns the record that is the parent (original) in a pair of linked Entity objects.
GetParentEntityDef Method	Returns the **EntityDef Object** that is the template for the parent (original) in a pair of linked Entity objects.
GetParentEntityDefName Method	Returns the name of the EntityDef object that is the template for the parent (original) Entity object.
GetParentEntityID Method	Returns the ID String of the Entity object that is the parent (original) in a pair of linked Entity objects.

GetChildEntity Method

Description:

Returns the Entity object that is the child (duplicate) in a pair of linked Entity objects.

VBScript Syntax:

```
link.GetChildEntity
```

Perl Syntax:

```
$link->GetChildEntity();
```

Identifier	Description
link	A Link object, which connects a parent and child Entity object to each other.
Return value	The Entity object that is the child (duplicate).

See Also:

GetAllDuplicates Method of the **Entity Object**
GetDuplicates Method of the **Entity Object**
IsOriginal Method of the **Entity Object**
Updating Duplicate Records to Match the Parent Record

GetChildEntityDef Method

Description:

Returns the **EntityDef Object** that is the template for the child (duplicate) in a pair of linked Entity objects.

VBScript Syntax:

```
link.GetChildEntityDef
```

Perl Syntax:

```
$link->GetChildEntityDef();
```

Identifier	Description
link	A Link object, which connects a parent and child Entity object to each other.
Return value	An EntityDef object representing the record type of the child (duplicate) record.

See Also:

GetParentEntityDef Method
GetEntityDef Method of the **Session Object**

GetChildEntityDefName Method

Description:

Returns the name of the **EntityDef Object** that is the template for the child (duplicate) Entity object.

VBScript Syntax:

```
link.GetChildEntityDefName
```

Perl Syntax:

```
$link->GetChildEntityDefName();
```

Identifier	Description
link	A Link object, which connects a parent and child Entity object to each other.
Return value	A String containing the name of the EntityDef object that was used as a template for the child (duplicate) Entity object.

See Also:

GetParentEntityDefName Method
GetEntityDefName Method of the Entity object
Entity Object

GetChildEntityID Method

Description:

Returns the ID String of the Entity object that is the child (duplicate) in a pair of linked Entity objects.

VBScript Syntax:

link.**GetChildEntityID**

Perl Syntax:

$link->**GetChildEntityID**();

Identifier	Description
link	A Link object, which connects a parent and child Entity object to each other.
Return value	A String that identifies the child (duplicate) Entity object. This ID is the unique key returned by the **GetDisplayName Method** of Entity.

See Also:

GetParentEntityID Method
GetDisplayName Method of the **Entity Object**

GetParentEntity Method

Description:

Returns the record that is the parent (original) in a pair of linked Entity objects.

VBScript Syntax:

link.**GetParentEntity**

Perl Syntax:

$link->**GetParentEntity**();

Identifier	Description
link	A Link object, which connects a parent and child Entity object to each other.
Return value	The Entity object that is the parent (original).

See Also:

GetOriginal Method of the **Entity Object**
IsDuplicate Method of the **Entity Object**

GetParentEntityDef Method

Description:

Returns the **EntityDef Object** that is the template for the parent (original) in a pair of linked Entity objects.

VBScript Syntax:

```
link.GetParentEntityDef
```

Perl Syntax:

```
$link->GetParentEntityDef();
```

Identifier	Description
link	A Link object, which connects a parent and child Entity object to each other.
Return value	An EntityDef object representing the record type of the parent (original) record.

See Also:

GetChildEntityDef Method
GetEntityDef Method of the **Session Object**

GetParentEntityDefName Method

Returns the name of the **EntityDef Object** that is the template for the parent (original) Entity object.

VBScript Syntax:

```
link.GetParentEntityDefName
```

Perl Syntax:

```
$link->GetParentEntityDefName();
```

Identifier	Description
link	A Link object, which connects a parent and child Entity object to each other.
Return value	A String containing the name of the EntityDef object that was used as a template for the parent (original) Entity object.

See Also:

GetChildEntityDefName Method
GetEntityDefName Method of the **Entity Object**

GetParentEntityID Method

Description:

Returns the ID String of the Entity object that is the parent (original) in a pair of linked Entity objects.

VBScript Syntax:

```
link.GetParentEntityID
```

Perl Syntax:

```
$link->GetParentEntityID();
```

Identifier	Description
link	A Link object, which connects a parent and child Entity object to each other.
Return value	The String that identifies the parent (original) Entity object. This ID is the unique key returned by the **GetDisplayName Method** of Entity.

See Also:

GetChildEntityID Method
GetDisplayName Method of the **Entity Object**

Links Object

The Links Object is a collection of **Link Object**s.

You can get the number of items in the collection by accessing the value in the Count Property. Use the Item Method to retrieve items from the collection.

Note: Links Objects and its property and methods are only applicable for usage with Perl script.

See Also:

Link Object

Links Object Property

The following list summarizes the Links Object property:

Property name	Description
Count Property	Returns the number of items in the collection.

Count Property

Description:

Returns the number of items in the collection. This property is read-only.

Perl Syntax:

```
$links->Count();
```

Identifier	Description
links	A Links collection object.
Data Type	A Long indicating the number of items in the collection object. This collection always contains at least one item.

See Also:

Item Method
Add Method

Links Object Methods

The following list summarizes the Links Object methods:

Method name	Description
Add Method	Adds an Attachment object to the collection.
Item Method	Returns the specified item in the collection.

Add Method

Description:

Adds a Link object to this Links collection.

The new Link object is added to the end of the collection. You can retrieve items from the collection using the **Item Method**.

Perl Syntax:

```
$links->Add(linkobject);
```

Identifier	Description
links	A Links collection object.
linkobject	The link object to add to this collection
Return value	A Boolean that is True if the link object was added successfully, otherwise False.

See Also:

Count Property
Item Method

Item Method

Description:

Returns the specified item in the Links collection.

The argument to this method can be either a numeric index (itemNum) or a String (name).

Perl Syntax:

```
$links->Item(itemNum);
$links->ItemByName(name);
```

Identifier	Description
links	A Links collection object.
itemNum	A Long that serves as an index into the collection. This index is 0-based so the first item in the collection is numbered 0, not 1.
name	A String that serves as a key into the collection. This string corresponds to the **GetChildEntityDefName Method** of the desired Links.
Return value	The Link object at the specified location in the collection.

See Also:

Count Property
Add Method

OleMailMsg Object

An OleMailMsg object represents an e-mail message that you can send to your users.

The main purpose for the OleMailMsg object is to send e-mail messages from an action notification hook. You can use the methods of this object to specify the contents of the e-mail message including the recipients, sender, subject, and body text. You can then use the **Deliver Method** of this object to send the e-mail message.

This object does not support Perl. To create a new OleMailMsg object, you must use the VBScript CreateObject method as follows:

```
Dim mailmsg Set mailmsg = CreateObject("PAINET.MAILMSG")
```

When you have an OleMailMsg object, you can

- add recipients using the AddTo, AddCc, and AddBcc methods
- set the return address using the SetFrom method
- add a subject line using the SetSubject method
- set the body text of the e-mail message using the SetBody and MoreBody methods

OleMailMsg Object Methods

Method name	Description
AddBcc Method	Add the e-mail address of a blind carbon-copy recipient to the mail message.
AddCc Method	Add the e-mail address of a carbon-copy recipient to the mail message.
AddTo Method	Add the e-mail address of a primary recipient to the mail message.
ClearAll Method	Resets the contents of the mail message object.
Deliver Method	Delivers the mail message.
MoreBody Method	Appends additional body text to the mail message.
SetBody Method	Sets the body text of the mail message.
SetFrom Method	Sets the return address of the mail message.
SetSubject Method	Sets the subject line of the e-mail message.

AddBcc Method

Description:

Add the e-mail address of a blind carbon-copy recipient to the mail message.

Call this method once for every e-mail address you want to add to the blind-carbon copy list. Each person you add to this list receives a copy of the e-mail message. However, the e-mail addresses of people on this list are not included anywhere in the e-mail message.

VBScript Syntax:

```
OleMailMsg.AddBcc newAddress
```

Perl Syntax:

```
$OleMailMsg->AddBcc(newAddress);
```

Identifier	Description
OleMailMsg	An OleMailMsg object, representing the mail message to be sent.
newAddress	A String containing the e-mail address of the recipient.
Return value	None.

See Also:

AddCc Method
AddTo Method
ClearAll Method
SetFrom Method
SetSubject Method

AddCc Method

Description:

Add the e-mail address of a carbon-copy recipient to the mail message.

Call this method once for every e-mail address you want to add to the carbon-copy list. Each person you add to this list receives a copy of the e-mail message. Addresses on the carbon-copy list appear in the header of the e-mail message.

VBScript Syntax:

```
OleMailMsg.AddCc newAddress
```

Perl Syntax:

```
$OleMailMsg->AddCc(newAddress);
```

Identifier	Description
OleMailMsg	An OleMailMsg object, representing the mail message to be sent.

Identifier	Description
newAddress	A String containing the e-mail address of the recipient.
Return value	None.

See Also:

AddBcc Method
AddTo Method
ClearAll Method
SetFrom Method
SetSubject Method

AddTo Method

Description:

Add the e-mail address of a primary recipient to the mail message.

Call this message once for every person you want to add to the recipient list. Each person you add to this list receives a copy of the e-mail message. Addresses on the recipient list appear in the header of the e-mail message.

VBScript Syntax:

```
OleMailMsg.AddTo newAddress
```

Perl Syntax:

```
$OleMailMsg->AddTo(newAddress);
```

Identifier	Description
OleMailMsg	An OleMailMsg object, representing the mail message to be sent.
newAddress	A String containing the e-mail address of the recipient.
Return value	None.

See Also:

AddBcc Method
AddCc Method
ClearAll Method
SetFrom Method
SetSubject Method

ClearAll Method

Description:

Resets the contents of the mail message object.

This method removes the intended recipients (including Cc and Bcc recipients), the subject line, and the body text of the message. This method also resets the return address to the e-mail address of the submitter of the record.

VBScript Syntax:

```
OleMailMsg.ClearAll
```

Perl Syntax:

```
$OleMailMsg->ClearAll();
```

Identifier	Description
OleMailMsg	An OleMailMsg object, representing the mail message to be sent.
Return value	None.

See Also:

AddBcc Method
AddCc Method
AddTo Method
MoreBody Method
SetBody Method

SetFrom Method
SetSubject Method

Deliver Method

Description:

Delivers the mail message.

After calling this method, you can make changes to the object without affecting the e-mail message that was just sent.

VBScript Syntax:

OleMailMsg.**Deliver**

Perl Syntax:

$*OleMailMsg***->Deliver();**

Identifier	Description
OleMailMsg	An OleMailMsg object, representing the mail message to be sent.
Return value	A Long indicating the success or failure of the delivery. A value of 1 indicates that the message was sent successfully. A value of 0 indicates that the message could not be delivered.

See Also:

AddBcc Method
AddCc Method
AddTo Method
ClearAll Method
MoreBody Method
SetBody Method
SetFrom Method
SetSubject Method

MoreBody Method

Description:

Appends additional body text to the mail message.

Use this method to add body text above and beyond what you added with the **SetBody Method**. You can call this method as many times as you like. Each call to this method appends the specified text to the end of the message content.

This method does not add end-of-line characters or any other formatting characters when appending the text; you must add these characters yourself to the string you pass in to the bodyText parameter.

VBScript Syntax:

OleMailMsg.**MoreBody** *bodyText*

Perl Syntax:

$*OleMailMsg***->MoreBody(***bodyText***);**

Identifier	Description
OleMailMsg	An OleMailMsg object, representing the mail message to be sent.
bodyText	A String containing the body text to add to the mail message.
Return value	None.

See Also:

ClearAll Method
SetBody Method

SetBody Method

Description:

Sets the body text of the mail message.

This method replaces any existing body text with the string you specify. If you added any body text with previous calls to SetBody or MoreBody method, that text will be lost.

This method does not add end-of-line characters or any other formatting characters when appending the text; you must add these characters yourself to the string you pass in to the bodyText parameter

VBScript Syntax:

```
OleMailMsg.SetBody bodyText
```

Perl Syntax:

```
$OleMailMsg->SetBody(bodyText);
```

Identifier	Description
OleMailMsg	An OleMailMsg object, representing the mail message to be sent.
bodyText	A String containing the main body text of the mail message.
Return value	None.

See Also:

ClearAll Method
MoreBody Method

SetFrom Method

Description:

Sets the return address of the mail message.

If you do not call this method, ClearQuest automatically sets the return address to the e-mail address of the submitter of the record. You can call this method only once to add a return address to the e-mail message.

VBScript Syntax:

```
OleMailMsg.SetFrom returnAddress
```

Perl Syntax:

```
$OleMailMsg->SetFrom(returnAddress);
```

Identifier	Description
OleMailMsg	An OleMailMsg object, representing the mail message to be sent.
returnAddress	A String containing the e-mail address to add to the From field of the mail message.
Return value	None.

See Also:

AddBcc Method
AddCc Method
AddTo Method
ClearAll Method
SetSubject Method

SetSubject Method

Description:

Sets the subject line of the e-mail message.

Call this method once to set text for the subject line. Subsequent calls to this method replace the existing subject line with the new string.

VBScript Syntax:

```
OleMailMsg.SetSubject subjectText
```

Perl Syntax:

```
$OleMailMsg->SetSubject(subjectText);
```

Identifier	Description
OleMailMsg	An OleMailMsg object, representing the mail message to be sent.
subjectText	A String containing the subject text to add to the message.
Return value	None.

See Also:

AddBcc Method
AddCc Method
AddTo Method
ClearAll Method
SetFrom Method

QueryDef Object

A QueryDef object defines the parameters for a query, which is used to retrieve specific records from a database.

A QueryDef object contains a query expression and a list of display fields. The query expression defines the search parameters for the query and can contain a complex set of conditional statements. To run the query, you must create a **ResultSet Object** and call its **Execute Method**. (You can use the Session object's **BuildResultSet Method** to create the ResultSet object.) The ResultSet object uses the list of display fields in the QueryDef object to summarize the search results.

To create a QueryDef object,

1. Call the Session object's **BuildQuery Method**. The BuildQuery methods returns an QueryDef object with display fields and filters undefined.

2. Add the filters and fields for your query to the QueryDef object.

   To create a query that returns all of the records in the database, you create the simplest QueryDef object by to the query one field that calls the QueryDef object's **BuildField Method**.

   You can add filters and nodes to a QueryDef object to create more complex queries. The nodes of a QueryDef object consist of one or more **QueryFilterNode Object**s, each containing one or more filters. Nodes group together each of their filters under a single boolean operator. You use the QueryDef object's **BuildFilterOperator Method** to create the root node in this tree. After that, you use the methods of QueryFilterNode to define the remaining nodes and filters. The filters themselves can use other comparison operators to test the relationship of a field to the specified data.

   Note: You can also construct a query from a raw SQL query string using the Session object's **BuildSQLQuery Method**. However, this technique does not create a QueryDef object.

See Also:

BuildQuery Method of the Session object
ResultSet Object
Session Object
Building Queries for Defects and Users

QueryDef Object Properties

The following list summarizes the QueryDef Object properties:

Property name	Access	Description
IsAggregated Property	Read-only	Returns a Boolean indicating whether any fields of the query are aggregated.
IsDirty Property	Read-only	Returns a Boolean indicating whether the query has changed.
IsMultiType Property	Read-only	Returns a Boolean indicating whether the QueryDef object is multitype.
Name Property	Read/Write	Sets or returns the name associated with the query.
QueryType Property	Read-only	Returns an Integer indicating list, report, or chart.
SQL Property	Read/Write	Sets or returns the SQL string associated with the query.

IsAggregated Property

Description:

Returns a Bool indicating whether any fields of the query are aggregated.

Aggregated fields are grouped together for display in the resulting query or chart. This property is read-only.

VBScript Syntax:

querydef.**IsAggregated**

Perl Syntax:

$querydef->**GetIsAggregated();**

Identifier	Description
querydef	A QueryDef object.
Return value	True if any of the fields in the query are aggregated, otherwise False.

See Also:

SQL Property

IsDirty Property

Description:

Returns a Boolean indicating whether the query has changed.

A QueryDef object is considered dirty if any of its fields or filters have changed since the last time it was saved.

VBScript Syntax:

querydef.**IsDirty**

Perl Syntax:

$querydef->**GetIsDirty();**

Identifier	Description
querydef	A QueryDef object.
Return value	True if the query has changed, otherwise False.

See Also:

Save Method

IsMultiType Property

Description:

Returns a Boolean indicating whether a given querydef has the property of being multitype.

One use case for this method is to support querying similar record types (for example, defects and enhancement requests) in a single query. This method can be used in conjunction with **GetEntityDefFamily Method** and **GetEntityDefFamilyNames Method**.

VBScript Syntax:

querydef.**IsMultiType**

Perl Syntax:

$*querydef***->IsMultiType();**

Identifier	Description
querydef	A QueryDef object.
Return value	True if the object is multitype.

See Also:

GetEntityDefFamily Method
GetEntityDefFamilyNames Method

Name Property

Sets or returns the name associated with the query.

VBScript Syntax:

querydef.**Name** *[value]*

Perl Syntax:

```
$querydef->GetName();
$querydef->SetName(newName);
```

Identifier	Description
querydef	A QueryDef object.
value	A String containing the name of the query.

See Also:

Save Method

QueryType Property

Description:

Returns an integer indicating whether the saved query has the property of being a list, a report, or a chart.

VBScript Syntax:

```
querydef.QueryType
```

Perl Syntax:

```
$querydef->GetQueryType();
```

Identifier	Description
querydef	A QueryDef object.
Return value	An Integer indicating whether a saved query is a list (**_LIST_QUERY**), a report (**_REPORT_QUERY**), or query (**_CHART_QUERY**).

See Also:

QueryType Constants

SQL Property

Description:

Sets or returns the SQL string associated with the query.

If you assign a value to this property, the QueryDef object uses your string instead of the terms you have built using other methods of this object.

If you get the value of this property, the QueryDef object returns the SQL string that will be executed when the query is run. If you had assigned a SQL string to this property earlier, that string is returned; otherwise, this method generates a SQL string from the terms that have been added to the QueryDef object so far.

VBScript Syntax:

```
querydef.SQL [= Value]
```

Perl Syntax:

```
$querydef->GetSQL();
$querydef->SetSQL(string_of_SQL_statements);
```

Identifier	Description
querydef	A QueryDef object.
value	A String containing the SQL that will be executed when the query is run.

See Also:

BuildSQLQuery Method of the Session object
Session Object

QueryDef Object Methods

Method name	Description
BuildField Method	Selects a field to include in the query's search results.
BuildFilterOperator Method	Creates the top-level **QueryFilterNode Object** for the query.
Save Method	Saves the query to the specified file.

BuildField Method

Description:

Selects a field to include in the query's search results.

Before you run a query, you must specify at least one field to display in the search results summary. You must call this method once to specify each field that you want to display. The ResultSet object displays the fields from left to right in the order in which you added them to the QueryDef object. In other words, each time you call this method, you add the specified field to the end of the list; you cannot change this ordering.

Because you associate a QueryDef object with an EntityDef object when you call the **BuildQuery Method**, the field_name parameter must contain the name of a valid field in that EntityDef object. To obtain valid values for the field_name argument, you can query the EntityDef object by calling its **GetFieldDefNames Method**.

You can call BuildField either before or after constructing the query expression (the tree of filter nodes).

VBScript Syntax:

```
querydef.BuildField field_name
```

Perl Syntax:

```
$querydef->BuildField(field_name);
```

Identifier	Description
querydef	A QueryDef object.
field_name	A String identifying a valid field of the associated EntityDef object.
Return value	None.

See Also:

BuildFilterOperator Method
BuildQuery Method of the Session object
GetFieldDefNames Method of the EntityDef object
EntityDef Object
ResultSet Object
Session Object
Building Queries for Defects and Users

BuildFilterOperator Method

Description:

Creates the top-level **QueryFilterNode Object** for the query.

This QueryDef method is the starting-point for building a query expression. You must call this method to obtain the first filter in the query expression. From this filter, you can construct additional filters to specify the criteria you want. The query expression is constructed as a tree of Boolean operators. The tree is not necessarily binary; you can add more than two conditions to a filter node.

VBScript Syntax:

```
querydef.BuildFilterOperator bool_operator
```

Perl Syntax:

```
$querydef->BuildFilterOperator(bool_operator);
```

Identifier	Description
querydef	A QueryDef object.
bool_operator	A Long whose value is one of the **BoolOp Constants**.
Return value	The newly created QueryFilterNode object.

Example:

```
(submitter = jjones OR submitter = clopez OR submitter = kwong)
AND
     submit_date < 01/03/2000
```

In this expression, the top-level Boolean operator is the AND operator. To start constructing this query expression, you use this method to create the filter that has the top-level operator:

```
filterNode1 = myQueryDef. BuildFilterOperator (AD_BOOL_OP_AND)
```

You use this method just once to construct the root of the tree. To continue adding filters, you call the methods of the returned QueryFilterNode objects. For example, to complete the previous expression, you would write the following code:

```
filterNode1.BuildFilter ('submit_date', AD_COMP_OP_LT,
'1997-11-19')
filterNode2 = filterNode1.BuildFilterOperator (AD_BOOL_OP_OR)
filterNode2.BuildFilter ('submitter', AD_COMP_OP_EQ, 'jjones')
filterNode2.BuildFilter ('submitter', AD_COMP_OP_EQ, 'clopez')
filterNode2.BuildFilter ('submitter', AD_COMP_OP_EQ, 'kwong')
```

More-complicated expressions are created by recursively attaching more nodes as needed. For more information, see the **QueryFilterNode Object**.

If a node contains only one condition, the value of the bool_operator parameter is irrelevant. For example, if the entire query expression is `'SUBMITTER = JJONES'`, you could construct the query expression as follows:

```
' You could use either AD_BOOL_OP_AND or AD_BOOL_OP_OR for this
' expression since there is only one condition.
filterNode = myQueryDef.BuildFilterOperator (AD_BOOL_OP_AND)
```

```
filterNode.BuildFilter ('submitter', AD_COMP_OP_EQ, 'jjones')
```

Note: It is perfectly legal to create a QueryDef object that has no filtering (in other words, no query expression). In this case, all of the records in the database are retrieved.

See Also:

BuildFilter Method of the QueryFilterNode object
BuildFilterOperator Method of the QueryFilterNode object
BoolOp Constants
QueryFilterNode Object
Building Queries for Defects and Users

Save Method

Description:

Saves the query to the specified file.

VBScript Syntax:

querydef.**Save** *fileName*

Perl Syntax:

$*querydef***->Save(***fileName***);**

Identifier	Description
querydef	A QueryDef object.
fileName	A String containing the name of the file.
Return value	A Bool containing the value True if the query was successfully saved, otherwise False.

See Also:

Name Property

QueryFilterNode Object

A QueryFilterNode object represents one node in the query-expression tree.

A query expression consists of one or more QueryFilterNode objects arranged hierarchically. The root node is created by the QueryDef object's **BuildFilterOperator Method**. The remaining nodes are all instances of the QueryFilterNode class. Each node consists of one or more filters and a Boolean operator (specified using the **BoolOp Constants**).

To add a filter to a node, you call the node's **BuildFilter Method**. Using this method, you specify a field and a specific value to compare, and you specify the comparison operator to use (one of the **CompOp Constants**). Although the node uses a Boolean operator, you can add any number of filters to a node with the BuildFilter method.

You can also add other nodes. Using the **BuildFilterOperator Method** of QueryFilterNode, you can add nodes just as if they were an additional filter. By nesting nodes in this fashion, you can create complex query expressions with the nodes and filters forming a tree.

See Also:

BuildQuery Method of the **Session Object**
ResultSet Object
QueryDef Object
Building Queries for Defects and Users

QueryFilterNode Object Methods

The following list summarizes the QueryFilterNode Object properties:

Method name	Description
BuildFilter Method	Adds a comparison operand to the node.
BuildFilterOperator Method	Creates a nested **QueryFilterNode Object** that contains the specified Boolean operator.

BuildFilter Method

Description:

Adds a comparison operand to the node.

The operand created by this method consists of a field name, a comparison operator, and a value. When the query is run, the value in the field is compared to the specified value using the given comparison operator. The comparison yields a boolean value, which the node uses in its own boolean comparison.

The value argument is a [VB Variant or] Perl reference to a string array to allow you to specify an Array of values when appropriate. For example, if you wanted to find the defects submitted between December 1 and December 15, 2000. you could construct the following filter:

VBScript Syntax:

```
node.BuildFilter field_name, comparison_operator, value
```

Perl Syntax:

```
$node->BuildFilter(field_name, comparison_operator, value);
```

Identifier	Description
node	A QueryFilterNode object, representing one node in the query expression.
field_name	A String containing the name of a valid field in the **EntityDef Object** on which the current **QueryDef Object** is based.
comparison_operator	A Long whose value is one of the CompOp enumeration constants.
value	[A VB Variant or] Perl reference to a string array containing the value that you want to compare to the value in the specified field.
Return value	None.

Example (in VBScript):

```
Dim dateRange as Variant(2)
dateRange(0) = 2000-12-01
dateRange(1) = 2000-12-15
node.BuildFilter("submit_date", AD_COMP_OP_IN, dateRange)
```

Example (in Perl):

```
@dateRange = (“2000-12-01”, “2000-12-15”);
$node->BuildFilter("submit_date", CQPerlExt::_COMP_OP_IN,
      \@dateRange);
```

Query expressions are not limited to being binary trees; you can call this method as many times as you want for a given QueryFilterNode object. See the example given for QueryDef's **BuildFilterOperator Method**.

To obtain valid values for the field_name argument, call the **GetFieldDefNames Method** of the EntityDef object upon which the query was based.

See Also:

BuildFilterOperator Method
BuildFilterOperator Method of the **QueryDef Object**
Building Queries for Defects and Users
Notation Conventions for VBScript
Notation Conventions for Perl

BuildFilterOperator Method

Description:

Creates a nested **QueryFilterNode Object** that contains the specified Boolean operator.

This method creates a nested node (or subnode) in the query expression. The newly created node operates at the same level as the filters in the QueryFilterNode object specified in the node parameter and is subject to the same conditions. You can add filters to the newly created node using the **BuildFilter Method** just as you would for any other node.

VBScript Syntax:

```
node.BuildFilterOperator bool_operator
```

Perl Syntax:

```
$node->BuildFilterOperator(bool_operator);
```

Identifier	Description
node	The QueryFilterNode object to which the newly created node will be attached.
bool_operator	A Long whose value is one of the BoolOp enumeration constants.
Return value	The newly created QueryFilterNode object.

See Also:

BuildFilter Method
BuildFilterOperator Method of the **QueryDef Object**
Building Queries for Defects and Users

ReportMgr Object

The ReportMgr object provides an interface for generating reports.

Remarks:

You can use this object to write external applications to execute reports defined in the ClearQuest workspace. You can also use the methods of this object to check the status and parameters of a report.

1. Associate the WORKSPACE object with a Session object.

   This association makes it possible to access reports in the ClearQuest workspace.

2. Get a ReportMgr object by calling the **GetReportMgr Method** of the **WORKSPACE Object**.

   When you call GetReportMgr, you must specify the name of the report you want to execute. ClearQuest associates that report with the returned ReportMgr object. To execute a different report, you must create a new ReportMgr object.

3. Set the name of the file in which to put the report data by calling the SetHTMLFileName method.

4. Execute the report by calling the **ExecuteReport Method**.

See Also:

WORKSPACE Object

ReportMgr Object Methods

The following list summarizes the ReportMgr Object methods:

Method name	Description
ExecuteReport Method	Executes the report and generates the resulting HTML file.
GetQueryDef Method	Returns the QueryDef object associated with the report.
GetReportPrintJobStatus Method	Returns the current status of the print job.
SetHTMLFileName Method	Sets the output file name for the report.

ExecuteReport Method

Description:

Executes the current report and generates the resulting HTML file.

This method executes the current report and puts the resulting data into the current destination file. You specify the report to execute when you create the ReportMgr object. To set the destination file, you must call the SetHTMLFileName method prior to calling this method.

ClearQuest outputs the report data in HTML format. You can view this data using an HTML browser.

VBScript Syntax:

```
reportMgr.ExecuteReport
```

Perl Syntax:

```
$reportMgr->ExecuteReport();
```

Identifier	Description
reportMgr	The ReportMgr object associated with the current session.
Return value	None.

See Also:

GetReportMgr Method
SetHTMLFileName Method

GetQueryDef Method

Description:

Returns the QueryDef object associated with the report.

You can use the returned QueryDef object to get information about the query that was used to generate the report.

VBScript Syntax:

```
reportMgr.GetQueryDef
```

Perl Syntax:

```
$reportMgr->GetQueryDef();
```

Identifier	Description
reportMgr	The ReportMgr object associated with the current session.
Return value	The QueryDef object associated with the report.

See Also:

QueryDef Object

GetReportPrintJobStatus Method

Description:

Returns the current status of the print job.

This method indicates whether or not the report writing tool has finished generating the report. Once this method returns crPrintingCompleted, you can open the generated report file and begin examining the data.

VBScript Syntax:

reportMgr.**GetReportPrintJobStatus**

Perl Syntax:

$reportMgr->**GetReportPrintJobStatus**();

Identifier	Description
reportMgr	The ReportMgr object associated with the current session.

Return value	An INT corresponding to one of the `tagPrintJobStatus` enumeration constants.
	`enum tagPrintJobStatus {`
	`  crPrintingNotInitialized = 0,`
	`  crPrintingNotStarted,`
	`  crPrintingInProgress,`
	`  crPrintingCompleted,`
	`  crPrintingFailed,`
	`  crPrintingCancelled,`
	`  crPrintingHalted`
	`}`

See Also:

SetHTMLFileName Method

SetHTMLFileName Method

Description:

Sets the output file name for the report.

You must call this method before calling the ExecuteReport method to set the location of the report output file. You can specify path information in the htmlPath parameter to put the report in a specific location.

VBScript Syntax:

```
reportMgr.SetHTMLFileName htmlPath
```

Perl Syntax:

```
$reportMgr->SetHTMLFileName(htmlPath);
```

Identifier	Description
reportMgr	The ReportMgr object associated with the current session.
htmlPath	A String containing the pathname for the report file.
Return value	None.

See Also:

ExecuteReport Method

ResultSet Object

You can use a ResultSet object to execute a query and browse the query results.

When you create queries using the **QueryDef Object**, you must create a corresponding ResultSet object to run the query and obtain the results. Each ResultSet object is customized for the query it is running. The ResultSet object contains data structures that organize data from the query into rows and columns, where each row represents a single data record and each column represents one field from that data record. After running the query, you can navigate (move) from row to row, and from column to column, to obtain the data you want.

See Also:

BuildResultSet Method of the Session object
QueryDef Object
Session Object
Running a Query and Reporting on its Result Set

ResultSet Object Methods

Method name	Description
AddParamValue Method	Assigns one or more values to a parameter.
ClearParamValues Method	Clears all values associated with a parameter.
Execute Method	Runs the query and fills the result set with data.
GetColumnLabel Method	Returns the heading text for the specified column.
GetColumnType Method	Returns the type of data stored in the specified column.
GetColumnValue Method	Returns the value stored in the specified column of the current row.
GetNumberOfColumns Method	Returns the number of columns in each row of the result set.
GetNumberOfParams Method	Returns the number of parameters in this query.
GetParamChoiceList Method	Returns a list of permitted values for the parameter.
GetParamComparisonOperator Method	Returns the comparison operator associated with the parameter.
GetParamFieldType Method	Returns the field type of the parameter.
GetParamLabel Method	Returns the name of the parameter.
GetParamPrompt Method	Returns the prompt string displayed to the user for the given parameter.
GetRowEntityDefName Method	Based on the result set, this method returns a String with the record type (EntityDef) name of the current row.
GetSQL Method	Returns the SQL string that expresses the query.
LookupPrimaryEntityDefName Method	Returns the name of the EntityDef object on which the query is based.
MoveNext Method	Moves the **cursor** to the next record in the data set.

AddParamValue Method

Description:

Assigns one or more values to a parameter.

The parameter number is a Long whose value is between 1 and the total number of parameters.

VBScript Syntax:

```
resultset.AddParamValue param_number, value
```

Perl Syntax:

```
$resultset->AddParamValue(param_number, value);
```

Identifier	Description
resultset	A ResultSet object, representing the rows and columns of data resulting from a query.
param_number	A Long identifying the parameter. See Remarks.
value	For VB, a Variant containing the value for the parameter. For Perl, a String containing the value for the parameter.
Return value	None.

See Also:

ClearParamValues Method

ClearParamValues Method

Description:

Clears all values associated with a parameter.

The parameter number is a Long whose value is between 1 and the total number of parameters.

VBScript Syntax:

resultset.**ClearParamValues** *param_number*

Perl Syntax:

$*resultset***->ClearParamValues(***param_number***);**

Identifier	Description
resultset	A ResultSet object, representing the rows and columns of data resulting from a query.
param_number	A Long identifying the parameter. See Remarks.
Return value	None.

See Also:

AddParamValue Method

Execute Method

Description:

Runs the query and fills the result set with data.

This method runs the query and creates the resulting data set in the database. Because the resulting data set could be huge, this method does not copy the data set into the program's memory. When this method returns, the cursor is positioned before the first record. You must call the **MoveNext Method** before retrieving the first record's values. To retrieve values from a record, use the **GetColumnValue Method**.

After executing the query, it is legal to get the SQL for the query by invoking the **GetSQL Method**.

You may call this method more than once. For example, you might want to rerun the query if the data could have changed since the last time, or if you made changes to the database yourself.

VBScript Syntax:

```
resultset.Execute
```

Perl Syntax:

```
$resultset->Execute();
```

Identifier	Description
resultset	A ResultSet object, representing the rows and columns of data resulting from a query.
Return value	None.

See Also:

GetColumnValue Method
GetSQL Method
MoveNext Method
BuildResultSet Method of the Session object
Session Object
Running a Query and Reporting on its Result Set

GetColumnLabel Method

Description:

Returns the heading text for the specified column.

Columns are numbered from 1 to N, not 0 to N-1.

VBScript Syntax:

```
resultset.GetColumnLabel columnNum
```

Perl Syntax:

```
$resultset->GetColumnLabel(columnNum);
```

Identifier	Description
resultset	A ResultSet object, representing the rows and columns of data resulting from a query.
columnNum	A Long that specifies the desired index (1-based) into the array of columns.
Return value	A String containing the column label.

See Also:

GetColumnType Method
GetColumnValue Method
GetNumberOfColumns Method
Running a Query and Reporting on its Result Set

GetColumnType Method

Description:

Returns the type of data stored in the specified column.

This method returns the underlying database type, rather than a FieldType, because the result of a complex SQL query can include a column that does not correspond to a field of a record.

Columns are numbered from 1 to N, not 0 to N-1.

VBScript Syntax:

resultset.**GetColumnType** *columnNum*

Perl Syntax:

$*resultset***->GetColumnType(***columnNum***);**

Identifier	Description
resultset	A ResultSet object, representing the rows and columns of data resulting from a query.
columnNum	A Long that specifies the desired index (1-based) into the array of columns.
Return value	A Long whose value is a CType enumeration constant representing this column's underlying storage type in the database.

See Also:

GetColumnLabel Method
GetColumnValue Method
GetNumberOfColumns Method

GetColumnValue Method

Description:

Returns the value stored in the specified column of the current row.

If the cursor is not positioned at a record, or the field's value has not been set, the returned Variant will be NULL. To advance the cursor to the next row, you must call the **MoveNext Method**.

In the current version of the ClearQuest API, the Variant is always set as a string, but future versions might let you initialize the Variant to the most appropriate native type.

Columns are numbered from 1 to N, not 0 to N-1.

VBScript Syntax:

```
resultset.GetColumnValue columnNum
```

Perl Syntax:

```
$resultset->GetColumnValue(columnNum);
```

Identifier	Description
resultset	A ResultSet object, representing the rows and columns of data resulting from a query.
columnNum	A Long that specifies the desired index (1-based) into the array of columns.
Return value	For VB, a Variant that contains the value stored in the specified column of the current row is returned. For Perl, a String containing the column value is returned

See Also:

GetColumnLabel Method
GetColumnType Method
GetNumberOfColumns Method
MoveNext Method
Running a Query and Reporting on its Result Set

GetNumberOfColumns Method

Description:

Returns the number of columns in each row of the result set.

VBScript Syntax:

resultset.**GetNumberOfColumns**

Perl Syntax:

$*resultset***->GetNumberOfColumns();**

Identifier	Description
resultset	A ResultSet object, representing the rows and columns of data resulting from a query.
Return value	A Long indicating the number of columns in the result set.

See Also:

GetColumnLabel Method
GetColumnType Method
GetColumnValue Method
Running a Query and Reporting on its Result Set

GetNumberOfParams Method

Description:

Returns the number of parameters in this query.

VBScript Syntax:

resultset.**GetNumberOfParams**

Perl Syntax:

$*resultset***->GetNumberOfParams();**

Identifier	Description
resultset	A ResultSet object, representing the rows and columns of data resulting from a query.
Return value	A Long indicating the number of parameters in the query.

See Also:

GetParamChoiceList Method
GetParamComparisonOperator Method
GetParamFieldType Method
GetParamLabel Method
GetParamPrompt Method

GetParamChoiceList Method

Description:

Returns a list of permitted values for the parameter.

The parameter number is a Long whose value is between 1 and the total number of parameters.

VBScript Syntax:

resultset.**GetParamChoiceList** *param_number*

Perl Syntax:

$*resultset***->GetParamChoiceList(***param_number***);**

Identifier	Description
resultset	A ResultSet object, representing the rows and columns of data resulting from a query.
param_number	A Long identifying the parameter. See Remarks.
Return value	For VB, a Variant containing the list of permitted values for the parameter is returned. If the Variant is empty, there are no restrictions on the parameter values. For Perl, a reference to a String Array is returned.

See Also:

GetNumberOfParams Method
GetParamComparisonOperator Method
GetParamFieldType Method
GetParamLabel Method
GetParamPrompt Method

GetParamComparisonOperator Method

Description:

Returns the comparison operator associated with the parameter.

The parameter number is a Long whose value is between 1 and the total number of parameters.

VBScript Syntax:

resultset.**GetParamComparisonOperator** *param_number*

Perl Syntax:

$*resultset***->GetParamComparisonOperator(***param_number***);**

Identifier	Description
resultset	A ResultSet object, representing the rows and columns of data resulting from a query.
param_number	A Long identifying the parameter. See Remarks.
Return value	A Long indicating the comparison operator for the parameter. The value corresponds to a value in the CompOp enumerated type.

See Also:

GetNumberOfParams Method
GetParamChoiceList Method
GetParamFieldType Method
GetParamLabel Method
GetParamPrompt Method
CompOp Constants

GetParamFieldType Method

Description:

Returns the field type of the parameter.

The parameter number is a Long whose value is between 1 and the total number of parameters.

VBScript Syntax:

resultset.**GetParamFieldType** *param_number*

Perl Syntax:

```
$resultset->GetParamFieldType(param_number);
```

Identifier	Description
resultset	A ResultSet object, representing the rows and columns of data resulting from a query.
param_number	A Long identifying the parameter. See Remarks.
Return value	A Long indicating the field type of the parameter. The value corresponds to a value in the FieldType enumerated type.

See Also:

GetNumberOfParams Method
GetParamChoiceList Method
GetParamComparisonOperator Method
GetParamLabel Method
GetParamPrompt Method
FieldType Constants

GetParamLabel Method

Description:

Returns the name of the parameter.

The parameter number is a Long whose value is between 1 and the total number of parameters.

The name of the parameter is the name associated directly with the field and may not correspond to the prompt displayed to the user.

VBScript Syntax:

```
resultset.GetParamLabel param_number
```

Perl Syntax:

```
$resultset->GetParamLabel(param_number);
```

Identifier	Description
resultset	A ResultSet object, representing the rows and columns of data resulting from a query.
param_number	A Long identifying the parameter. See Remarks.
Return value	A String containing the name of the parameter.

See Also:

GetNumberOfParams Method
GetParamChoiceList Method
GetParamComparisonOperator Method
GetParamFieldType Method
GetParamPrompt Method

GetParamPrompt Method

Description:

Returns the prompt string displayed to the user for the given parameter.

The parameter number is a Long whose value is between 1 and the total number of parameters.

VBScript Syntax:

resultset.**GetParamPrompt** *param_number*

Perl Syntax:

$*resultset***->GetParamPrompt(***param_number***);**

Identifier	Description
resultset	A ResultSet object, representing the rows and columns of data resulting from a query.

Identifier	Description
param_number	A Long identifying the parameter. See Remarks.
Return value	A String containing the prompt string displayed to the user.

See Also:

GetNumberOfParams Method
GetParamChoiceList Method
GetParamComparisonOperator Method
GetParamFieldType Method
GetParamLabel Method

GetRowEntityDefName Method

Description:

Based on the result set, this method returns a String with the record type (EntityDef) name of the current row.

For a single-type query, the record type associated with the row is always the primary entitydef. For multitype query, the entitydef can vary row by row. For example, defect versus enhancement.

VBScript Syntax:

resultset.**GetRowEntityDefName**

Perl Syntax:

$*resultset***->GetRowEntityDefName();**

Identifier	Description
resultset	A ResultSet object, representing the rows of data that meet query criteria.
Return value	A String containing the name of the record type of the specified row.

Examples:

See **Running a Query Against More than One Record Type**.

See Also:

IsMultiType Property

GetSQL Method

Description:

Returns the SQL string that expresses the query.

A ResultSet can be based on either a QueryDef object or an SQL string. In either case, you can retrieve the SQL commands that express the query. It is legal to invoke this method either before or after you run the query by calling the **Execute Method**.

VBScript Syntax:

```
resultset.GetSQL
```

Perl Syntax:

```
$resultset->GetSQL();
```

Identifier	Description
resultset	A ResultSet object, representing the rows and columns of data resulting from a query.
Return value	A String containing the raw SQL that expresses the query upon which this ResultSet is based.

See Also:

Execute Method

LookupPrimaryEntityDefName Method

Description:

Returns the name of the **EntityDef Object** on which the query is based.

A ResultSet can based on either a **QueryDef Object** or an SQL string. A query that uses a QueryDef object must also have an associated EntityDef object, and thus this method returns the name of that object.

VBScript Syntax:

resultset.**LookupPrimaryEntityDefName**

Perl Syntax:

$*resultset*->**LookupPrimaryEntityDefName()**;

Identifier	Description
resultset	A ResultSet object, representing the rows and columns of data resulting from a query.
Return value	A String containing the name of the EntityDef object. If the query was defined using the BuildSQLQuery method of Session, the resulting String is Empty.

See Also:

BuildQuery Method of the Session object
EntityDef Object
QueryDef Object
Session Object
Running a Query and Reporting on its Result Set

MoveNext Method

Description:

Moves the **cursor** to the next record in the data set.

The **Execute Method** positions the cursor before the first record in the result set (not at the first record). Before you can retrieve the data from the first record, you must call this method to advance the cursor to that record.

VBScript Syntax:

```
resultset.MoveNext
```

Perl Syntax:

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

Identifier	Description
resultset	A ResultSet object, representing the rows and columns of data resulting from a query.
Return value	A Long whose value is a FetchStatus enumeration constant indicating whether the cursor movement was successful.

See Also:

Execute Method
GetColumnValue Method
Running a Query and Reporting on its Result Set

Schema Object

A Schema object contains information about a particular schema.

A Schema object represents a single schema in a master database. Use Schema objects to refer to schemas and to get a list of the revisions of the schema that are available.

Note: The API does not allow you to create new schemas or modify existing schemas. Schemas must be created or modified by using ClearQuest Designer. You can get a list of schemas defined in the schema repository (master database) by accessing the **Schemas Property** of the **AdminSession Object**.

See Also:

Schemas Property of the **AdminSession Object**
Schemas Object
SchemaRev Object
SchemaRevs Object

Schema Object Properties

The following list summarizes the Schema Object properties:

Property name	Access	Description
Name Property	Read-only	Returns a string containing the name of the schema.
SchemaRevs Property	Read-only	Returns the collection containing schema revisions.

Name Property

Description:

Returns the name of this schema.

This is a read-only property; it can be viewed but not set.

VBScript Syntax:

schema.**Name**

Perl Syntax:

$schema->**GetName**();

Identifier	Description
schema	A Schema object.
Return value	A String containing the name of this schema.

See Also:

SchemaRevs Property

SchemaRevs Property

Description:

Returns the schema revisions associated with this schema.

This is a read-only property; it can be viewed but not set.

Each element in the returned collection is a SchemaRev object.

VBScript Syntax:

schema.**SchemaRevs**

Perl Syntax:

$schema->**GetSchemaRevs**();

Identifier	Description
schema	A Schema object.
Return value	A SchemaRevs collection object containing the schema revisions associated with this schema.

See Also:

SchemaRev Object

SchemaRev Object

A SchemaRev object contains information about a single schema revision, including information about its packages.

Schema revisions identify a particular version of a schema. You use schema revisions when creating and updating databases.

To set the schema revision of a new database, create the database, then call the database object's SetInitialSchemaRev method.

To change the schema revision of an existing database, call the database object's Upgrade method.

To discover which packages and package revisions apply to the current user database, use the GetEnabledPackageRevs Method and the GetEnabledEntityDefs Method.

See Also:

SetInitialSchemaRev Method of the **Database Object**
Upgrade Method of the **Database Object**
Schema Object

SchemaRev Object Properties

The following list summarizes the SchemaReve Object properties:

Property name	Access	Description
Description Property	Read-only	Returns a description of this schema revision.
RevID Property	Read-only	Returns the version ID of this schema revision.
Schema Property	Read-only	Returns the schema to which this revision belongs.

Description Property

Description:

Returns a description of this schema revision.

This is a read-only property; it can be viewed but not set.

The descriptive text is the comment string entered by the user when the schema was checked in.

VBScript Syntax:

schemaRev.**Description**

Perl Syntax:

*$schemaRev->***GetDescription**();

Identifier	Description
schemaRev	A SchemaRev object.
Return value	A String containing the description of the schema revision.

See Also:

Schema Object

RevID Property

Description:

Returns the version number of this schema revision.

This is a read-only property; it can be viewed but not set.

VBScript Syntax:

schemaRev.**RevID**

Perl Syntax:

$schemaRev->**GetRevID**();

Identifier	Description
schemaRev	A SchemaRev object.
Return value	A Long indicating the version number associated with this schema revision.

See Also:

Schema Object

Schema Property

Description:

Returns the schema to which this revision belongs.

This is a read-only property; it can be viewed but not set.

VBScript Syntax:

schemaRev.**Schema**

Perl Syntax:

```
$schemaRev->GetSchema();
```

Identifier	Description
schemaRev	A SchemaRev object.
Return value	A Schema object corresponding to the schema to which this revision belongs.

See Also:

Schema Object

SchemaRev Object Methods

The following list summarizes the SchemaRev Object methods:

GetEnabledEntityDefs Method	Returns the EntityDefs collection object enabled in the current schema for a given package revision.
GetEnabledPackageRevs Method	Returns the package name and revision string for the current package revision in an EntityDefs collection object.

GetEnabledEntityDefs Method

Description:

Returns the EntityDefs collection object enabled in the current schema for a given package revision.

Use with GetEnabledPackageRevs Method to discover which packages and package revisions apply to the current user database.

VBScript Syntax:

```
schemaRev.GetEnabledEntityDefs packName, rev
```

Perl Syntax:

```
$schemaRev->GetEnabledEntityDefs(packName, rev);
```

Identifier	Description
packName	A String that specifies the package name.
rev	A String that specifies the package revision.
Return value	The EntityDefs object for the current package revision.

See Also:

GetEnabledPackageRevs Method

GetEnabledPackageRevs Method

Description:

Returns a collection object representing the packageRev set that is enabled in the current revision of the schema.

Use with GetEnabledEntityDefs Method to discover which packages and package revisions apply to the current user database.

You can also call this method from the Session object, in which case the schema revision is already known when you log onto the user database.

VBScript Syntax:

schemaRev.**GetEnabledPackageRevs** *PackageName, RevString*

Perl Syntax:

$schemaRev->**GetEnabledPackageDefs**(*PackageName, RevString);*

Identifier	Description
PackageName	Name of the package.
RevString	Represents the revision of the package.
Return values	The collection object of the packageRev set.

See Also:

GetEnabledEntityDefs Method
GetEnabledPackageRevs Method in **Session Object**

SchemaRevs Object

A SchemaRevs object is a collection object for SchemaRev objects.

You can get the number of items in the collection by accessing the value in the **Count Property**. Use the **Item Method** to retrieve items from the collection.

See Also:

Schemas Property of the **AdminSession Object**
SchemaRev Object
Schemas Object

SchemaRevs Object Property

The following list summarizes the SchemaRevs Object properties:

Property	Access	Description
Count Property	Read-only	Returns the number of items in the collection.

Count Property

Description:

Returns the number of items in the collection. This is a read-only property; it can be viewed but not set.

VBScript Syntax:

collection.**Count**

Perl Syntax:

$collection->**Count();**

Identifier	Description
collection	A SchemaRevs collection object, representing the set of schema revisions associated with the current master database.
Return value	A Long indicating the number of items in the collection object. This method returns zero if the collection contains no items.

See Also:

Item Method

SchemaRevs Object Method

The following list summarizes the SchemaRevs Object method:

Method	Access	Description
Item Method	Read-only	Returns the item at the specified index in the collection.

Item Method

Description:

Returns the specified item in the collection.

The argument to this method can be either a numeric index (*itemNum*) or a String (*name*).

VBScript Syntax:

```
collection.Item itemNum
collection.Item name
```

Perl Syntax:

```
$collection->Item(itemNum);
$collection->ItemByName(name);
```

Identifier	Description
collection	A SchemaRevs collection object, representing the set of schema revisions associated with the current master database.
itemNum	A Long that serves as an index into the collection. This index is 0-based so the first item in the collection is numbered 0, not 1.
name	A String that serves as a key into the collection. This string corresponds to the unique key of the desired SchemaRev object.
Return value	The SchemaRev object at the specified location in the collection.

See Also:

Count Property

Schemas Object

A Schemas object is a collection object for Schema objects.

You can get the number of items in the collection by accessing the value in the **Count Property**. Use the **Item Method** to retrieve items from the collection.

See Also:

Schema Object
SchemaRev Object

Schemas Object Property

The following list summarizes the Schemas Object property:

Property name	Access	Description
Count Property	Read-only	Returns the number of items in the collection.

Count Property

Description:

Returns the number of items in the collection.

This is a read-only property; it can be viewed but not set.

VBScript Syntax:

collection.**Count**

Perl Syntax:

$collection->**Count();**

Identifier	Description
collection	A Schemas collection object, representing the set of schemas associated with the current master database.
Return value	A Long indicating the number of items in the collection object. This method returns zero if the collection contains no items.

See Also:

Item Method

Schemas Object Method

The following list summarizes the Schemas Object method:

Method name	Description
Item Method	Returns the item at the specified index in the collection.

Item Method

Description:

Returns the specified item in the collection.

The argument to this method can be either a numeric index (*itemNum*) or a String (*name*).

VBScript Syntax:

```
collection.Item itemNum
collection.Item name
```

Perl Syntax:

```
$collection->Item(itemNum);
$collection->ItemByName(name);
```

Identifier	Description
collection	A Schemas collection object, representing the set of schemas associated with the current master database.
itemNum	A Long that serves as an index into the collection. This index is 0-based so the first item in the collection is numbered 0, not 1.
name	A String that serves as a key into the collection. This string corresponds to the unique key of the desired Schema object.
Return value	The Schema object at the specified location in the collection.

See Also:

Count Property

Session Object

Users access a ClearQuest database through a Session object. This object provides methods for logging on to the database, viewing records (entities), and creating queries. You can also use the **Session Object** to store variables for the session.

Getting a Session Object

The Session object is the entry point for accessing ClearQuest databases. If you are writing an external application, you must create a Session object and use it to log on to a database. Once you have logged on to a database, you can use the Session object to

- create new records or queries
- edit existing records
- view information about the database

For script hooks (VBScript and Perl), ClearQuest creates a Session object for your hooks automatically when the user logs on to the database. The session object is available through the entity object. In the context of a hook, to get a session object from an entity object, use the following syntax.

Scripting Language	Syntax for making a call to an Entity object in a hook
VBScript	`set currentSession = GetSession` VBScript hooks implicitly associate the Entity object with the current record.
Perl	When writing ClearQuest hooks, a session object is created and made available through the context variable $session. You do not need to perform any explicit call to create it. If you need a session object in some other context (such as when writing a standalone program) you can get a session object by using the following syntax: $session=$entity->GetSession();

For external applications, you must create a Session object manually. If you want to use the adminSession object, the same rule applies.

Language Example	Syntax for manually creating the Session object (or the AdminSession object) in an external application
Visual Basic	set currentSession = CreateObject("CLEARQUEST.SESSION") set adminSession = CreateObject("CLEARQUEST.ADMINSESSION")
Perl	$CQSession = CQPerlExt::CQSession_Build(); $AdminSession= CQPerlExt::CQAdminSession_Build(); When you are done with the object, destroy it: CQAdminSession::Unbuild($currentSession); CQAdminSession::Unbuild($currentAdminSession);

Logging on to a database

To protect your databases from unauthorized users, ClearQuest requires that you log on to a database before accessing its records. For hooks, this user authentication is handled automatically by the ClearQuest client application. However, external applications must log on programmatically by using the Session object.

To determine which database to log on to, and to perform the log on, follow these steps:

1. Get a list of the databases associated with a schema repository by calling the **GetAccessibleDatabases Method** of the Session object.

   This method returns a collection of DatabaseDesc objects, each of which contains information about a single user database.

2. Get the name of the database and enter an empty string (`""`) for the database set (the set of databases to which a database belongs) by using the methods of the **DatabaseDesc Object**.

3. Log on to the database by calling the **UserLogon Method** of the Session object.

You must have a valid login ID and password to log on to the database. As soon as you log on, you can start looking through records and creating queries. (See the description of the **UserLogon Method** for usage information.)

Note: If your external application uses Session methods, the general rule is to call UserLogon before calling other Session methods. However, there are two Session methods that you can call before calling UserLogon: **GetAccessibleDatabases Method**, **OutputDebugString Method**, **UnmarkEntityAsDuplicate Method**.

See Also:

Getting Session and Database Information

Session Object Property

The following list summarizes the Session Object properties:

Property name	Access	Description
NameValue Property	Read/Write	Gets or sets the value of one of this property's named variables.

NameValue Property

Description:

Gets or sets the value of one of this property's named variables.

Use this property to get and set the values for session-wide variables. Because this property consists of an array of values, you must specify the name of the variable you are interested in. If you set the value of a variable that does not exist, it is created with the specified value assigned to it. If you try to get the value of a variable that does not exist, an empty Variant is returned.

VBScript Syntax:

```
session.NameValue name
session.NameValue name = newValue
```

Perl Syntax:

```
$session->GetNameValue(name);
$session->SetNameValue(name, newValue);
```

Identifier	Description
session	The Session object that represents the current database-access session.
name	A String containing the name of the variable to get or set.

Identifier	Description
newValue	For VB, a reference to a Variant containing the new value for the variable. For Perl, a string containing the new value.
Return value	For VB, a Variant when getting a value is returned. Nothing when setting a value. For Perl, a string is returned with GetNameValue(). Nothing is returned with SetNameValue().

Example (in VBScript):

```
set sessionObj = GetSession

' Get the old value of the session variable "foo"
fooValue = sessionObj.NameValue("foo")

' Set the new value of "foo"
sessionObj.NameValue "foo", "bar"
```

Example (in Perl):

```
$sessionObj = $entity->GetSession();

# Get the old value of the session variable "foo"
$fooValue = $sessionObj->GetNameValue("foo") ;

# Set the new value of "foo"
$sessionObj->SetNameValue("foo", "bar");
```

See Also:

HasValue Method

Session Object Methods

The following list summarizes the Session Object methods:

Method name	Description
BuildEntity Method	Creates a new record of the specified type and begins a Submit action.
BuildQuery Method	Creates and returns a new QueryDef object for the specified record type.
BuildResultSet Method	Creates and returns a result set that can be used to run a query.
BuildSQLQuery Method	Creates and returns a ResultSet object using a raw SQL string.
DeleteEntity Method	Deletes the specified record from the current database.
EditEntity Method	Performs the specified action on a record and makes the record available for editing.
FireRecordScriptAlias Method	Calls the action that calls the hook script; use to simulate a user choosing an action that launches a hook.
GetAccessibleDatabases Method	Returns a list of databases that are available for the specified user to log in to.
GetAuxEntityDefNames Method	Returns an array of Strings, each of which corresponds to the name of one of the **schema** stateless record types.
GetDefaultEntityDef Method	Returns the schema's default EntityDef object.
GetEnabledEntityDefs Method	Returns the EntityDefs collection object enabled in the current schema for a given package revision.
GetEnabledPackageRevs Method	Returns a collection object representing the packageRev set that is enabled for the current revision of the schema.
GetEntity Method	Returns the specified record.
GetEntityByDbId Method	Returns the record with the specified database ID.
GetEntityDef Method	Returns the specified EntityDef object if it is a family.
GetEntityDefFamily Method	Returns the requested EntityDef object if it is a family.

Method name	Description
GetEntityDefFamilyNames Method	Returns an array containing the requested EntityDef family names.
GetEntityDefNames Method	Returns an array containing the names of the record types in the current database's **schema**.
GetInstalledMasters Method	Returns the list of registered database sets and master databases.
GetQueryEntityDefNames Method	Returns an array containing the names of the record types that are suitable for use in queries.
GetReqEntityDefNames Method	Returns an array containing the names of the state-based record types in the current database's **schema**.
GetServerInfo Method	Returns the name of the session's OLE server.
GetSessionDatabase Method	Returns general information about the database that is being accessed in the current session.
GetSubmitEntityDefNames Method	Returns an array containing the names of the record types that are suitable for use in creating a new record.
GetUserEmail Method	Returns the electronic mail address of the user who is logged in for this session.
GetUserFullName Method	Returns the full name of the user who is logged in for this session.
GetUserGroups Method	Returns a list of the groups to which the current user belongs.
GetUserLoginName Method	Returns the name that was used to log in for this session.
GetUserMiscInfo Method	Returns miscellaneous information about the user who is logged in for this session.
GetUserPhone Method	Returns the telephone number of the user who is logged in for this session.
GetWorkSpace Method	Returns the session's WORKSPACE object.
HasValue Method	Returns a Bool indicating whether the specified session variable exists.
IsMetadataReadonly Method	Returns a boolean indicating whether the session's metadata is read-only.

Method name	Description
MarkEntityAsDuplicate Method	Modifies the specified record to indicate that it is a **duplicate** of another record.
OpenQueryDef Method	Loads a query from a file.
OutputDebugString Method	Specifies a message that can be displayed by a debugger or a similar tool.
UnmarkEntityAsDuplicate Method	Removes the indication that the specified record is a **duplicate** of another record.
UserLogon Method	Log in as the specified user for a database session.

BuildEntity Method

Description:

Creates a new record of the specified type and begins a "submit" action.

This method creates a new record and initiates a "submit" action, thus enabling you to begin editing the record's contents. (You do not need to call **EditEntity Method** to make the record editable.) You can assign values to the new record's fields using the **SetFieldValue Method** of the returned Entity object. When you are done updating the record, use the **Validate Method** and **Commit Method** of the Entity object to validate and commit any changes you made to the record, respectively.

The name you specify in the entitydef_name parameter must also correspond to an appropriate record type in the schema. To obtain a list of legal names for entitydef_name, use the **GetSubmitEntityDefNames Method**.

VBScript Syntax:

```
session.BuildEntity entitydef_name
```

Perl Syntax:

```
$session->BuildEntity(entitydef_name);
```

Identifier	Description
session	The Session object that represents the current database-access session.
entitydef_name	A String containing the name of the **EntityDef Object** to use as a template when creating the record.
Return value	A new **Entity Object** that was built using the named EntityDef object as a template.

Example (in VBScript):

```
set sessionObj = GetSession

' Create a new "defect" record
set entityObj = sessionObj.BuildEntity("defect")
```

Example (in Perl):

```
$sessionobj = $entity->GetSession();
# Create a new "defect" record

$entityobj = $sessionobj->BuildEntity("defect");
```

See Also:

EditEntity Method
GetEntity Method
GetSubmitEntityDefNames Method
Commit Method of the Entity object
SetFieldValue Method of the Entity object
Validate Method of the Entity object
Entity Object
Managing Records (Entities) that are Stateless and Stateful

BuildQuery Method

Description:

Creates and returns a new QueryDef object for the specified record type.

You can use the returned QueryDef object to build a query for searching records whose record type matches the specified EntityDef. Before you can perform the search, you must add at least one field to query's display list by calling the **BuildField Method** of the QueryDef object. You can also add filters to the QueryDef object to specify the search criteria. For more information on specifying this information, see the description and methods of the **QueryDef Object**.

The name you specify in the entitydef_name parameter must correspond to an appropriate record type in the schema. To obtain a list of legal names for entitydef_name, use the **GetQueryEntityDefNames Method**.

Before you can run the query, you must associate the QueryDef object with a **ResultSet Object**. See the **BuildResultSet Method** for information on how to do this.

VBScript Syntax:

session.**BuildQuery** *entitydef_name*

Perl Syntax:

$*session***->BuildQuery(***entitydef_name***);**

Identifier	Description
session	The Session object that represents the current database-access session.
entitydef_name	A String containing the name of the **EntityDef Object** to use as a template when creating the record.
Return value	A new **QueryDef Object**. This object contains no filters or build fields.

Example (in VBScript):

```
set sessionObj = GetSession

' Create a query for "defect" records
set queryDefObj = sessionObj.BuildQuery("defect")
```

Example (in Perl):

```
$sessionObj = $entity->GetSession();
```

```
# Create a query for "defect" records
$queryDefObj = $sessionObj->BuildQuery("defect");
```

See Also:

BuildResultSet Method
GetEntityDefNames Method
GetQueryEntityDefNames Method
BuildField Method of the **QueryDef Object**
ResultSet Object
Building Queries for Defects and Users

BuildResultSet Method

Description:

Creates and returns a result set that can be used to run a query.

This method creates a ResultSet object for the specified QueryDef object. You can then use the returned ResultSet object to run the query and store the resulting data.

Do not call this method until you have added all of the desired fields and filters to the QueryDef object. This method uses the information in the QueryDef object to build the set of data structures needed to store the query data. If you add new fields or filters to the QueryDef object after calling this method, the ResultSet object will not reflect the new additions. To run the query and fetch the resulting data, you must subsequently call the ResultSet object's **Execute Method**.

Note: To obtain the QueryDef object that you pass to this method, you must call the **BuildQuery Method**. To construct a ResultSet object directly from a raw SQL query string, use the **BuildSQLQuery Method**.

VBScript Syntax:

session.**BuildResultSet** *querydef*

Perl Syntax:

$*session***->BuildResultSet(***querydef***);**

Identifier	Description
session	The Session object that represents the current database-access session.
querydef	A **QueryDef Object** that defines the desired query.
Return value	A **ResultSet Object** suitable for eventual execution of the query.

Example (in VBScript):

```
set sessionObj = GetSession
' Create a query and result set to search for all records.

set queryDefObj = sessionObj.BuildQuery("defect")
queryDefObj.BuildField("id")
set resultSetObj = sessionObj.BuildResultSet(queryDefObj)
```

Example (in Perl):

```
$sessionObj = $entity->GetSession();
# Create a query and result set to search for all records.

$queryDefObj = $sessionObj->BuildQuery("defect");
$queryDefObj->BuildField("id");
$resultSetObj = $sessionObj->BuildResultSet($queryDefObj);
```

See Also:

BuildQuery Method
BuildSQLQuery Method
Execute Method of the **ResultSet Object**
QueryDef Object
Building Queries for Defects and Users
Running a Query and Reporting on its Result Set

BuildSQLQuery Method

Description:

Creates and returns a ResultSet object using a raw SQL string.

We recommend you use the ClearQuest API to define a query and filter(s), as opposed to writing raw SQL.

Like **BuildResultSet Method**, this method creates a ResultSet object that you can use to run a query. Unlike BuildResultSet, this method uses a raw SQL string instead of a QueryDef object to build the data structures of the ResultSet object. Do not call this method until you have completely constructed the SQL query string.

Like **BuildResultSet Method**, this method generates the data structures needed to store the query data but does not fetch the data. To run the query and fetch the resulting data, you must call the ResultSet object's **Execute Method**.

Unlike BuildResultSet, BuildSQLQuery makes no use of a QueryDef object, so the query defined by the SQL string cannot be manipulated before constructing the ResultSet.

VBScript Syntax:

session.**BuildSQLQuery** *SQL_string*

Perl Syntax:

$*session***->BuildSQLQuery(***SQL_string***);**

Identifier	Description
session	The Session object that represents the current database-access session.
SQL_string	A String containing the raw SQL commands for the query.
Return value	A **ResultSet Object** suitable for running the query.

Example (in VBScript):

```
set sessionObj = GetSession

' Create a SQL string to find all records and display their
' ID and headline fields

sqlString  = "select T1.id,T1.headline from defect T1 where
   T1.dbid <> 0"
set resultSetObj = sessionObj.BuildSQLQuery(sqlString)
```

Example (in Perl):

```
$sessionobj = $entity->GetSession();

# Create a SQL string to find all records and display their
# ID and headline fields

$sqlString = "select T1.id,T1.headline from defect T1 where
   T1.dbid <> 0";
$resultSetObj = $sessionobj->BuildSQLQuery($sqlString);
```

See Also:

BuildQuery Method
ResultSet Object
Building Queries for Defects and Users

DeleteEntity Method

Description:

Deletes the specified record from the current database.

When you call this method, ClearQuest deletes the specified entity using the action whose name you specified in the deleteActionName parameter. This action name must correspond to a valid action in the schema and it must be legal to perform the action on the specified entity.

VBScript Syntax:

session.**DeleteEntity** *entity, deleteActionName*

Perl Syntax:

$*session***->DeleteEntity(***entity, deleteActionName***);**

Identifier	Description
session	The Session object that represents the current database-access session.
entity	The Entity object corresponding to the record to be deleted.

Identifier	Description
deleteActionName	A String containing the name of the action to use when deleting the entity.
Return value	If there was a problem deleting the entity, this method returns a String containing the error message, otherwise this method returns an empty string ("").

Example (in VBScript):

```
set sessionObj = GetSession

' Delete the record whose ID is "BUGDB00000042" using the "delete"
' action
set objToDelete = sessionObj.GetEntity("defect", "BUGDB00000042")
sessionObj.DeleteEntity objToDelete, "delete"
```

Example (in Perl):

```
$sessionobj = $entity->GetSession();

# Delete the record whose ID is "BUGDB00000010" using the "delete"
# action
$objtodelete = $sessionobj->GetEntity("defect", "BUGDB00000010");
$sessionobj->DeleteEntity($objtodelete,"delete");
```

See Also:

BuildEntity Method
EditEntity Method
GetEntity Method
Entity Object

EditEntity Method

Description:

Performs the specified action on a record and makes the record available for editing.

The Entity object you specify in the entity parameter must have been previously obtained by calling the **GetEntityByDbId Method** or **GetEntity Method** method, or by

running a query. If you created the Entity object using the **BuildEntity Method** and have not yet committed it to the database, the object is already available for editing.

To obtain a list of legal values for the edit_action_name parameter, call the **GetActionDefNames Method** of the appropriate EntityDef object.

After calling this method, you can call the methods of the Entity object to modify the fields of the corresponding record. When you are done editing the record, validate it and commit your changes to the database by calling the Entity object's Validate and Commit methods, respectively.

VBScript Syntax:

session.**EditEntity** *entity, edit_action_name*

Perl Syntax:

$*session***->EditEntity(***entity, edit_action_name***);**

Identifier	Description
session	The Session object that represents the current database-access session.
entity	The **Entity Object** corresponding to the record that is to be edited.
edit_action_name	A String containing the name of the action to initiate for editing. (For example: "modify" or "resolve")
Return value	None.

Example (in VBScript):

```
set sessionObj = GetSession

' Edit the record whose ID is "BUGDB00000010" using the "modify" '
' action
set objToEdit = sessionObj.GetEntity("defect", "BUGDB00000010")
sessionObj.EditEntity objToEdit, "modify"
```

Example (in Perl):

```
$sessionobj = $entity->GetSession();
```

```
# Edit the record whose ID is "BUGDB00000010" using the "modify"
# action
$objtoedit = $sessionobj->GetEntity("defect", "BUGDB00000010");
$sessionobj->EditEntity($objtoedit,"modify");
```

See Also:

BuildEntity Method
GetEntity Method
GetEntityByDbId Method
Commit Method of the **Entity Object**
Validate Method of the **Entity Object**
GetActionDefNames Method of the **Entity Object**
ActionType Constants
Updating Duplicate Records to Match the Parent Record
Managing Records (Entities) that are Stateless and Stateful

FireRecordScriptAlias Method

Description:

Calls the action that calls the hook script.

You can use this method to programmatically simulate a user choosing an action that launches a hook. The method wraps a named hook script in an action.

VBScript Syntax:

*session.***FireRecordScriptAlias** *entity, editActionName*

Perl Syntax:

$*session***->FireRecordScriptAlias**(*entity, editActionName)*;

Identifier	Description
entity	The entity must be an entity object previously returned by BuildEntity, GetEntityById, or GetEntityByDisplayName.
editActionName	The edit action name must be the name of a valid action as defined in the metadata.The action type must be RECORD_SCRIPT_ALIAS or this method fails.
Return value	A String containing the script return value determined by the hook.

See Also:

_RECORD_SCRIPT_ALIAS constant in **ActionType Constants**
Commit Method
EditEntity Method
Validate Method
FireNamedHook Method
Notation Conventions for VBScript
Notation Conventions for Perl

GetAccessibleDatabases Method

Description:

Returns a list of databases that are available for the specified user to log in to.

This method returns only the databases that the specified user is allowed to log in to. If the user_login_name parameter contains an empty String, this method returns a list of all of the databases associated with the specified master database.

You can examine each DatabaseDescription object to get the corresponding database's name and other information needed to log in to it.

VBScript Syntax:

```
session.GetAccessibleDatabases master_db_name, user_login_name,
database_set
```

Perl Syntax:

$*session***->GetAccessibleDatabases(***master_db_name, user_login_name, database_set***);**

Identifier	Description
session	The Session object that represents the current database-access session.
master_db_name	A String that specifies the **schema repository**.
user_login_name	A String that specifies the user's login.
database_set	A String that specifies the **database set** in which to look for accessible databases. By default, this argument should contain the empty String.
Return value	For VB, a Variant containing an Array whose elements are Variants of type **DatabaseDesc Object** is returned. For Perl, a **DatabaseDescs Object** collection is returned.

Example (in VBScript):

```
set sessionObj = GetSession

' Get the list of databases in the
' "ClearQuest 1.1" master database set.
databases = sessionObj.GetAccessibleDatabases("ClearQuest 1.1",
   "", "")
for each db in databases
  ' Get the name of the database
  dbName = db.GetDatabaseName
  Next
```

See Also:

UserLogon Method

GetDatabaseName Method of the **DatabaseDesc Object**

GetAuxEntityDefNames Method

Description:

Returns an array of Strings, each of which corresponds to the name of one of the **schema** stateless record types.

The Array is never empty; at a minimum it will contain the names "users", "groups", "attachments", and "history" which correspond to the system-defined stateless record types.

Once you have the name of a stateless record type, you can retrieve the **EntityDef Object** for that record type by calling the **GetEntityDef Method**.

VBScript Syntax:

session.**GetAuxEntityDefNames**

Perl Syntax:

$session->GetAuxEntityDefNames();

Identifier	Description
session	The Session object that represents the current database-access session.
Return value	For VB, a Variant containing an Array of Strings is returned. Each String contains the name of a stateless record type. For Perl, a reference to a String Array is returned.

Example (in VBScript):

```
set sessionObj = GetSession

' Get the list of names for the stateless record types.
entityDefNames = sessionObj.GetAuxEntityDefNames

' Iterate over the non-system stateless record types
for each name in entityDefNames
  if name <> "users" And name <> "groups" _
      And name <> "attachments" And name <> "history" Then
         set entityDefObj = sessionObj.GetEntityDef(name)
```

```
    ' Do something with the EntityDef object
  End If
Next
```

Example (in Perl):

```
$sessionObj = $entity->GetSession();
# Get the list of names for the stateless record types.

$entityDefNames = $sessionObj->GetAuxEntityDefNames();

# Iterate over the non-system stateless record types
$count = $entityDefNames->count();

for ($i = 0; $i < $count; $i++)
  {
  $field = $entityDefNames -> Item($i);
     if ( ($name ne "users") &&
        ($name ne "groups") &&
        ($name ne "attachments") &&
        ($name ne "history") )
     {
        $entitydefobj = $sessionObj->GetEntityDef($name);

        # Do something with the entitydef object
     }
  }
```

See Also:

GetEntityDef Method
GetEntityDefNames Method
GetQueryEntityDefNames Method
GetReqEntityDefNames Method
GetSubmitEntityDefNames Method

GetDefaultEntityDef Method

Description:

Returns the schema’s default EntityDef object.

This method returns the default EntityDef object as defined in the schema. For methods that require a named EntityDef object, ClearQuest uses the default EntityDef object when the name is the empty string ("").

VBScript Syntax:

session.**GetDefaultEntityDef**

Perl Syntax:

$session->GetDefaultEntityDef();

Identifier	Description
session	The Session object that represents the current database-access session.
Return value	The default EntityDef object.

Example (in VBScript):

```
set sessionObj = GetSession

set defEntityDef = sessionObj.GetDefaultEntityDef
```

Example (in Perl):

```
#Create a ClearQuest admin session
$sessionObj = $entity->GetSession();

#Get the default record type of the schema
$defEntityDef = $sessionObj->GetDefaultEntityDef();
```

See Also:

GetEntityDef Method
EntityDef Object

GetEnabledEntityDefs Method

Description:

Returns the EntityDefs collection object enabled in the current schema for a given package revision.

Use with GetEnabledPackageRevs Method to discover which packages and package revisions apply to the current user database.

VBScript Syntax:

```
schemaRev.GetEnabledEntityDefs packName, rev
```

Perl Syntax:

```
$schemaRev->GetEnabledEntityDefs(packName, rev);
```

Identifier	Description
packName	A String that specifies the package name.
rev	A String that specifies the package revision.
Return value	The EntityDefs object for the current package revision.

See Also:

GetEnabledPackageRevs Method
GetEnabledEntityDefs Method of the SchemaRev Object

GetEnabledPackageRevs Method

Description:

Returns a collection object representing the packageRev set that is enabled in the current revision of the schema.

You can call this method from the Session object, in which case the schema revision is already known when you log onto the user database.

See this method, **GetEnabledPackageRevs Method**, in its other object, **SchemaRev Object**, for an alternative use.

VBScript Syntax:

```
session.GetEnabledPackageRevs PackageName, RevString
```

Perl Syntax:

```
$session->GetEnabledPackageRevs(PackageName, RevString);
```

Identifier	Description
PackageName	Name of the package.
RevString	Represents the revision of the package.
Return values	The collection object of the packageRev set.

See Also:

GetEntity Method
GetEnabledPackageRevs Method in **SchemaRev Object**

GetEntity Method

Description:

Returns the specified record.

When requesting a state-based record type, the display_name parameter must contain the visible ID of the record (for example, “DEF00013323”). For stateless **record type**, this parameter must contain the value of the record’s unique key field.

To request a record using its database ID instead of its visible ID, use the **GetEntityByDbId Method**.

VBScript Syntax:

```
session.GetEntity entity def_name, display_name
```

Perl Syntax:

```
$session->GetEntity(entity def_name, display_name);
```

Identifier	Description
session	The Session object that represents the current database-access session.
entity def_name	A String that identifies the name of the record type to which the record belongs.
display_name	A String that identifies the record.
Return value	An **Entity Object** corresponding to the requested record.

Example (in VBScript):

```
set sessionObj = GetSession

set record1 = sessionObj.GetEntity("defect", "DEF00013323")
```

Example (in Perl)

```
#Create a ClearQuest admin session
$sessionObj = $entity->GetSession();

#Get record DEF00013323
$record1 = $sessionObj->GetEntity( "defect", "DEF00013323" );
```

See Also:

BuildEntity Method
EditEntity Method
GetEntityByDbId Method
Managing Records (Entities) that are Stateless and Stateful

GetEntityByDbId Method

Description:

Returns the record with the specified database ID.

Use this method to get a record whose database ID you know. You can get the database ID of a record by calling the **GetDbId Method** of the corresponding Entity object.

To request the record using its visible ID instead of its database ID, use the **GetEntity Method**.

VBScript Syntax:

session.**GetEntityByDbId** *entitydef_name, db_id*

Perl Syntax:

$session->**GetEntityByDbId**(*entitydef_name, db_id);*

Identifier	Description
session	The Session object that represents the current database-access session.
entitydef_name	A String that identifies the name of the record type to which the desired record belongs.
db_id	A Long that is the number used by the database to identify the record.
Return value	An **Entity Object** corresponding to the requested record.

Example (in VBScript):

```
' Save this record's ID for later use.
set sessionObj = GetSession

id = entity.GetDbId

...
' Get the record again
set record = sessionObj.GetEntityByDbId("defect", id)
```

Example (in Perl):

```
#Create a ClearQuest admin session
$sessionObj = $entity->GetSession();

#Save the record's id for later user
```

```
$id = $entity->GetEntityByDbId();
```

See Also:

BuildEntity Method
EditEntity Method
GetEntity Method
GetDbId Method of the Entity object
Managing Records (Entities) that are Stateless and Stateful

GetEntityDef Method

Description:

Returns the specified EntityDef object.

You can use this method to get an EntityDef object for either state-based or stateless record types. To get a list of all EntityDef names in the schema, call the **GetEntityDefNames Method**. You can call other methods of Session to return the names of specific EntityDef subsets. To get an EntityDef that belongs to a family, use the methods specifically for families (given in See Also below).

VBScript Syntax:

session.**GetEntityDef** *entitydef_name*

Perl Syntax:

$session->**GetEntityDef**(*entitydef_name*);

Identifier	Description
session	The Session object that represents the current database-access session.
entitydef_name	A String containing the name of an EntityDef object.
Return value	The requested EntityDef object.

Example (in VBScript):

```
set sessionObj = GetSession

' Get the list of names of the state-based record types.
entityDefNames = sessionObj.GetEntityDefNames

' Iterate over the state-based record types
for each name in entityDefNames
  set entityDefObj = sessionObj.GetEntityDef(name)
  ' Do something with the EntityDef object
  Next
```

Example (in Perl):

```
#Create a ClearQuest admin session
$sessionObj = $entity->GetSession();

# Get an array containing the names of the record types in the
# current database's schema.
$entityDefNames = $sessionObj->GetEntityDefNames();

  #Iterate over the state-based record types
$count = $entityDefNames->count();

for ($i = 0; $i < $count; $i++)
  {
  $field = $entityDefNames -> Item($i);
         $entityDefObj = $sessionObj->GetEntityDef( $name );
         #Do something with the EntityDef object
         }
```

See Also:

GetAuxEntityDefNames Method
GetEntityDefNames Method
GetQueryEntityDefNames Method
GetReqEntityDefNames Method
GetSubmitEntityDefNames Method
EntityDef Object
GetEntityDefFamily Method
GetEntityDefFamilyNames Method

GetEntityDefFamily Method

Description:

Returns the named family EntityDef object.

Returns a valid object if entitydefName corresponds to a family. This method is convenient if you expect the record type to belong to an family. Otherwise, see the **IsFamily Method**.

VBScript Syntax:

session.**GetEntityDefFamily** *entitydefName*

Perl Syntax:

$session->**GetEntityDefFamily**(*entitydefName*);

Identifier	Description
session	The Session object that represents the current database-access session.
entitydefName	A String containing the name of an EntityDef object.
Return value	The requested EntityDef object.

Example (in VBScript):

```
GetEntityDefFamily
```

Returns the name of a family EntityDef.

See Also:

IsFamily Method
GetEntityDefFamilyNames Method
GetIsMaster Method to the **DatabaseDesc Object**

GetEntityDefFamilyNames Method

Description:

Returns an array that contains the names of all family EntityDefs in the schema repository. Provides support for multitype queries

VBScript Syntax:

```
session.GetEntityDefFamilyNames
```

Perl Syntax:

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

Identifier	Description
session	The Session object that represents the current database-access session.
Return value	The requested EntityDef names in a String array.

See Also:

IsFamily Method
GetEntityDefFamily Method
GetEntityDefNames Method

GetEntityDefNames Method

Description:

Returns an array containing the names of the record types in the current database's **schema**.

This method returns the names of all state-based and stateless record types.

After using this method to get the list of names, you can retrieve the **EntityDef Object** for a given record type by calling the **GetEntityDef Method**.

VBScript Syntax:

session.**GetEntityDefNames**

Perl Syntax:

$session->**GetEntityDefNames**();

Identifier	Description
session	The Session object that represents the current database-access session.
Return value	For VB, a Variant containing an array of Strings is returned. Each string in the array contains the name of a single EntityDef in the schema. For Perl, a reference to a String Array is returned.

Example (in VBScript):

```
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
```

Example (in Perl):

```
#Create a ClearQuest admin session
$sessionObj = $entity->GetSession();

#Get an array containing the names of the record types in the
# current database's schema.
$entityDefNames = $sessionObj->GetEntityDefNames();

  #Iterate over the record types
  foreach $name ( @$entityDefNames )
```

```
{
$entityDefObj = $sessionObj->GetEntityDef( $name );
#Do something with the EntityDef object
}
```

See Also:

GetAuxEntityDefNames Method
GetEntityDef Method
GetQueryEntityDefNames Method
GetReqEntityDefNames Method
GetSubmitEntityDefNames Method
EntityDef Object

GetInstalledMasters Method

Description:

Returns the list of registered database sets and master databases.

The returned Variants always contain the same number of strings. The contents of both Variants are ordered so that each schema repository (master database) listed in masterDBs belongs to the database set at the same index in dbSets.

VBScript Syntax:

session.**GetInstalledMasters** *dbSets, masterDBs*

Perl Syntax:

$session->**GetInstalledMasterDbs**();
$session->**GetInstalledDbSets**();

Identifier	Description
session	The Session object that represents the current database-access session.
dbSets	For VB, an empty Variant, which on return contains an array of strings. Each string in the array corresponds to the name of a registered database set.

Identifier	Description
masterDBs	For VB, an empty Variant, which on return contains an array of strings. Each string in the array corresponds to the name of a registered master database.
Return value	For VB, the values are inserted in the empty Variants for the arguments. For Perl, Two separate methods are use to return a reference to a String Array for the database sets and the master databases.

Example (in VBScript):

```
set sessionObj = GetSession

Dim dbSets, masterDBs

sessionObj.GetInstalledMasters dbSets, masterDBs
For Each db in dbSets
  ...
  Next
```

Example (in Perl):

```
#Create a ClearQuest admin session
$sessionObj = $entity->GetSession();

#Get the list of registered database sets and master databases.
#Upon return, the $dbSets variable will contain an array of
#strings, each of which is the name of a registered database set.
#The $masterDBs variable will contain an array of
#strings, each of which is the name of a registered master
#database.
$sessionObj->GetInstalledMasters( @dbSets, @masterDBs );

#Iterate over the database sets
foreach $db ( @$dbSets )
  {
  #Do something with the EntityDef object
  ...
  }
```

See Also:

GetIsMaster Method of the **DatabaseDesc Object**

GetQueryEntityDefNames Method

Description:

Returns an array containing the names of the record types that are suitable for use in queries.

You can use any of the names returned by this method in the entitydef_name parameter for the **BuildQuery Method**. (You can also retrieve an EntityDef object by calling the **GetEntityDef Method**.)

Note: The record types built into ClearQuest can be used in queries, so the returned array is never empty.

VBScript Syntax:

session.**GetQueryEntityDefNames**

Perl Syntax:

$session->**GetQueryEntityDefNames**();

Identifier	Description
session	The Session object that represents the current database-access session.
Return value	For VB, a Variant containing an array of Strings is returned. Each String contains the name of an EntityDef that can be used in a query. For Perl, a reference to a String Array is returned.

Example (in VBScript):

```
set sessionObj = GetSession
' Get the list of names of the record types that support queries.
entityDefNames = sessionObj.GetQueryEntityDefNames

' Iterate over all record types
```

```
for each name in entityDefNames
  set queryDefObj = sessionObj.BuildQuery(name)
  ' Fill in the query parameters and run it
  Next
```

Example (in Perl):

```
$sessionObj = $entity->GetSession();

# Get the list of names of the record types that support queries.
# NOTE: GetQueryEntityDefNames() returns a *REFERENCE* to an
# array.
$entityDefNames = $sessionObj->GetQueryEntityDefNames();
#Iterate over the state-based record types
foreach $name ( @$entityDefNames ){
  $queryDefObj = $sessionObj->BuildQuery( $name );
  #Fill in the query parameters and run it
  ...
  }
```

See Also:

BuildQuery Method
GetAuxEntityDefNames Method
GetEntityDef Method
GetEntityDefNames Method
GetReqEntityDefNames Method
GetSubmitEntityDefNames Method
EntityDef Object

GetReqEntityDefNames Method

Description:

Returns an array containing the names of the state-based record types in the current database's **schema**.

State-based record types are templates for state-based records. Most databases have at least one state-based record type defining the type of data stored by the database. The database may also have several supporting stateless **record type** containing secondary information.

Typically, the return value contains at least one name; however, the return value can be an empty Variant if no state-based record types exist in the schema.

After using this method to get the list of names, you can retrieve the **EntityDef Object** for a given record type by calling the **GetEntityDef Method**.

VBScript Syntax:

session.**GetReqEntityDefNames**

Perl Syntax:

$session->**GetReqEntityDefNames**();

Identifier	Description
session	The Session object that represents the current database-access session.
Return value	For VB, a Variant containing an array of Strings is returned. Each string in the array contains the name of one of the desired record types. For Perl, a reference to a String Array is returned.

Example (in VBScript):

```
set sessionObj = GetSession

' Get the list of names of the state-based record types.
entityDefNames = sessionObj.GetReqEntityDefNames

' Iterate over the state-based record types
for each name in entityDefNames
  set entityDefObj = sessionObj.GetEntityDef(name)
  ' Do something with the EntityDef object
  Next
```

Example (in Perl):

```
#Create a ClearQuest admin session
$sessionObj = $entity->GetSession();

#Get an array containing the names of the state-based record
```

```
#types in the current database's schema.
$entityDefNames = $sessionObj->GetReqEntityDefNames();

#Iterate over the state-based record types
$count = $entityDefNames->count();

for ($i = 0; $i < $count; $i++)
  {
  $field = $entityDefNames -> Item($i);
  $entityDefObj = $sessionObj->GetEntityDef( $name );
  #Do something with the EntityDef object
  ...
  }
```

See Also:

BuildQuery Method
GetAuxEntityDefNames Method
GetEntityDef Method
GetEntityDefNames Method
GetQueryEntityDefNames Method
GetSubmitEntityDefNames Method
EntityDef Object

GetServerInfo Method

Description:

Returns a string identifying the session's OLE server.

Usually, this method returns a string such as "cqole" but the OLE server may choose to return a string that contains other information for identifying the server.

VBScript Syntax:

session.**GetServerInfo**

Perl Syntax:

$session->**GetServerInfo**();

Identifier	Description
session	The Session object that represents the current database-access session.
Return value	A String identifying the OLE server.

Example (in VBScript):

```
set sessionObj = GetSession

serverName = sessionObj.GetServerInfo
```

Example (in Perl):

```
#Create a ClearQuest admin session
$sessionObj = $entity->GetSession();

#Get the session's OLE server
$serverName = $sessionObj->GetServerInfo();
```

See Also:

GetSessionDatabase Method

GetSessionDatabase Method

Description:

Returns information about the database that is being accessed in the current session.

This method differs from the GetAccessibleDatabases method in that it returns the DatabaseDescription object associated with the current session. You can only call this method after the user has logged in to a particular database.

VBScript Syntax:

session.**GetSessionDatabase**

Perl Syntax:

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

Identifier	Description
session	The Session object that represents the current database-access session.
Return value	A **DatabaseDesc Object** that contains information about the current database.

Example (in VBScript):

```
set sessionObj = GetSession

set dbDescObj = sessionObj.GetSessionDatabase
```

Example (in Perl):

```
#Create a ClearQuest admin session
$sessionObj = $entity->GetSession();

#Get information about the database that's being accessed in the
#current session
$dbDescObj = $sessionObj->GetSessionDatabase();
```

See Also:

GetAccessibleDatabases Method
DatabaseDesc Object
Getting Session and Database Information

GetSubmitEntityDefNames Method

Description:

Returns an array containing the names of the record types that are suitable for use in creating a new record.

This method returns the names that are valid to use for the entitydef_name parameter of the **BuildEntity Method**. Not all record types are appropriate for submitting new records. For example, entries for the "users" stateless record type are added using the ClearQuest Designer interface, so "users" is not included in the returned list of names. On the other hand, "projects" would be included because the projects stateless record type has a submit action.

Typically, the return value contains at least one name; however, the return value can be an empty Variant if no state-based record types exist in the schema.

After using this method to get the list of names, you can retrieve the **EntityDef Object** for a given record type by calling the **GetEntityDef Method**.

VBScript Syntax:

```
session.GetSubmitEntityDefNames
```

Perl Syntax:

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

Identifier	Description
session	The Session object that represents the current database-access session.
Return value	For VB, a Variant containing an array of Strings is returned. Each string contains the name of one of the desired record types. For Perl, a reference to a String Array is returned.

Example (in VBScript):

```
set sessionObj = GetSession

' Get the list of names of the appropriate record types.
entityDefNames = sessionObj.GetSubmitEntityDefNames

' Iterate over the appropriate record types
for each name in entityDefNames
  set entityDefObj = sessionObj.GetEntityDef(name)
  ' Do something with the EntityDef object
  Next
```

Example (in Perl):

```
#Create a ClearQuest admin session
$sessionObj = $entity->GetSession();

#Get the list of names of the record types that are suitable for
#use in creating a new record. Returns an array.
$entityDefNames = $sessionObj->GetSubmitEntityDefNames();

#Iterate over the suitable record types
$count = $entityDefNames->count();

for ($i = 0; $i < $count; $i++)
  {
  $field = $entityDefNames -> Item($i);
  $entityDefObj = $sessionObj->GetEntityDef( $name );

  #Do something with the EntityDef object
  ...
  }
```

See Also:

GetAuxEntityDefNames Method
GetEntityDef Method
GetEntityDefNames Method
GetQueryEntityDefNames Method
GetReqEntityDefNames Method
EntityDef Object

GetUserEmail Method

Description:

Returns the electronic mail address of the user who is logged in for this session.

If you have access to the schema repository, you can change the text of the user's email address using the schema repository object User. Simply assign a new value to the Email property of User.

VBScript Syntax:

```
session.GetUserEmail
```

Perl Syntax:

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

Identifier	Description
session	The Session object that represents the current database-access session.
Return value	A String containing the email address of the user who is logged in for this session.

Example (in VBScript):

```
set sessionObj = GetSession

' Get the user's personal information
userName = sessionObj.GetUserFullName
userLogin = sessionObj.GetUserLoginName
userEmail = sessionObj.GetUserEmail
userPhone = sessionObj.GetUserPhone
userMisc = sessionObj.GetUserMiscInfo
```

Example (in Perl):

```
#Create a ClearQuest admin session
$sessionObj = $entity->GetSession();

#Get the user's personal information
$userName = $sessionObj->GetUserFullName();
$userLogin = $sessionObj->GetUserLoginName();
$userEmail = $sessionObj ->GetUserEmail();
$userPhone = $sessionObj->GetUserPhone();
$userMisc = $sessionObj->GetUserMiscInfo();
```

See Also:

GetUserFullName Method
GetUserGroups Method

GetUserFullName Method

Description:

Returns the full name of the user who is logged in for this session.

If you have access to the schema repository, you can change the text for the user's full name using the schema repository object User. Simply assign a new value to the Fullname property of User.

VBScript Syntax:

session.**GetUserFullName**

Perl Syntax:

$session->**GetUserFullName**();

Identifier	Description
session	The Session object that represents the current database-access session.
Return value	A String containing the full name (such as "Jenny Jones") of the user who is logged in for this session.

Example (in VBScript):

```
set sessionObj = GetSession

' Get the user's personal information
userName = sessionObj.GetUserFullName
userLogin = sessionObj.GetUserLoginName
userEmail = sessionObj.GetUserEmail
userPhone = sessionObj.GetUserPhone
userMisc = sessionObj.GetUserMiscInfo
```

Example (in Perl):

```
#Create a ClearQuest admin session
$sessionObj = $entity->GetSession();

#Get the user's personal information
$userName = $sessionObj->GetUserFullName();
$userLogin = $sessionObj->GetUserLoginName();
$userEmail = $sessionObj->GetUserEmail();
$userPhone = $sessionObj->GetUserPhone();
$userMisc = $sessionObj->GetUserMiscInfo();
```

See Also:

GetUserEmail Method
GetUserGroups Method
GetUserLoginName Method
GetUserMiscInfo Method
GetUserPhone Method
Fullname Property of the **User Object**
Getting Session and Database Information

GetUserGroups Method

Description:

Returns a list of the groups to which the current user belongs.

The returned Variant can be empty.

If you have access to the schema repository, you can change the groups to which the user belongs using the Group object. To add a user to a group, call the AddUser method of Group.

VBScript Syntax:

session.**GetUserGroups**

Perl Syntax:

$session->**GetUserGroups**();

Identifier	Description
session	The Session object that represents the current database-access session.
Return value	For VB, a Variant containing an array String of Variants is returned. Each String names a group to which the current user belongs (that is, the user under whose login name the database is currently being accessed). For Perl, a reference to a String Array is returned.

Example (in VBScript):

```
set sessionObj = GetSession

' Iterate over the user's groups
userGroups = sessionObj.GetUserGroups
If IsEmpty(userGroups) = 0 Then
  ‘ Code to handle if no user groups exist
Else
  For Each group in userGroups
     ...
Next
```

Example (in Perl):

```
#Create a ClearQuest admin session
$sessionObj = $entity->GetSession();

#Get the list of groups to which the current user belongs. Returns
#an array of strings.
$userGroups = $sessionObj->GetUserGroups();

if ( IsEmpty( $userGroups ) = 0 )
 {
  #Code to handle if no user groups exist
  ...
  }
else {
$count = $userGroups->count();

for ($i = 0; $i < $count; $i++)
  {
```

```
  $field = $userGroups -> Item($i);
      ...
  }
}
```

See Also:

GetUserEmail Method
GetUserFullName Method
GetUserLoginName Method
GetUserMiscInfo Method
GetUserPhone Method
AddUser Method of the **Group Object**
Getting Session and Database Information

GetUserLoginName Method

Description:

Returns the name that was used to log in for this session.

Once created, you cannot change the login name of a user account. You must instead create a new user with the new account name. You can do this from ClearQuest Designer, or if you have access to the schema repository, you can use the AdminSession object to create a new User object.

VBScript Syntax:

session.**GetUserLoginName**

Perl Syntax:

$session->**GetUserLoginName**();

Identifier	Description
session	The Session object that represents the current database-access session.
Return value	A String containing the login name (such as "jjones") of the user who is logged in for this session.

Example (in VBScript):

```
set sessionObj = GetSession

' Get the user's personal information
userName = sessionObj.GetUserFullName
userLogin = sessionObj.GetUserLoginName
userEmail = sessionObj.GetUserEmail
userPhone = sessionObj.GetUserPhone
userMisc = sessionObj.GetUserMiscInfo
```

Example (in Perl):

```
$sessionObj = $entity->GetSession();

#Get the user's personal information
$userName = $sessionObj->GetUserFullName();
$userLogin = $sessionObj->GetUserLoginName();
$userEmail = $sessionObj ->GetUserEmail();
$userPhone = $sessionObj->GetUserPhone();
$userMisc = $sessionObj->GetUserMiscInfo();
```

See Also:

GetUserEmail Method
GetUserFullName Method
GetUserGroups Method
GetUserMiscInfo Method
GetUserPhone Method
AdminSession Object
User Object
Getting Session and Database Information

GetUserMiscInfo Method

Description:

Returns miscellaneous information about the user who is logged in for this session.

Miscellaneous information is any information that has been entered by the administrator into that user’s profile. Information about the user’s login name, full name, email

address, phone number, and groups is stored separately and can be retrieved by the corresponding Session methods.

If you have access to the schema repository, you can change the text of the miscellaneous information using the schema repository object User. Simply assign a new value to the MiscInfo property of User.

VBScript Syntax:

session.**GetUserMiscInfo**

Perl Syntax:

$session->**GetUserMiscInfo**();

Identifier	Description
session	The Session object that represents the current database-access session.
Return value	A String containing any miscellaneous information about the user.

Example (in VBScript):

```
set sessionObj = GetSession

' Get the user's personal information
userName = sessionObj.GetUserFullName
userLogin = sessionObj.GetUserLoginName
userEmail = sessionObj.GetUserEmail
userPhone = sessionObj.GetUserPhone
userMisc = sessionObj.GetUserMiscInfo
```

Example (in Perl):

```
#Create a ClearQuest admin session
$sessionObj = $entity->GetSession();

#Get the user's personal information
$userName = $sessionObj->GetUserFullName();
$userLogin = $sessionObj->GetUserLoginName();
$userEmail = $sessionObj ->GetUserEmail();
```

```
$userPhone = $sessionObj->GetUserPhone();
$userMisc = $sessionObj->GetUserMiscInfo();
```

See Also:

GetUserEmail Method
GetUserFullName Method
GetUserGroups Method
GetUserLoginName Method
GetUserPhone Method
MiscInfo Property of the **User Object**
Getting Session and Database Information

GetUserPhone Method

Description:

Returns the telephone number of the user who is logged in for this session.

If you have access to the schema repository, you can change the text for the user's phone number using the schema repository object User. Simply assign a new value to the Phone property of User.

VBScript Syntax:

session.**GetUserPhone**

Perl Syntax:

$session->**GetUserPhone**();

Identifier	Description
session	The Session object that represents the current database-access session.
Return value	A String containing the telephone number (if known) of the user who is logged in for this session.

Example (in VBScript):

```
set sessionObj = GetSession

' Get the user's personal information
userName = sessionObj.GetUserFullName
userLogin = sessionObj.GetUserLoginName
userEmail = sessionObj.GetUserEmail
userPhone = sessionObj.GetUserPhone
userMisc = sessionObj.GetUserMiscInfo
```

Example (in Perl):

```
Get a ClearQuest session
$sessionObj = $entity->GetSession();

#Get the user's personal information
$username = $sessionObj->GetUserFullName();
$userLogin = $sessionObj->GetUserLoginName();
$userEmail = $sessionObj->GetUserEmail();
$userPhone = $sessionObj->GetUserPhone();
$userMisc = $sessionObj->GetUserMiscInfo();
```

See Also:

GetUserEmail Method
GetUserFullName Method
GetUserGroups Method
GetUserLoginName Method
GetUserMiscInfo Method
Phone Property of the **User Object**
Getting Session and Database Information

GetWorkSpace Method

Description:

Returns the session's WORKSPACE object.

You can use the WORKSPACE object to manipulate saved queries, charts, and reports in the ClearQuest workspace.

VBScript Syntax:

session.**GetWorkSpace**

Perl Syntax:

$session->**GetWorkSpace**();

Identifier	Description
session	The Session object that represents the current database-access session.
Return value	The WORKSPACE object belonging to the current session.

Example (in VBScript):

```
set sessionObj = GetSession

' Get the workspace for manipulating query, chart, and report
info.
wkSpc = sessionObj.GetWorkspace
```

Example (in Perl):

```
#Get a ClearQuest session
$sessionObj = $entity->GetSession();

#Get the workspace for manipulating query, chart, and report
$wkSpc = $sessionObj->GetWorkspace();
```

See Also:

WORKSPACE Object

HasValue Method

Description:

Returns a Bool indicating whether the specified session variable exists.

Session variables persist until the Session object is deleted. To get or set variables, use the NameValue method.

VBScript Syntax:

session.**HasValue** *name*

Perl Syntax:

$session->**HasValue**(*name*);

Identifier	Description
session	The Session object that represents the current database-access session.
name	A String containing the name of the session variable.
Return value	True if the variable exists in this session, otherwise False.

Example (in VBScript):

```
set sessionObj = GetSession

If HasValue("foo") Then
  fooValue = sessionObj.NameValue("foo")
End If
```

Example (in Perl):

```
#Get a ClearQuest session
$sessionObj = $entity->GetSession();

#If the specified session variable "foo" exists
if ( $sessionObj->HasValue( "foo" ) ) {
  #Get the old value of the session variable "foo"
  $fooValue = $sessionObj->NameValue( "foo" );
}
```

See Also:

NameValue Property

IsMetadataReadonly Method

Description:

Returns a Bool indicating whether the session's metadata is read-only.

VBScript Syntax:

session.**IsMetadataReadonly**

Perl Syntax:

$session->**IsMetadataReadonly**();

Identifier	Description
session	The Session object that represents the current database-access session.
Return value	True if the metadata is read-only, otherwise False.

Example (in VBScript):

```
set sessionObj = GetSession

If sessionObj.IsMetadataReadonly Then
  ...
End If
```

Example (in Perl):

```
#Get a ClearQuest session
$sessionObj = $entity->GetSession();

#If the session's metadata is read-only, perform some action.
if ( $sessionObj->IsMetadataReadonly ) {
  ...
}
```

See Also:

EntityDef Object

MarkEntityAsDuplicate Method

Description:

Modifies the specified record to indicate that it is a **duplicate** of another record.

This method modifies the duplicate record but leaves the original unchanged. The **state** of the duplicate may change, depending on the **schema**. Appropriate links are added to the database. The duplicate is left in the "modify" state, which means that you can subsequently update its fields and that eventually you must eventually validate and commit it.

The administrator can set up different actions of type DUPLICATE. (For example, the actions might have different restrictions on when they are available, or they might have different hooks.) You must specify an action of type DUPLICATE in the duplicate_action_name parameter.

VBScript Syntax:

```
session.MarkEntityAsDuplicate duplicate, original,
     duplicate_action_name
```

Perl Syntax:

```
$session->MarkEntityAsDuplicate(duplicate, original,
   duplicate_action_name);
```

Identifier	Description
session	The Session object that represents the current database-access session.
duplicate	The **Entity Object** that is to be marked as a duplicate of original.
original	The Entity object that is the original data record.
duplicate_action_name	A String that specifies an action whose ActionType is DUPLICATE. This parameter must identify a valid action for the duplicate record.
Return value	None.

Example (in VBScript):

```
set sessionObj = GetSession

' Mark the entity with ID="BUGID00010345" as a duplicate of this
  ' entity.
' Use the action named "duplicate".
set dupEntityObj = sessionObj.GetEntity("defect",
     "BUGID00010345")
sessionObj.MarkEntityAsDuplicate dupEntityObj, entity,
     "duplicate"

' Validate and commit the duplicate entity since it
' is currently modifiable.
error = dupEntityObj.Validate
if error = "" then
  dupEntityObj.Commit
End If
```

Example (in Perl):

```
#Get a ClearQuest session
$sessionObj = $entity->GetSession();

#Mark the entity with ID="BUGID00010345" as a duplicate of this
#entity. Use the action named "duplicate".
$dupEntityObj = $sessionObj->GetEntity("defect",
"BUGID00010345");
$sessionObj->MarkEntityAsDuplicate( $dupEntityObj, $entity,
"duplicate" );

#Validate and commit the duplicate entity since it is currently
modifiable
$error = $dupEntityObj->Validate();
if ( $error eq "" ) {
  $dupEntityObj->Commit();
}
```

See Also:

UnmarkEntityAsDuplicate Method
Notation Conventions for VBScript
Notation Conventions for Perl

OpenQueryDef Method

Description:

Loads a query from a file.

This method loads a previously-defined query from a file. The query can be either a built-in query or one saved by the user from ClearQuest.

VBScript Syntax:

session.**OpenQueryDef** *filename*

Perl Syntax:

$session->**OpenQueryDef**(*filename*);

Identifier	Description
session	The Session object that represents the current database-access session.
filename	The name of the file from which to load the query information.
Return value	A QueryDef object containing the query information.

Example (in VBScript):

```
set sessionobj = GetSession

' Get the query from file "C:\queries\myQuery.txt"
set queryDefObj =
            sessionObj.OpenQueryDef("C:\queries\myQuery.txt")
```

Example (in Perl):

```
sessionObj = $entity->GetSession();

#Get the query from file "C:\queries\myQuery.txt"
$queryDefObj =
$sessionObj->OpenQueryDef("C:\queries\myQuery.txt");
```

See Also:

QueryDef Object

OutputDebugString Method

Description:

Specifies a message that can be displayed by a debugger or a similar tool.

The value of debugString is passed to the Win32 API call OutputDebugMessage. Various tools like debuggers and Purify can detect this call and report the content of the string. Normally, the debug message is invisible to users.

VBScript Syntax:

session.**OutputDebugMessage** *debugString*

Perl Syntax:

$session->**OutputDebugMessage**(*debugString*);

Identifier	Description
session	The Session object that represents the current database-access session.
debugString	A String containing the text to be displayed.
Return value	None.

Example (in VBScript):

```
set sessionObj = GetSession
sessionObj.OutputDebugMessage "This is a test message."
```

Example (in Perl):

```
#Get a ClearQuest session
$sessionObj = $entity->GetSession();

#Display a debug string via a debugger
$sessionObj->OutputDebugString("This is a test message.");
```

See Also:

UnmarkEntityAsDuplicate Method

UnmarkEntityAsDuplicate Method

Description:

Removes the indication that the specified record is a **duplicate** of another record.

This method breaks the linkage between a duplicate and original Entity object. You can call this method to break a link that was established by the user or by calling the **MarkEntityAsDuplicate Method**. If the DUPLICATE action being undone caused a state transition, that transition is undone unless a subsequent state transition occurred after the DUPLICATE action. After this method returns, the record is editable and must be validated and committed using the Entity object's **Validate Method** and **Commit Method**, respectively.

VBScript Syntax:

```
session.UnmarkEntityAsDuplicate duplicate, action_name
```

Perl Syntax:

```
$session->UnmarkEntityAsDuplicate(duplicate, action_name);
```

Identifier	Description
session	The Session object that represents the current database-access session.
duplicate	The **Entity Object** (currently marked as a duplicate) that is to be modified.
action_name	A String that specifies the action to be performed on the duplicate. This parameter must contain the name of a valid action as defined in the schema. Such an action must have the ActionType UNDUPLICATE.
Return value	None.

Example (in VBScript):

```
set sessionObj = GetSession

' Remove the duplicate status of the entity with
ID="BUGID00010345".
' Use the action named "unduplicate".
set oldDupEntityObj = sessionObj.GetEntity("defect",
"BUGID00010345")
sessionObj.UnmarkEntityAsDuplicate oldDupEntityObj, "unduplicate"

' Validate and commit the entity since it is currently modifiable.
error = oldDupEntityObj.Validate
if error = "" then
  oldDupEntityObj.Commit
End If
```

Example (in Perl):

```
#Get a ClearQuest session
$sessionObj = $entity->GetSession();

#Get the entity BUGID00010345
$oldDupEntityObj = $sessionObj->GetEntity( "defect",
"BUGID00010345" );

#Remove the duplicate status of the entity with
#ID="BUGID00010345"
#using the action "unduplicate"
$sessionObj->UnmarkEntityAsDuplicate( $oldDupEntityObj,
  "unduplicate" );

#Validate and commit the entity since it is currently modifiable.
$error = $oldDupEntityObj->Validate();

if ( $error eq "" ) {
  $oldDupEntityObj->Commit();
}
```

See Also:

MarkEntityAsDuplicate Method
Validate Method of the **Entity Object**

Notation Conventions for VBScript
Notation Conventions for Perl

UserLogon Method

Description:

Log in as the specified user for a database session.

Before calling this method, you should have already created and initialized a new Session object. No other Session methods should be invoked before UserLogon, with the exception of the **GetAccessibleDatabases Method**, **OutputDebugString Method**, and **UnmarkEntityAsDuplicate Method**.

If you are writing hook code, you should not need to call this method. ClearQuest creates the Session object for you and logs the user in before calling any hooks.

VBScript Syntax:

```
session.UserLogon login_name, password, database_name,
session_type, database_set
```

Perl Syntax:

```
$session->UserLogon(login_name, password, database_name,
session_type, database_set);
```

Identifier	Description
session	The Session object that represents the current database-access session.
login_name	A String that specifies the login name of the user.
password	A String that specifies the user's password.
database_name	A String that specifies the name of the desired user database. (You must not login to the master database using this method.)

Identifier	Description
session_type	A Long whose value is a **SessionType Constants** specifying whether the session is shared (SHARED_SESSION) or private (PRIVATE_SESSION). Data from a shared session can be accessed by more than one client at a time. (ADMIN_SESSION is not permitted.)
database_set	A String that specifies the name of the master database. You should set this string to the empty string ("").
Return value	None.

Example (in VBScript):

The following example shows you how to log on to the database from a Visual Basic application.

```
set sessionObj = CreateObject("CLEARQUEST.SESSION")

' Login to each database successively.
set databases = sessionObj.GetAccessibleDatabases
For Each db in databases
  dbName = db.GetDatabaseName
  sessionObj.UserLogon "joe", "gh36ak3", dbName,
  AD_PRIVATE_SESSION, ""   ' Access the database

  ...
Next
```

Example (in Perl):

```
#Start a ClearQuest session
$AdminSession= CQPerlExt::CQAdminSession_Build();
$SessionObj = CQPerlExt::CQSession_Build();
#Get a list of accessible databases
@databases = $sessionObj->GetAccessibleDatabases("ClearQuest
     2.0", "", "");

#Foreach accessible database, login as joe with password gh36ak3
Foreach $db ( @databases ) {
  $dbName = $db->GetDatabaseName();
  # Logon to the database
```

```
$sessionObj->UserLogon( "joe", "gh36ak3", $dbName,
AD_PRIVATE_SESSION, "" );
...
}
```

See Also:

GetDatabaseConnectString Method of the DatabaseDesc Object
Getting Session and Database Information
Notation Conventions for VBScript
Notation Conventions for Perl

User Object

User (OAdUser) contains information about a user in the master database. A user's login, access privilege and some other information can be retrieved through this object.

You can get the number of items in the collection by accessing the value in the **Count Property**. Use the **Item Method** to retrieve items from the User Object collection.

See Also:

Users Object

User Object Properties

The following list summarizes the User Object properties:

Property name	Access	Description
Active Property	Read/Write	Indicates whether or not the user's account is active.
AppBuilder Property	Read/Write	Indicates whether or not the user has AppBuilder privileges.
Email Property	Read/Write	Sets or returns the user's email address.
Fullname Property	Read/Write	Sets or returns the user's full name.
Groups Property	Read-only	Returns a collection of groups to which the user belongs.
MiscInfo Property	Read/Write	Sets or returns the user's miscellaneous information.
Name Property	Read-only	Returns the user's login name.
Phone Property	Read/Write	Sets or returns the user's phone number.
SubscribedDatabases Property	Read-only	Returns the collection of databases to which the user is subscribed.
SuperUser Property	Read/Write	Indicates whether or not the user has SuperUser privileges.
UserMaintainer Property	Read/Write	Indicates whether or not the user has user administration privileges.

Active Property

Indicates whether or not the user's account is active.

This property can be returned or set.

Users whose accounts are inactive are not allowed to access any databases associated with this master database. Setting this property to false effectively disables the user's

account. To limit the user's access to a specific set of databases, use the SubscribeDatabase and UnsubscribeDatabase methods instead.

VBScript Syntax:

```
user.Active [= value]
```

Perl Syntax:

```
$user->GetActive();
$user->SetActive(boolean_value);
```

Identifier	Description
user	A User object.
value	A Bool indicating whether the user's account is active.

See Also:

SubscribeDatabase Method
UnsubscribeDatabase Method
SubscribedDatabases Property
AppBuilder Property
SuperUser Property
UserMaintainer Property

AppBuilder Property

Indicates whether or not the user has AppBuilder privileges.

This property can be returned or set.

Users with AppBuilder privileges can create and modify schemas in the master database. (The value in this property corresponds to the Schema Designer checkbox in the User Information dialog.)

VBScript Syntax:

```
user.AppBuilder [= value]
```

Perl Syntax:

```
$user->GetAppBuilder();
$user->SetAppBuilder(boolean_value);
```

Identifier	Description
user	User object.
value	A Bool indicating whether or not the user's account has AppBuilder privileges.

See Also:

Active Property
SuperUser Property
UserMaintainer Property
Schema Object

Email Property

Sets or returns the user's e-mail address.

VBScript Syntax:

```
user.Email [= value]
```

Perl Syntax:

```
$user->GetEmail();
$user->SetEmail(e-mail_address_string);
```

Identifier	Description
user	A User object.
value	A String containing the user's email address.

See Also:

Fullname Property
Name Property

Fullname Property

Sets or returns the user's full name.

VBScript Syntax:

```
user.Fullname [= value]
```

Perl Syntax:

```
$user->GetFullname();
$user->SetFullname(full_name_string);
```

Identifier	Description
user	A User object.
value	A String containing the user's full name, as opposed to the user's login name.

See Also:

Email Property
Name Property

Groups Property

Returns a collection of groups to which the user belongs.

This is a read-only property; it can be viewed but not set.

Each element of the returned collection is a Group object. To add users to a group, use the AddUser method of the Group object.

VBScript Syntax:

```
user.Groups
```

Perl Syntax:

```
$user->GetGroups();
```

Identifier	Description
user	A User object.
Return value	A Groups collection object containing the groups to which this user belongs.

See Also:

AddUser Method of the **Group Object**
Users Object
Groups Object

MiscInfo Property

Sets or returns the user's miscellaneous information.

This property can be returned or set.

You can use the miscellaneous property to store extra information about the user, such as the user's postal address or an alternate phone number.

VBScript Syntax:

```
user.MiscInfo [= value]
```

Perl Syntax:

```
$user->GetMiscInfo();
$user->SetMiscInfo(user_info_string);
```

Identifier	Description
user	A User object.
value	A String containing miscellaneous information about the user.

See Also:

Fullname Property
Name Property

Name Property

Returns the user's login name.

VBScript Syntax:

```
user.Name
```

Perl Syntax:

```
$user->GetName();
```

Identifier	Description
user	A User object.
Return value	A String containing the user's login name.

See Also:

Fullname Property

Phone Property

Sets or returns the user's phone number.

VBScript Syntax:

```
user.Phone [= value]
```

Perl Syntax:

```
$user->GetPhone();
$user->SetPhone(phone_number_string);
```

Identifier	Description
user	A User object.
value	A String containing the user's phone number.

See Also:

Fullname Property
Name Property

SubscribedDatabases Property

Returns the collection of databases to which the user is subscribed.

This is a read-only property; it can be viewed but not set.

Each element in the returned collection is a Database object. If this returns an empty collection or the collection has zero elements, the user is subscribed to all databases.

VBScript Syntax:

```
user.SubscribedDatabases
```

Perl Syntax:

```
$user->GetSubscribedDatabases();
```

Identifier	Description
user	A User object.
Return value	A Databases collection object containing the databases to which the user is subscribed.

See Also:

SubscribeDatabase Method
UnsubscribeAllDatabases Method
UnsubscribeDatabase Method
Database Object
Databases Object

SuperUser Property

The SuperUser Property can be used to retrieve whether user has SuperUser privileges or to set SuperUser privileges for a specified user.

Users with SuperUser privileges have full access to the master database and can perform user administration tasks or create and modify schemas.

VBScript Syntax:

```
user.SuperUser [= value]
```

Perl Syntax:

```
$user->GetSuperUser();
$user->SetSuperUser(boolean_value);
```

Identifier	Description
user	A User object.
value	A Bool indicating whether or not the user's account has SuperUser privileges.

See Also:

Active Property
AppBuilder Property
UserMaintainer Property

UserMaintainer Property

Indicates whether or not the user has user administration privileges.

This property can be returned or set.

Users with UserMaintainer privileges can perform user administration tasks, such as adding new users or modifying the accounts of existing users.

VBScript Syntax:

```
user.UserMaintainer [= value]
```

Perl Syntax:

```
$user->user.GetUserMaintainer();
$user->user.SetUserMaintainer(boolean_value);
```

Identifier	Description
user	A User object.
value	A Bool indicating whether or not the user's account has user administration privileges.

See Also:

Active Property
AppBuilder Property
SuperUser Property

User Object Methods

The following list summarizes the User Object methods:

Method name	Description
SubscribeDatabase Method	Subscribes this user to the specified database.
UnsubscribeAllDatabases Method	Unsubscribes the user from all databases.
UnsubscribeDatabase Method	Unsubscribes the user from the specified database.

SubscribeDatabase Method

Subscribes this user to the specified database.

Use this method to subscribe the user to additional databases. To unsubscribe the user, use the UnsubscribeDatabase method. To get a list of the databases to which the user belongs, get the collection of Database objects in the SubscribedDatabases property.

VBScript Syntax:

```
user.SubscribeDatabase database
```

Perl Syntax:

```
$user->SubscribeDatabase(database);
```

Identifier	Description
user	A User object.
database	The Database object to which the user will be subscribed.
Return value	None.

See Also:

UnsubscribeAllDatabases Method
UnsubscribeDatabase Method
SubscribedDatabases Property

UnsubscribeAllDatabases Method

Unsubscribes the user from all databases.

Calling this method disassociates the user from all user databases in the master database. The user's account is still active.

VBScript Syntax:

```
user.UnsubscribeAllDatabases
```

Perl Syntax:

```
$user->UnsubscribeAllDatabases();
```

Identifier	Description
user	A User object.
Return value	None.

See Also:

SubscribeDatabase Method
UnsubscribeDatabase Method
Active Property
SubscribedDatabases Property

UnsubscribeDatabase Method

Unsubscribes the user from the specified database.

Use this method to unsubscribe the user from a specific database. The user must be subscribed to the specified database before calling this method. To get a list of the

databases to which the user belongs, get the collection of Database objects in the SubscribedDatabases property.

VBScript Syntax:

user.**UnsubscribeDatabase** *database*

Perl Syntax:

$*user*->UnsubscribeDatabase(*database***);**

Identifier	Description
user	A User object.
database	The Database object from which the user will be unsubscribed.
Return value	None.

See Also:

SubscribeDatabase Method
UnsubscribeAllDatabases Method
SubscribedDatabases Property

Users Object

A Users object is a collection object for User objects.

For example a Database object's `SubscirbedUser` property may return an Users object.

This object can only be instantiated when you are in a AdminSession.

See Also:

User Object
AdminSession Object

Users Object Properties

The following list summarizes the Users Object properties:

Property name	Access	Description
Count Property	Read-only	Returns the number of items in the collection.

Count Property

Description:

Returns the number of items in the collection.

This is a read-only property; it can be viewed but not set.

VBScript Syntax:

collection.**Count**

Perl Syntax:

$collection->**Count();**

Identifier	Description
collection	A Users collection object, representing the set of users associated with the current master database.
Return value	A Long indicating the number of items in the collection object. This method returns zero if the collection contains no items.

See Also:

Item Method

Users Object Methods

The following list summarizes the Users Object methods:

Method name	Description
Item Method	Returns the item at the specified index in the collection.

Item Method

Description:

Returns the specified item in the collection.

The argument to this method can be either a numeric index (*itemNum*) or a String (*name*).

VBScript Syntax:

```
collection.Item itemNum
collection.Item name
```

Perl Syntax:

```
$collection->Item(itemNum);
$collection->ItemByName(name);
```

Identifier	Description
collection	A Users collection object, representing the set of users associated with the current master database.
itemNum	A Long that serves as an index into the collection. This index is 0-based so the first item in the collection is numbered 0, not 1.
name	A String that serves as a key into the collection. This string corresponds to the unique key of the desired User object.
Return value	The User object at the specified location in the collection.

See Also:

Count Property

WORKSPACE Object

The WORKSPACE object provides an interface for manipulating saved queries, reports, and charts in the ClearQuest workspace.

You can use this object to

- write external applications to examine the contents of the ClearQuest workspace
- in conjunction with the **QueryDef Object** to execute saved queries, the **CHARTMGR Object** to execute charts, and the **ReportMgr Object** to execute reports.

If you already have a Session object, you can get the WORKSPACE object associated with the current session by calling the Session object's **GetWorkSpace Method**.

If you do not have a Session object, your VB code can create a new WORKSPACE object directly using the CreateObject method as follows:

set wkspcObj = CreateObject("CLEARQUEST.WORKSPACE")

Your Perl code uses this syntax:

```
$wkspcObj = new CQWorkspaceMgr
```

Before you can use a WORKSPACE object created using CreateObject, you must assign a Session object to it. To assign a Session object, you must call the **SetSession Method** of the WORKSPACE object.

You use the methods of the WORKSPACE object to get information about the contents of the ClearQuest workspace. You can get a list of the queries, charts, or reports in the workspace. You can also separate items based on whether they are in the Public Queries folder or in a user's Personal Queries folder. You can also use this object to save queries back to the workspace.

Pathnames in the Workspace

The workspace organizes items into a hierarchical structure that you navigate as a series of nested folders. This hierarchy resembles the Windows Explorer in that you can expand or collapse folders to reveal the layered contents.

You identify individual queries, charts, and reports using the pathname information for that item. The pathname for an item is composed of the folder names enclosing it. Folder names are separated using a forward slash (/) character. For example, the pathname of a query called `All Defects` and located in the `Public Queries` folder would have the pathname `Public Queries/All Defects`.

ClearQuest does not provide an explicit way to create new folders. However, you can create nested folders implicitly when you save a query. The SaveQueryDef method lets you specify pathname information for a query. If the folders in the pathname do not exist, ClearQuest creates them (unless they are top-level folder). ClearQuest does not allow you to create top-level folders; all elements must be nested inside either the `Public Queries` or `Personal Queries` folders.

See Also:

GetWorkSpace Method of the Session Object
CHARTMGR Object
ReportMgr Object

WORKSPACE Object Methods

The following list summarizes the WORKSPACE Object methods:

Method name	Description
GetAllQueriesList Method	Returns the complete list of queries in the workspace.
GetChartDef Method	Returns the QueryDef object associated with the specified chart.
GetChartList Method	Returns the specified list of charts.
GetChartMgr Method	Returns the CHARTMGR object associated with the current session.
GetQueryDef Method	Returns the QueryDef object associated with the specified workspace query.
GetQueryList Method	Returns the specified list of workspace queries.
GetReportList Method	Returns the specified list of reports.
GetReportMgr Method	Returns the ReportMgr object associated with the current session.
SaveQueryDef Method	Saves the query to the specified location in the workspace.
SetSession Method	Associates the specified Session object with this object.
SetUserName Method	Sets the current user name when searching for queries, charts, or reports.
ValidateQueryDefName Method	Verifies that the specified query name and path info are correct.

GetAllQueriesList Method

Description:

Returns the complete list of queries in the workspace.

This method returns both the public queries defined by the ClearQuest administrator and personal queries created by individual users.

VBScript Syntax:

workspace.**GetAllQueriesList**

Perl Syntax:

$workspace->**GetAllQueriesList**();

Identifier	Description
workspace	The Workspace object obtained from the current session.
Return value	An array of strings, each of which contains the pathname of a query.

See Also:

GetQueryDef Method
GetQueryList Method
QueryDef Object

GetChartDef Method

Description:

Returns the QueryDef object associated with the specified chart.

You can use this method to get the query information associated with the specified chart. You can also use the returned QueryDef object to get information about the query, including the name of the query and the SQL string used to execute the query.

VBScript Syntax:

workspace.**GetChartDef** *chartName*

Perl Syntax:

$workspace->**GetChartDef**(*chartName*);

Identifier	Description
workspace	The Workspace object obtained from the current session.
chartName	A String containing the workspace pathname of the chart.
Return value	The QueryDef object associated with the chart.

See Also:

GetChartList Method
GetChartMgr Method
CHARTMGR Object
QueryDef Object

GetChartList Method

Description:

Returns the specified list of charts.

Returns the pathnames of the public or personal charts defined in the ClearQuest workspace. The *typeOfCharts* parameter lets you specify the type of charts to return. Specifying the constant `OLEWKSPCSYSTEMQUERIES` returns only the public charts defined by the ClearQuest administrator. Specifying the constant `OLEWKSPCBOTHQUERIES` returns a list of all of the charts in the workspace (including those of all users).

To return only the charts defined by a particular user, first set the current user name by calling the SetUserName method, then, call this method, specifying the constant `OLEWKSPCUSERQUERIES` for the *typeOfCharts* parameter.

VBScript Syntax:

```
workspace.GetChartList typeOfCharts
```

Perl Syntax:

```
$workspace->GetChartList(typeOfCharts);
```

Identifier	Description
workspace	The Workspace object obtained from the current session.
typeOfCharts	An INT indicating which types of charts should be returned. This value corresponds to one of the **OLEWKSPCQUERYTYPE Constants** enumerated constants.
Return value	An array of Strings, each of which contains the pathname of a single chart.

See Also:

GetChartDef Method
GetChartMgr Method
SetUserName Method
CHARTMGR Object

GetChartMgr Method

Description:

Returns the CHARTMGR object associated with the current session.

You can use the CHARTMGR object to generate charts and control the appearance of the output files.

VBScript Syntax:

```
workspace.GetChartMgr
```

Perl Syntax:

```
$workspace->GetChartMgr();
```

Identifier	Description
workspace	The Workspace object obtained from the current session.
Return value	The CHARTMGR object associated with the current session.

See Also:

GetChartDef Method
GetChartList Method
CHARTMGR Object

GetQueryDef Method

Description:

Returns the QueryDef object associated with the specified workspace query.

You use this method to get the information associated with the specified workspace query. You can use the returned QueryDef object to get information about the query, including the name of the query and the SQL string used to execute the query.

VBScript Syntax:

workspace.**GetQueryDef** *queryName*

Perl Syntax:

$workspace->**GetQueryDef**(*queryName*);

Identifier	Description
workspace	The Workspace object obtained from the current session.
queryName	A String containing the workspace pathname of the query
Return value	The QueryDef object associated with the query.

See Also:

GetQueryList Method
QueryDef Object

GetQueryList Method

Description:

Returns the specified list of workspace queries.

This method returns the pathnames of the public or personal queries defined in the ClearQuest workspace. The *typeOfCharts* parameter lets you specify the type of queries to return. Specifying the constant `OLEWKSPCSYSTEMQUERIES` returns only the public queries defined by the ClearQuest administrator. Specifying the constant `OLEWKSPCBOTHQUERIES` returns a list of all of the queries in the workspace (including those of all users).

To return only the queries defined by a particular user, you must first set the current user name by calling the SetUserName method. You can then call this method, specifying the constant `OLEWKSPCUSERQUERIES` for the *typeOfCharts* parameter.

VBScript Syntax:

```
workspace.GetQueryList typeOfQuery
```

Perl Syntax:

```
$workspace->GetQueryList(typeOfQuery);
```

Identifier	Description
workspace	The Workspace object obtained from the current session.
typeOfQuery	An INT indicating which types of queries should be returned. This value corresponds to one of the **OLEWKSPCQUERYTYPE Constants** enumerated constants.
Return value	An array of Strings, each of which contains the pathname of a single query.

See Also:

GetQueryDef Method
SetUserName Method
QueryDef Object

GetReportList Method

Description:

Returns the specified list of reports.

This method returns the pathnames of the public or personal reports defined in the ClearQuest workspace. The *typeOfCharts* parameter lets you specify the type of reports to return. Specifying the constant `OLEWKSPCSYSTEMREPORTS` returns only the public reports defined by the ClearQuest administrator. Specifying the constant `OLEWKSPCBOTHREPORTS` returns a list of all of the reports in the workspace (including those of all users).

To return only the reports defined by a particular user, you must first set the current user name by calling the SetUserName method. You can then call this method, specifying the constant `OLEWKSPCUSERREPORTS` for the *typeOfCharts* parameter.

VBScript Syntax:

```
workspace.GetReportList typesOfReports
```

Perl Syntax:

```
$workspace->GetReportList(typesOfReports);
```

Identifier	Description
workspace	The Workspace object obtained from the current session.
typesOfReports	An INT indicating which types of reports should be returned. This value corresponds to one of the **OLEWKSPCQUERYTYPE Constants** enumerated constants.
Return value	An array of Strings, each of which contains the pathname of a single report.

See Also:

GetReportMgr Method
ReportMgr Object

GetReportMgr Method

Description:

Returns the ReportMgr object associated with the current session.

You can use the ReportMgr object to execute the specified report, check the status of the report while it is being processed, or check the report parameters.

VBScript Syntax:

```
workspace.GetReportMgr reportName
```

Perl Syntax:

```
$workspace->GetReportMgr(reportName);
```

Identifier	Description
workspace	The Workspace object obtained from the current session.
reportName	A String containing the name of the report to run with the returned ReportMgr object.
Return value	A ReportMgr object you can use to run the specified report.

See Also:

ReportMgr Object

SaveQueryDef Method

Description:

Saves the query to the specified location in the workspace.

The user logged into the current session must have access to the pathname specified in the *qdefPath* parameter. (Thus, only users with administrative privileges can save queries to the Public Queries folder.) If the pathname you specify in the *qdefPath* parameter contains subfolders that do not exist, ClearQuest creates those folders implicitly.

VBScript Syntax:

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

Perl Syntax:

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

Identifier	Description
workspace	The Workspace object obtained from the current session.
qdefName	A String containing the name of the query.
qdefPath	A String containing the pathname of the folder in which you want to save the query.
queryDef	The QueryDef object representing the query you want to save.
overwrite	A Bool indicating whether this query should overwrite a query with the same name and path information.
Return value	None.

See Also:

GetQueryDef Method
GetQueryList Method
QueryDef Object

SetSession Method

Description:

Associates the specified Session object with this object.

If you create a WORKSPACE object without first having a Session object, you must call this method before attempting to access any of the queries, charts, or reports in the workspace.

VBScript Syntax:

```
workspace.SetSession sessionObj
```

Perl Syntax:

```
$workspace->SetSession(sessionObj);
```

Identifier	Description
workspace	The Workspace object obtained from the current session.
sessionObj	The Session object to associate with this object.
Return value	None.

See Also:

Session Object

SetUserName Method

Description:

Sets the current user name when searching for queries, charts, or reports.

You should call this method before attempting to get any information located in a user's Personal Queries folder. You must call this method before requesting user-specific items with the GetChartList, GetQueryList, or GetReportList methods.

VBScript Syntax:

```
workspace.SetUserName userName
```

Perl Syntax:

```
$workspace->SetUserName(userName);
```

Identifier	Description
workspace	The Workspace object obtained from the current session.
userName	A String containing the login ID of the user.
Return value	None.

See Also:

GetChartList Method
GetQueryList Method
GetReportList Method

ValidateQueryDefName Method

Description:

Verifies that the specified query name and path information are correct.

VBScript Syntax:

workspace.**ValidateQueryDefName** *qdefName, qdefPath*

Perl Syntax:

$workspace->**ValidateQueryDefName**(*qdefName, qdefPath*);

Identifier	Description
workspace	The Workspace object obtained from the current session.
qdefName	A String containing the name of the query.
qdefPath	A String containing the pathname of the folder containing the query.
Return value	None.

You can use this method to ensure that the given name and path are valid in the workspace.

See Also:

SaveQueryDef Method

Enumerated Constants

This topic lists all the constants used as arguments or return values by the methods and properties in the ClearQuest API, except as otherwise noted. The constants are grouped into the following categories:

- **ActionType Constants**
- **Behavior Constants**
- **BoolOp Constants**
- **CompOp Constants**
- **DatabaseVendor Constants**
- **EntityType Constants**
- **EventType Constants**
- **FetchStatus Constants**
- **FieldType Constants**
- **FieldValidationStatus Constants**
- **QueryType Constants**
- **SessionType Constants**
- **ValueStatus Constants**
- **OLEWKSPCERROR Constants**
- **OLEWKSPCQUERYTYPE Constants**
- **OLEWKSPCREPORTTYPE Constants**

Note: For the difference between VBScript and Perl constants, see **Notation Conventions for VBScript** and **Notation Conventions for Perl**.

ActionType Constants

The ActionType constants define the legal action types in VBScript.

Constant	Value	Description
_SUBMIT	1	Create a new record.
_MODIFY	2	Change the contents of a record.
_CHANGE_STATE	3	Change the state of a record.
_DUPLICATE	4	Mark the record as a duplicate of another record.
_GETACTIONNAME		Get the name that belongs to the current Entity object action.
_GETACTIONTYPE		Get the action type that belongs to the current Entity object method.
_UNDUPLICATE	5	Undo the DUPLICATE action.
_IMPORT	6	Import a new record.
_DELETE	7	Delete an entity.
_BASE	8	Base actions fire with all other actions. [See the Schemas and Packages appendix of *Administrating ClearQuest*.]
_RECORD_SCRIPT_ALIAS	9	Allows you to call one single method, **GetActionName Method**, instead of having to call three: **EditEntity Method**, **Validate Method**, and **Commit Method**.

Behavior Constants

The Behavior constants identify the behavior of the designated field.

Constant	Value	Description
_MANDATORY	1	A value must be provided. Corresponds to the MANDATORY field behavior in the user interface.
_OPTIONAL	2	A value may be provided but is not required. Corresponds to the OPTIONAL field behavior in the user interface.
_READONLY	3	The designated field cannot be changed. Corresponds to the READONLY field behavior in the user interface.
_USE_HOOK	4	The behavior of the field is determined by calling the associated hook. Corresponds to the USE_HOOK field behavior in the user interface.

BoolOp Constants

The BoolOp constants identify the valid boolean operations.

Constant	Value	Description
_BOOL_OP_AND	1	Boolean AND operator
_BOOL_OP_OR	2	Boolean OR operator

CompOp Constants

The CompOp constants identify the valid comparison operators.

Constant	Value	Description
_COMP_OP_EQ	1	Equality operator (=)
_COMP_OP_NEQ	2	Inequality operator (<>)
_COMP_OP_LT	3	Less-than operator (<)
_COMP_OP_LTE	4	Less-than or Equal operator (<=)
_COMP_OP_GT	5	Greater-than operator (>)
_COMP_OP_GTE	6	Greater-than or Equal operator (>=)
_COMP_OP_LIKE	7	Like operator (value is a substring of the string in the given field)
_COMP_OP_NOT_LIKE	8	Not-like operator (value is not a substring of the string in the given field)
_COMP_OP_BETWEEN	9	Between operator (value is between the specified delimiter values)
_COMP_OP_NOT_BETWEEN	10	Not-between operator (value is not between specified delimiter values)
_COMP_OP_IS_NULL	11	Is-NULL operator (field does not contain a value)
_COMP_OP_IS_NOT_NULL	12	Is-not-NULL operator (field contains a value)
_COMP_OP_IN	13	In operator (value is in the specified set)
_COMP_OP_NOT_IN	14	Not-in operator (value is not in the specified set)

DatabaseVendor Constants

The DatabaseVendor constants identify the supported database types.

Constant	Value	Description
_SQL_SERVER	1	An SQL Server database.
_MS_ACCESS	2	An MS Access database.
_SQL_ANYWHERE	3	An SQL Anywhere database.
_ORACLE7	4	An Oracle database using Oracle7 client networking software

EntityType Constants

The EntityType constants identify state-based or stateless records.

Constant	Value	Description
_REQ_ENTITY	1	State-based records
_AUX_ENTITY	2	Stateless records
_ANY_ENTITY	3	Either state-based or stateless records

EventType Constants

The Type constants identify the cause of hook invocations.

Constant	Value	Description
_BUTTON_CLICK	1	The hook invocation is triggered by a push button click.
_SUBDIALOG_BUTTON_CLICK	2	The hook invocation is triggered by a click on the subdialog button.
_SELECTION	3	The hook invocation is triggered by an item selection.
_DBLCLICK	4	The hook invocation is triggered by a point device double-click.
_CONTEXTMENU_ITEM_SELECTION	5	The hook invocation is triggered by a contextual menu selection.
_CONTEXTMENU_ITEM_CONDITION	6	Indicates whether the hook should enable or disable a contextual menu item. A string value of "1" indicates the item should be enabled. A String value of "0" indicates the item should be disabled.

FetchStatus Constants

The FetchStatus constants identify the status of moving the cursor in a request set.

Constant	Value	Description
_SUCCESS	1	The next record in the request set was successfully obtained.
_NO_DATA_FOUND	2	No more records were found in the request set.
_MAX_ROWS_EXCEEDED	3	Not used.

FieldType Constants

The FieldType constants identify the information contained in a field.

Constant	Value	Description
_SHORT_STRING	1	Simple text field (255 character limit)
_MULTILINE_STRING	2	Arbtitrarily long text
_INT	3	Integer
_DATE_TIME	4	Timestamp information
_REFERENCE	5	A pointer to a stateless record
_REFERENCE_LIST	6	A list of references
_ATTACHMENT_LIST	7	A list of attached files
_ID	8	A special string ID for records
_STATE	9	The current state of a state-based record
_JOURNAL	10	A special list of rows in a subtable that belongs exclusively to this record
_DBID	11	A special internal numeric ID

FieldValidationStatus Constants

The FieldValidationStatus constants identify the status of the designated field.

Constant	Value	Description
_KNOWN_VALID	1	The field's value is known to be valid.
_KNOWN_INVALID	2	The field's value is known to be invalid.
_NEEDS_VALIDATION	3	The field's value may be valid but has not been checked.

QueryType Constants

The QueryType constants identify the type of stored query.

Constant	Value	Description
_LIST_QUERY	1	A list that corresponds to the result set grid in ClearQuest Designer.
_REPORT_QUERY	2	A report that corresponds to a report in the ClearQuest Designer workspace.
_CHART_QUERY	3	A chart that corresponds to a chart in the ClearQuest Designer workspace.

SessionType Constants

The SessionType constants identify the type of session desired.

Constant	Value	Description
_SHARED_SESSION	1	More than one client can access this session's data
_PRIVATE_SESSION	2	Only one client can access this session's data
_ADMIN_SESSION	3	The system administrator is logged into the session.

ValueStatus Constants

The ValueStatus constants identify the status of a field.

Constant	Value	Description
_HAS_NO_VALUE	1	The field has no value set.
_HAS_VALUE	2	The field has a value.
_VALUE_NOT_AVAILABLE	3	The current state of the field prevents it from returning a value.

OLEWKSPCERROR Constants

The OLEWKSPCERROR constants identify errors that can be returned from Workspace-related operations.

Constant	Value
OLEWKSPC_E_NOSESSIONSET	740
OLEWKSPC_E_SESSIONALREADYSET	741
OLEWKSPC_E_CANTCREATESESSION	742
OLEWKSPC_E_CANTCREATEWORKSPACE	743
OLEWKSPC_E_QUERYLISTFAILURE	744
OLEWKSPC_E_QUERYLISTSAFEARRAYFAILURE	745
OLEWKSPC_E_QUERYDEFNOTFOUND	746
OLEWKSPC_E_GETQUERYDEFFAILURE	747
OLEWKSPC_E_QUERYDEFGETBUCKETFAILURE	748
OLEWKSPC_E_QUERYDEFBUCKETGETQUERYDEFFAILURE	749
OLEWKSPC_E_CHARTLISTFAILURE	750
OLEWKSPC_E_CHARTLISTSAFEARRAYFAILURE	751
OLEWKSPC_E_CHARTDEFNOTFOUND	752
OLEWKSPC_E_GETCHARTDEFFAILURE	753
OLEWKSPC_E_CHARTDEFGETBUCKETFAILURE	754
OLEWKSPC_E_CHARTDEFBUCKETGETCHARTDEFFAILURE	755
OLEWKSPC_E_REPORTLISTFAILURE	756
OLEWKSPC_E_REPORTLISTSAFEARRAYFAILURE	757
OLEWKSPC_E_CANTCREATEREPORTMGR	758
OLEWKSPC_E_REPORTMGRNOTFOUND	759
OLEWKSPC_E_GETREPORTMGRFAILURE	760

Constant	Value
OLEWKSPC_E_REPORTMGRGETBUCKETFAILURE	761
OLEWKSPC_E_REPORTMGRBUCKETGETREPORTFAILURE	762
OLEWKSPC_E_REPORTMGR_EXEC_EMPTYHTMLFILENAME	763
OLEWKSPC_E_REPORTMGR_EXEC_RPTENGINEINUSE	764
OLEWKSPC_E_REPORTMGR_EXEC_RPT_EXTRACT_FAILURE	765
OLEWKSPC_E_REPORTMGR_EXEC_RPT_ENGINE_SET_REPORT	766
OLEWKSPC_E_REPORTMGR_EXEC_CADORS_CREATE_FAILURE	767
OLEWKSPC_E_REPORTMGR_EXEC_ATTACH_RS_FAILURE	768
OLEWKSPC_E_REPORTMGR_EXEC_CHECK_FILEPATH_FAILURE	769
OLEWKSPC_E_REPORTMGR_EXEC_ENGINE_FAILURE	770
OLEWKSPC_E_REPORTMGR_EXEC_CAUGHT_EXCEPTION_FAILURE	771
OLEWKSPC_E_NORMALIZEDATETIME_FAIL	772
OLEWKSPC_E_NORMALIZEDATETIME_NULL_INPUT_FAIL	773
OLEWKSPC_E_NORMALIZEDATETIME_PARSE_FAIL	774
OLEWKSPC_E_NORMALIZEDATETIME_FORMAT_FAIL	775
OLEWKSPC_E_NORMALIZEDATETIME_EXCEPTION_FAIL	776
OLEWKSPC_E_QUERYNAMEEXISTS	777
OLEWKSPC_E_INVALIDQUERYNAME	778
OLEWKSPC_E_QUERYDEFSAVEBUCKETFAILURE	779

OLEWKSPCQUERYTYPE Constants

Note: The following constants do not use the notational convention.

The OLEWKSPCQUERYTYPE constants identify the desired source of a query.

Constant	Value	Description
OLEWKSPCQUERIESNONE	0	Do not return queries.
OLEWKSPCSYSTEMQUERIES	1	Return system queries only.
OLEWKSPCUSERQUERIES	2	Return user queries only.
OLEWKSPCBOTHQUERIES	3	Return either system or user queries.

OLEWKSPCREPORTTYPE Constants

Note: The following constants do not use the notational convention.

The OLEWKSPCREPORTTYPE constants identify the desired source of a report.

Constant	Value	Description
OLEWKSPCREPORTSNONE	0	Do not return reports
OLEWKSPCSYSTEMREPORTS	1	Return system reports only.
OLEWKSPCUSERREPORTS	2	Return user reports only.
OLEWKSPCBOTHREPORTS	3	Return either system or user reports.

Examples of Hooks and Scripts

In addition to the examples of hooks and external applications provided in this chapter, see the following resources:

- ClearQuest Designer Help > Working with hooks
- the ClearQuest database that contains ClearQuest hooks, which is at `http://clearquest.rational.com/cqhooks/`

Note: ClearQuest examples do not include error checking and assume that each call is to a valid object.

In addition to the examples of hooks and external applications provided in this chapter, see the following resources:

- ClearQuest Designer Help > Working with hooks
- the ClearQuest database that contains ClearQuest hooks, which is at `http://clearquest.rational.com/cqhooks/`

Note: ClearQuest examples do not include error checking and assume that each call is to a valid object.

Getting and Setting Attachment Information

ClearQuest supports attachments, which enable ClearQuest users to add to a change request record one or more files (text, spreadsheets, screen shots, diagrams, and more). You can both get and set certain kinds of attachment information, such as the attachment description.

The following code fragment iterates over all the attachment fields of a record. For each of the attachment fields, this code

- prints the field names of the attachment_list type, which is a list of attached files (for more information, see GetValueAsList Method).
- iterates over that attachment field's attachments to print the file name, file size, description, and content of each attachment.

To illustrate that the attachment's description is a read/write property, the code also

- alters the description of the attachment
- prints the new description

Note: The following code fragment is a hook (for example, an **action initialization hook**), and therefore "gets" the session object. However, you can also include this code in an **external application** if you manually create the session object and log on to the database.

Example (in VBScript):

```
REM Start of Global Script ShowAttachmentInfo
Sub ShowAttachmentInfo(actionname, hookname)
  DBGOUT "Entering '" & actionname & "' action's " & hookname & "
                       script (VB version)"
  DIM MyAttachmentFields ' The list of attachment fields
  DIM MyAttachmentField  ' An attachment field (contains a list of
                         'attachments)
  DIM MyAttachment       ' An Attachment object

  ' Tell how many attachment fields there are and show their
  ' names...
  M = "This entity contains " & AttachmentFields.Count & "
                                  attachment field(s)" & VBCrLf
  For Each MyAttachmentField in AttachmentFields
     M = M & "    " & MyAttachmentField.Fieldname & VBCrLf
```

```
Next
DBGOUT M

' Iterate over the attachment fields; for each one, list the
' attachments it contains in the current record...
For Each MyAttachmentField in AttachmentFields
   M = "Attachment field '" & MyAttachmentField.Fieldname & "'
                              contains:" & VBCrLf
   ' Iterate over the attachments in this field...
   AtCount = 0
   For Each MyAttachment in MyAttachmentField.Attachments
     AtCount = AtCount + 1
     ' Demonstrate how to set an attachment's description...
     If (Len(MyAttachment.Description) = 0 or
                                MyAttachment.Description = " ")
     Then
       ' DBGOUT "Description before: '" &
          MyAttachment.Description & "'"
          MyAttachment.Description = "Not very descriptive!"
       ' DBGOUT "Description after: '" &
                                MyAttachment.Description & "'"
     End If

     ' Demonstrate how to write out the attachment's contents
     ' to an external file...
     If (MyAttachment.Filename = "foo.doc") Then
        F = "C:\TEMP\" & GetDisplayName() & "_" &
                                            MyAttachment.FileName
         MyAttachment.Load F
        DBGOUT "Attachment " & MyAttachment.FileName & " was
                                                 written to " & F
     End If

     ' Report info about this attachment...
     M = M & "Filename='" & MyAttachment.FileName & "'"  &_
              " FileSize=" & MyAttachment.FileSize &_
              " Description='" & MyAttachment.Description & "'"
               & VBCrLf
   Next
   M = M & "Total attachments: " & AtCount
   DBGOUT M
Next
DBGOUT "Exiting '" & actionname & "' action's " & hookname &
       " script (VB version)"
```

```
End Sub
REM End of Global Script ShowAttachmentInfo
REM Start of Global Script DBGOUT
sub DBGOUT(Message)
    MsgBox Message
end sub
REM End of Global Script DBGOUT
```

Example (in Perl):

```
# Start of Global Script ShowAttachmentInfo
# ShowAttachmentInfo() -- Display information about
# attachments...
sub ShowAttachmentInfo {
  # $actionname as string
  # $hookname as string
  my($actionname, $hookname) = @_;
  my($M) = "Entering '".$actionname."' action's ".$hookname."
    script (Perl version)\n\n";
  # DBGOUT($M); $M="";

  # Get a list of the attachment fields in this record type...
  my($AttachmentFields) = $entity->GetAttachmentFields();

  # Tell how many attachment fields there are and show their
  # names...
  $M = $M . "This entity contains " . $AttachmentFields->Count() .
      " attachment field(s)\n";
  for ($A = 0; $A < $AttachmentFields->Count(); $A++)
  {
    $M = $M . "    " . ($AttachmentFields->Item($A) )
                        ->GetFieldName() . "\n";
  }
  $M .= "\n";

  # Iterate over the attachment fields; for each one, list the
  # attachments it contains in the current record...
  for (my($AF) = 0; $AF < $AttachmentFields->Count(); $AF++) {
    my ($AttachmentField) = $AttachmentFields->Item($AF);
    $M = $M ."Attachment field '"
          . $AttachmentField->GetFieldName().
           "' contains:\n";
    # Iterate over the attachments in this field...
    my($Attachments) = $AttachmentField->GetAttachments();
```

```
    for (my($A) = 0; $A < $Attachments->Count(); $A++) {
      my($Attachment) = $Attachments->Item($A);
      # Demonstrate how to set an attachment's description...
      if ($Attachment->GetDescription() eq " ") {
        # DBGOUT("Description before:
                        '".$Attachment->GetDescription()."'");
         $Attachment->SetDescription("Not too descriptive!");
         # DBGOUT("Description after:
                        '".$Attachment->GetDescription()."'");
      }
      # Demonstrate how to write out the attachment's contents
      # to an external file...
      if ($Attachment->GetFileName() eq "foo.doc") {
        my($F) = "C:\\TEMP\\" . $entity->GetDisplayName()
                              . '_' . $Attachment->GetFileName();
        $Attachment->Load($F);
        DBGOUT("Attachment written to $F
      }
      # Report info about this attachment...
      $M = $M .
      "    Filename='" . $Attachment->GetFileName() . "'" .
      " FileSize=" . $Attachment->GetFileSize() .
      " Description='" . $Attachment->GetDescription() . "'" .
      "\n";
    }
    $M = $M . "Total attachments: " . $Attachments->Count() .
               "\n\n";
  }
  # Display the results...
  DBGOUT($M); $M="";
}
# End of Global Script ShowAttachmentInfo

# Start of Global Script DBGOUT
sub DBGOUT {
  my($Msg) = shift;
  my($FN) = $ENV{'TEMP'}.'\STDOUT.txt';
  open(DBG, ">>$FN") || die "Failed to open $FN";
  print DBG ($Msg);
  close(DBG);
  system("notepad $FN");
  system("del $FN");
}
# End of Global Script DBGOUT
```

Building Queries for Defects and Users

The following code fragments show how to build queries that fetch records from the database by using criteria about defects and users. The samples use the QueryDef and QueryFilterNode objects, as well as a Structured Query Language (SQL) query.

Note: You can use any of the following code fragments in a hook such as a **field choice list hook** or a **field validation hook**. However, you can also include this code in an **external application** if you manually create the session object and log on to the database (instead of getting the session object).

- The following example selects all defects that belong to the “defect” record type.

Example (in VBScript)

```
set session = GetSession
set querydef = session.BuildQuery("defect")
querydef.BuildField("id")
querydef.BuildField("headline")

set resultset = session.BuildResultSet(querydef)
```

Example (in VBScript):

The following example selects defects that match these criteria:

- “beta2” planned release
- “assigned to” user “johndoe”

```
set session = GetSession
set querydef = session.BuildQuery("defect")
querydef.BuildField("id")
querydef.BuildField("headline")

set operator = querydef.BuildFilterOperator(AD_BOOL_OP_AND)
operator.BuildFilter("assigned_to", AD_COMP_OP_EQ, "johndoe")
operator.BuildFilter("planned_release", AD_COMP_OP_EQ, "beta2")

set resultset = session.BuildResultSet(querydef)
```

Example (in VBScript)

The following example select defects that match these criteria:

- assigned to a certain set of users
- planned for resolution during this release
- not yet resolved

```
set session = GetSession
set querydef = session.BuildQuery("defect")
querydef.BuildField("id")
querydef.BuildField("component")
querydef.BuildField("priority")
querydef.BuildField("assigned_to.login_name")
querydef.BuildField("headline")

set operator = querydef.BuildFilterOperator(AD_BOOL_OP_AND)
operator.BuildFilter("planned_release", AD_COMP_OP_EQ, "beta")
operator.BuildFilter("state", AD_COMP_OP_NOT_IN,
"'resolved','verified'")
operator.BuildFilter("priority", AD_COMP_OP_IN, "(1,2)")

set suboperator = operator.BuildFilterOperator(AD_BOOL_OP_OR)
suboperator.BuildFilter("assigned_to",AD_COMP_OP_IN, _
     "'lihong','gonzales','nougareau','makamoto'")

set resultset = session.BuildResultSet(querydef)
```

Example (in VBScript)

The following example finds the users in a certain group (software engineering, sw_eng).

```
set session = GetSession
set querydef = session.BuildQuery("users")
querydef.BuildField("login_name")

set operator = querydef.BuildFilterOperator(AD_BOOL_OP_AND)
operator.BuildFilter("group.name", AD_COMP_OP_EQ, "sw_eng")

set resultset = session.BuildResultSet(querydef)
```

Example (in VBScript)

The following example finds the default settings for when a user, John Doe (johndoe), submits a record. In this example, certain field values are in a database table named “defect” (for the “defect” record type). This code builds a SQL query.

```
set session = GetSession
set resultset = session.BuildSQLQuery("select project, component,
                 severity from defect where user='johndoe'")
```

Updating Duplicate Records to Match the Parent Record

The following code fragment example checks to see whether the record (entity) has any duplicates (children). If so, the hook edits each of the duplicates with the "dupone" action name, and sets the "action_info" field to indicate that the original (parent) record is tested.

Note: We recommend you synchronize duplicates records with the original record by using **an action notification hook**. An action notification hook fires after a record has been successfully committed to the database. You can use an **action commit hook** instead of an action notification hook. However, using an action commit hook creates a risk: if the parent record is *not* committed to the database, but the children records *are* committed to the database, your records will be out of synch.

Example (in VBScript)

```
Dim session ' The current Session object
Dim parent_id ' The current Entity's display name (ID string)
Dim dups  ' Array of all direct duplicates of this Entity
Dim dupvar ' Variant containing a Link to a duplicate
Dim dupobj  ' The same Link, but as an Object rather than
     ' a Variant
Dim entity ' The Entity extracted from the Link

If (HasDuplicates()) Then
  Set session = GetSession
  dups = GetDuplicates
  parent_id = GetDisplayName
  For Each dupvar In dups
      ' You could check these various functions for failures and
      ' then  report any failures to the user (for example, using
      ' MsgBox).
      ' Failures are unlikely, but possible--for example, someone
      ' could concurrently "unmark" an entity as a duplicate.
      Set dupobj = dupvar
      Set entity = dupobj.GetChildEntity
      session.EditEntity entity, "dupdone"
      SetFieldValue "action_info", _
            "Original " & parent_id & " is tested"
      ' commit the record to the database if validation returns no
      ' errors
```

```
    status = Validate
    if status = "" then
     Commit
    else
       Revert
    End If
 Next
    End If
```

Example (in Perl)

```
my($session);    # The current Session object
my($parent_id);  # The current Entity's display name (ID string)
my(@dups );      # Array of all direct duplicates of this Entity
my($dupvar);     # Variant containing a Link to a duplicate
my($dupobj);     # The Entity extracted from the Link
my($status);

if ($entity->HasDuplicates()) {
    $session   = $entity->GetSession();
    @dups      = $entity->GetDuplicates();
    $parent_id = $entity->GetDisplayName();
    foreach $dupvar (@dups) {
        # You could check these various functions for failures
        # and then report any failures to the user (for example,
        # using MsgBox).
        # Failures are unlikely, but possible--for example,
        # someone could concurrently "unmark" an entity as a
        # duplicate.
        $dupobj = $dupvar->GetChildEntity();
        $session->EditEntity($dupobj, "dupdone");
        $dupobj->SetFieldValue("action_info", "Original " .
                               $parent_id . " is tested");
        # commit the record to the database if validation returns
        # no errors
```

```
            $status = $dupobj->Validate();
            if ($status eq "") {
                $dupobj->Commit();
            }
            else {
                $dupobj->Revert();
            }
        }
    }
```

Managing Records (Entities) that are Stateless and Stateful

Your schema has stateless records, such as the Project, and stated records, such as Defect, which move from state to state. The ClearQuest API enables you to get and set field values for both kinds of records.

The example shown in this section is an **external application** example that contains two subroutines: `No_state` for stateless records, and `Has_state` for records that have states. The example does the following:

1. Uses the Session's **BuildEntity Method** to create an **Entity Object**.
2. Set the values in one or more fields.
3. Validates and commits the entity.
4. Retrieves and modifies the entity.
5. Reverts the entity.

The code invokes some external routines that are not shown here:

- StdOut, which prints its arguments to a file
- DumpFields, which prints out an entity's fields to the standard output
- ValidateAndCommit, which calls the Entity object's **Validate Method** and **Commit Method**

Example (in VBScript):

```
' subroutine for stateless records
Sub No_state(session As Object)
  Dim entity As Object
  Dim failure As String

  StdOut "Test for stateless entities is starting"
  StdOut "submit a stateless entity"
  Set entity = session.BuildEntity("project")

  ' ignore failure
  failure = entity.SetFieldValue("name", "initial project name")
```

```
  DumpFields entity
  ValidateAndCommit entity
  Set entity = Nothing

  StdOut "Reload, show values before modification"
  Set entity = session.GetEntity("project", "initial project _
        name")
  DumpFields entity

  StdOut "Modify, then show new values"
  session.EditEntity entity, "modify"

  ' ignore the failure
  failure = entity.SetFieldValue("name", "modified project name")
  DumpFields entity

  StdOut "revert, then show restored values"
  entity.Revert
  DumpFields entity

  StdOut "Modify again, and commit"
  session.EditEntity entity, "modify"

  ' ignore failure
  failure = entity.SetFieldValue("name", "final project name")
  ValidateAndCommit entity
  Set entity = Nothing

  StdOut "Reload, and show final result"
  Set entity = session.GetEntity("project", "final project name")
  DumpFields entity
  Set entity = Nothing

  StdOut "Test for stateless entities is done"
End Sub

' subroutine for stateful records
Sub Has_states(session As Object)
  Dim entity As Object ' the entity that is stateful
   ' failure message from functions that return strings
  Dim failure As String
  Dim failures As Object ' iterator containing list of failure _
        reasons
  Dim id As Long ' ClearQuest defect database ID
```

```
StdOut "Test for stateful entities is starting"
StdOut "submit a stateful entity"
Set entity = session.BuildEntity("defect")

' ignore failures
failure = entity.SetFieldValue("headline", "man bites dog!")
failure = entity.SetFieldValue("project", "final project name")
failure = entity.SetFieldValue("submit_date", "03/18/2000 _
       10:09:08")
id = entity.GetDbId

Open "XXStdout" For Append As #1
Print #1, "Entity id is"; id; Chr(10);
Close #1

DumpFields entity
ValidateAndCommit entity
Set entity = Nothing

StdOut "Reload, show values before modification"
Set entity = session.GetEntityByDbId("defect", id)
DumpFields entity

StdOut "Modify then show new values"
session.EditEntity entity, "modify"

' ignore failure
failure = entity.SetFieldValue("headline", "man bites tree!")
DumpFields entity

StdOut "revert, then show restored values"
entity.Revert
DumpFields entity

StdOut "Modify again and commit"
session.EditEntity entity, "modify"

' ignore failure
failure = entity.SetFieldValue("headline", "tree bites man!")
ValidateAndCommit entity
Set entity = Nothing

StdOut "Reload and show before changing state"
```

```
  Set entity = session.GetEntityByDbId("defect", id)
  DumpFields entity

  StdOut "Change to new state, then show new values"
  session.EditEntity entity, "close"
  failure = entity.SetFieldValue("description", _
        "looked like an oak tree") ' ignore failure
  DumpFields entity

  StdOut "revert then show restored values"
  entity.Revert
  DumpFields entity

  StdOut "Change to new state again then commit"
  session.EditEntity entity, "close"
  failure = entity.SetFieldValue("description", _
     "man of steel, tree of maple") ' ignore failure
  ValidateAndCommit entity
  Set entity = Nothing

  StdOut "Reload, show final values"
  Set entity = session.GetEntityByDbId("defect", id)
  DumpFields entity
  Set entity = Nothing

  tdOut "Test of stateful entities is done"
End Sub
```

Example (in Perl):

```
sub No_state {
    my($session) = @_;
    my($entity);
    my($failure);

    print "Test for stateless entities is starting";
    print "submit a stateless entity";
    $entity = $session->BuildEntity("project");

    # ignore failure
    $failure = $entity->SetFieldValue("name", "initial project
        name");

    DumpFields($entity);
```

```
    ValidateAndCommit($entity);
    $entity = "";

    print "Reload, show values before modification";
    $entity = $session->GetEntity("project", "initial project
        name");
    DumpFields($entity);

    print "Modify, then show new values";
    $session->EditEntity($entity, "modify");

    # ignore the failure
    $failure = $entity->SetFieldValue("name", "modified project
        name");
    DumpFields($entity);

    print "revert, then show restored values";
    $entity->Revert();
    DumpFields($entity);

    print "Modify again, and commit"
    $session->EditEntity($entity, "modify");

    # ignore failure
    $failure = $entity->SetFieldValue("name", "final project
        name");
    ValidateAndCommit($entity);
    $entity = "";

    print "Reload, and show final result";
    $entity = $session->GetEntity("project", "final project
        name");
    DumpFields($entity);
    $entity = "";

    print "Test for stateless entities is done";
}
```

Example (in Perl)

The following is an example of testing for stateful entities:

```
sub Has_states {
    my($session) = @_;
```

```
my($entity); # the entity that is stateful
# failure message from functions that return strings
my($failure);
my($id) # ClearQuest defect database ID

print "Test for stateful entities is starting";
print "submit a stateful entity";
$entity = $session->BuildEntity("defect");

# ignore failures
$failure = $entity->SetFieldValue("headline", "man bites
    dog!");
$failure = $entity->SetFieldValue("project", "final project
    name");
$failure = $entity->SetFieldValue("submit_date", "03/18/2000
    10:09:08");
$id = $entity->GetDbId();

open(FILE, ">>XXStdout");
print FILE, "Entity id is", $id, "\n";
close FILE;

DumpFields($entity);
ValidateAndCommit($entity);
$entity = "";

print "Reload, show values before modification";
$entity = $session->GetEntityByDbId("defect", $id);
DumpFields($entity);

print "Modify then show new values"
$session->EditEntity($entity, "modify");

# ignore failure
$failure = $entity->SetFieldValue("headline", "man bites
    tree!");
DumpFields($entity);

print "revert, then show restored values";
$entity->Revert();
DumpFields($entity);

print "Modify again and commit";
$session->EditEntity($entity, "modify");
```

```
    # ignore failure
    $failure = $entity->SetFieldValue("headline", "tree bites
        man!");
    ValidateAndCommit($entity);
    $entity = "";

    print "Reload and show before changing state";
    $entity = $session->GetEntityByDbId("defect", $id);
    DumpFields($entity);

    print "Change to new state, then show new values";
    $session->EditEntity($entity, "close");
    $failure = $entity->SetFieldValue("description",
                            "looked like an oak tree"); # ignore
        # failure
    DumpFields($entity);

    print "revert then show restored values";
    $entity->Revert();
    DumpFields($entity);

    print "Change to new state again then commit";
    $session->EditEntity($entity, "close");
    $failure = $entity->SetFieldValue("description",
           "man of steel, tree of maple"); # ignore failure
    ValidateAndCommit($entity);
    $entity = "";

    print "Reload, show final values";
    $entity = $session->GetEntityByDbId("defect", $id);
    DumpFields($entity);
    $entity = "";

    print "Test of stateful entities is done";
}
```

Extracting Data About an EntityDef (Record Type)

To illustrate that you can manipulate metadata, the following example of an **external application** prints the following:

- the name of the EntityDef
- the names and types of each field and action it contains
- the names of each state it contains

This subroutine makes use of another routine (not included here) called StdOut, which prints its arguments to the standard output.

Example (in VBScript):

```
Sub DumpOneEntityDef(edef As Object)
  ' The parameter is an EntityDef object.
  Dim names As Variant
  Dim name As String
  Dim limit As Long
  Dim index As Long

  StdOut "Dumping EntityDef " & edef.GetName

  StdOut " FieldDefs:"
  names = edef.GetFieldDefNames
  If IsArray(names) Then
     index = LBound(names)
     limit = UBound(names) + 1
     Do While index < limit
        name = names(index)
        StdOut " " & name & " type=" & edef.GetFieldDefType(name)
        index = index + 1
     Loop
  End If

  names = edef.GetActionDefNames
  If IsArray(names) Then
     index = LBound(names)
     limit = UBound(names) + 1
     Do While index < limit
        name = names(index)
        StdOut " " & name & " type=" &
```

```
                edef.GetActionDefType(name)
            index = index + 1
        Loop
    End If

    If edef.GetType() = AD_REQ_ENTITY Then
        ' stated record type
        StdOut " EntityDef is a REQ entity def"
        StdOut " StateDefs:"
        names = edef.GetStateDefNames
        If IsArray(names) Then
            index = LBound(names)
            limit = UBound(names) + 1
            Do While index < limit
                name = names(index)
                StdOut " " & name
                index = index + 1
            Loop
        End If
        Else
            ' stateless record type
            StdOut " EntityDef is an AUX entity def"
    End If

    StdOut ""
End Sub
```

Example (in Perl)

```
sub DumpOneEntityDef {
my($edef)=@_;
#  The parameter is an EntityDef object.

my(@names);
my($name);
my($limit);
my($index);

   print "Dumping EntityDef ", $edef->GetName;
   print " FieldDefs:";
   @names = $edef->GetFieldDefNames;
   if ( @#names > 0 ) {
      $index = 0;
```

```
      $limit = @#names;
      while $index < $limit {
         $name = @names[$index];
         print " " , $name , " type=" ,
                $edef->GetFieldDefType($name);
         $index = $index + 1;
      }
   }

   print " ActionDefs:"
   @names = $edef->GetActionDefNames;
   if ( @#names > 0 ) {
      $index = 0;
      $limit = @#names;
      while $index < $limit {
        $name = @names[$index];
         print " " , $name , " type=" ,
        $edef->GetFieldDefType($name);
         $index = $index + 1;
      }
   }

   if ($edef->GetType == AD_REQ_ENTITY ) {
      #  stated record type
      print " EntityDef is a REQ entity def"
      print " StateDefs:"
      @names = $edef->GetStateDefNames;
   if ( @#names > 0 ) {
      $index = 0;
      $limit = @#names;
      while $index < $limit {
         $name = @names[$index];
         print " " , $name
         $index = $index + 1;
         }
      }
    }
    else {
         #  stateless record type
         print " EntityDef is an AUX entity def";
     }
print "";
}
```

Extracting Data About a Field in a Record

One of the most common API calls is to the **FieldInfo Object**. For example, the FieldInfo object has the **GetValue Method** that enables you to get the value of a field in a record.

The following Visual Basic **external application** subroutine prints out the information stored in a FieldInfo object. The code invokes an external routine that is not shown here: StdOut, which prints its arguments to a file.

Example (in VBScript):

```
Sub DumpFieldInfo(info As Object) ' The parameter is a FieldInfo
object.
  Dim temp As Long
  Dim status As String
  Dim validity As String
  Dim valuechange As String
  Dim validchange As String
  Dim value As String

  temp = info.GetValueStatus()
  If temp = AD_VALUE_NOT_AVAILABLE Then
     status = "VALUE_NOT_AVAILABLE"
  ElseIf temp = AD_HAS_VALUE Then
     status = "HAS_VALUE" value = "'" & info.GetValue() & "'"
  ElseIf temp = AD_HAS_NO_VALUE Then
     status = "NO_VALUE"
  Else
     status = "<invalid value status: " & temp & ">"
  End If

  temp = info.GetValidationStatus()
  If temp = AD_KNOWN_INVALID Then
     validity = "INVALID"
  ElseIf temp = AD_KNOWN_VALID Then
     validity = "VALID"
  ElseIf temp = AD_NEEDS_VALIDATION Then
     validity = "NEEDS_VALIDATION"
  Else
     validity = "<invalid validation status: " & temp & ">"
  End If
```

```
valuechange = ""
If info.ValueChangedThisSetValue() Then
   valuechange = valuechange & " setval=Y"
Else
   valuechange = valuechange & " setval=N"
End If

If info.ValueChangedThisGroup() Then
   valuechange = valuechange & " group=Y"
Else
   valuechange = valuechange & " group=N"
End If

If info.ValueChangedThisAction() Then
   valuechange = valuechange & " action=Y"
Else
   valuechange = valuechange & " action=N"
End If

validchange = ""
If info.ValidityChangedThisSetValue() Then
   validchange = validchange & " setval=Y"
Else
   validchange = validchange & " setval=N"
End If

If info.ValidityChangedThisGroup() Then
   validchange = validchange & " group=Y"
Else
   validchange = validchange & " group=N"
End If

If info.ValidityChangedThisAction() Then
   validchange = validchange & " action=Y"
Else
   validchange = validchange & " action=N"
End If

StdOut "FieldInfo for field " & info.GetName()
StdOut " field's value = " & value
StdOut " value status = " & status
StdOut " value change =" & valuechange
StdOut " validity = " & validity
StdOut " validity change =" & validchange
```

```
  StdOut " error = '" & info.GetMessageText() & "'"
End Sub
```

Example (in Perl):

```
use CQPerlExt;

$CQsession = CQPerlExt::CQSession_Build();
$CQsession->UserLogon("admin", "", "perl", "");
$record = $CQsession->GetEntity("Defect", "perl00000001");
$fieldInfo = $record->GetFieldValue("id");

$temp = $fieldInfo->GetValueStatus();
if ($temp = AD_VALUE_NOT_AVAILABLE) {
  $status = "VALUE_NOT_AVAILABLE";
} elseif ($temp = AD_HAS_VALUE) {
  $status = "HAS_VALUE";
  $value = "'" & fieldinfo.GetValue() & "'"
} elseif ($temp = AD_HAS_NO_VALUE) {
  $status = "NO_VALUE";
} else {
  $status = "<invalid value status: " & temp & ">";
}

$temp = $fieldInfo->GetValidationStatus();
if ($temp = AD_KNOWN_INVALID) {
  $validity = "INVALID";
} elseif ($temp = AD_KNOWN_VALID) {
  $validity = "VALID";
} elseif ($temp = AD_NEEDS_VALIDATION) {
  $validity = "NEEDS_VALIDATION";
} else {
  $validity = "<invalid validation status: " & temp & ">";
}

$valuechange = "";
if ($fieldInfo->ValueChangedThisSetValue()) {
  $valuechange = $valuechange & " setval=Y";
} else {
  $valuechange = $valuechange & " setval=N";
}

if ($fieldInfo->ValueChangedThisGroup()) {
  $valuechange = $valuechange & " group=Y";
```

```
} else {
  $valuechange = $valuechange & " group=N";
}

if ($fieldInfo->ValueChangedThisAction()) {
  $valuechange = $valuechange & " action=Y";
} else {
  $valuechange = $valuechange & " action=N";
}

$validchange = "";
if ($fieldInfo->ValidityChangedThisSetValue()) {
  $validchange = $validchange & " setval=Y";
} else {
  $validchange = $validchange & " setval=N";
}

if ($fieldInfo->ValidityChangedThisGroup()) {
  $validchange = $validchange & " group=Y";
} else {
  $validchange = $validchange & " group=N";
}

if ($fieldInfo->ValidityChangedThisAction()) {
  $validchange = $validchange & " action=Y";
} else {
  $validchange = $validchange & " action=N";
}

print "FieldInfo for field = ", $fieldInfo->GetName(), "\n";
print "Field's value       = ", $value, "\n";
print "Value status        = ", $status, "\n";
print "Value change        = ", $valuechange, "\n";
print "Validity            = ", $validity, "\n";
print "Validity change     = ", $validchange, "\n";
print "Error = '", $fieldInfo->GetMessageText(), "'";
```

Showing Changes to an Entity (Record)

The following example illustrates how to use the ClearQuest API to get information about field values before and after the user has updated a record. This example is structured as a global hook that can be called from any other hook, such as from the ACTION_COMMIT hook.

Example (in VBScript):

```
REM Start of Global Script ShowOldNewValues
REM TODO -- put your script code here
Sub ShowOldNewValues (actionname, hookname)
  M = "'" & actionname & "' action's " & hookname & " script (VB
          version):" & VBCrLf & VBCrLf
  ' Get a list of the fields in this record type...
  fieldnames = GetFieldNames

  ' Loop through the fields, showing name, type, old/new value...
  if IsArray(fieldnames) Then
     i = LBound(fieldnames)
     Do While i <= UBound(fieldnames)
        FN = fieldnames(i)
        M = M & FN & ":"

        ' Get the field's original value...
        set FieldInfo = GetFieldOriginalValue(FN)
        FieldValueStatus = FieldInfo.GetValueStatus()
        If FieldValueStatus = AD_HAS_NO_VALUE Then
           OldFV = "<no value>"
        ElseIf FieldValueStatus = AD_VALUE_NOT_AVAILABLE Then
           OldFV = "<value not available>"
        ElseIf FieldValueStatus = AD_HAS_VALUE Then
           OldFV = FieldInfo.GetValue()
        Else
           OldFV = "<Invalid value status: " & FieldValueStatus &
                     ">"
        End If

        ' Get the current value (it may have been updated during
        ' this action)...
        set FieldInfo = GetFieldValue(FN)
        FieldValueStatus = FieldInfo.GetValueStatus()
```

```
If FieldValueStatus = AD_HAS_NO_VALUE Then
   NewFV = "<no value>"
ElseIf FieldValueStatus = AD_VALUE_NOT_AVAILABLE Then
   NewFV = "<value not available>"
ElseIf FieldValueStatus = AD_HAS_VALUE Then
   NewFV = FieldInfo.GetValue()
Else
   NewFV = "<Invalid value status: " & FieldValueStatus &
   ">"
End If

' Get and reformat the field's type...
FieldType = FieldInfo.GetType()
is_short = 1
If FieldType = AD_SHORT_STRING Then
   FieldType = "Short String"
ElseIf FieldType = AD_MULTILINE_STRING Then
   FieldType = "Multiline String"
   is_short = 0
ElseIf FieldType = AD_INT Then
   FieldType = "Integer"
ElseIf FieldType = AD_DATE_TIME Then
   FieldType = "Date time"
ElseIf FieldType = AD_REFERENCE Then
   FieldType = "Reference"
ElseIf FieldType = AD_REFERENCE_LIST Then
   FieldType = "Reference List"
   is_short = 0

ElseIf FieldType = AD_ATTACHMENT_LIST Then
   FieldType = "Attachment List"
   is_short = 0
ElseIf FieldType = AD_ID Then
   FieldType = "ID"
ElseIf FieldType = AD_STATE Then
   FieldType = "State"
ElseIf FieldType = AD_JOURNAL Then
   FieldType = "Journal"
   is_short = 0
ElseIf FieldType = AD_DBID Then
   FieldType = "DBID"
Else
   FieldType = "<UNKNOWN TYPE: " & FieldType & ">"
   is_short = 0
```

```
            End IF
            M = M & "  Type=" & FieldType & "."

            ' Display the results. For the purposes of this example,
            ' we show values as follows:
            ' 1. Identify whether the field's value has changed or
            '     not during the current action and indicate that in
            '     the output.
            ' 2. For single-line fields (integer, short_string, etc.)
            '     show the field's value.
            ' 3. For single-line fields whose values have changed
            '    during the current action, show the old and the new
            '     values.
            If OldFV = NewFV Then
               M = M & " Value is unchanged."
               If is_short = 1 Then
                  M = M & " Value='" & OldFV & "'"
               End If
            Else
               M = M & " Value has changed."
               If is_short = 1 Then
                  M = M & " Old value='" & OldFV & "' New value='" &
                          NewFV & "'"
               End If
            End If
            M = M & VBCrLf
            i = i + 1
         Loop
      Else
         M = M & "fieldnames is not an array" & VBCrLf
      End If

      ' At this point we could write this information to a file,
      ' present it in a message box, or write it to the debug window
      ' using the session.OutputDebugString() method. Here, we use
      ' the Windows 'MsgBox' API to present the results to the user
      ' in a message box (note this only works for the first 1024
      ' characters of "M" due to limitations in the Windows MsgBox
      ' API...)
      MsgBox M
   End Sub
   REM End of Global Script ShowOldNewValues
```

Example (in Perl):

```
# Start of Global Script ShowOldNewValues
# ShowOldNewValues: Show field values in the current
# record, drawing attention to fields whose values have changed
# during the current action.
sub ShowOldNewValues {
  # $actionname as string
  # $hookname as string
  my($actionname, $hookname) = @_;
  my($M) = "'".$actionname."' action's ".$hookname." script (Perl
        version):\n\n";
  # Get a list of the fields in this record type
  # (NOTE: GetFieldNames() returns a *REFERENCE* to an array)...
  my($FieldNamesRef) = $entity->GetFieldNames();

  # Loop through the fields, showing name, type, old/new value...
  foreach $FN (@$FieldNamesRef) {
     $M .= $FN . ":"; # Show the field name...

     # Get the field's original value...
     $FieldInfo = $entity->GetFieldOriginalValue($FN);
     $FieldValueStatus = $FieldInfo->GetValueStatus();
     if ($FieldValueStatus == $CQPerlExt::CQ_HAS_NO_VALUE) {
        $OldFV = "<no value>";
     } elseif ($FieldValueStatus ==
                        $CQPerlExt::CQ_VALUE_NOT_AVAILABLE) {
        $OldFV = "<value not available>";
     } elseif ($FieldValueStatus == $CQPerlExt::CQ_HAS_VALUE) {
        $OldFV = $FieldInfo->GetValue();
     } else {
        $OldFV = "<Invalid value status:
                  " . $FieldValueStatus . ">";
     }

     # Get the current value (may have been updated during this
     # action)...
     $FieldInfo = $entity->GetFieldValue($FN);
     $FieldValueStatus = $FieldInfo->GetValueStatus();
     if ($FieldValueStatus == $CQPerlExt::CQ_HAS_NO_VALUE) {
        $NewFV = "<no value>";
     } elseif ($FieldValueStatus ==
                $CQPerlExt::CQ_VALUE_NOT_AVAILABLE) {
        $NewFV = "<value not available>";
```

```
} elseif ($FieldValueStatus == $CQPerlExt::CQ_HAS_VALUE) {
   $NewFV = $FieldInfo->GetValue();
} else {
   $NewFV = "<Invalid value status:
          " . $FieldValueStatus .  ">";
}

# Get and reformat the field's type...
$FieldType = $FieldInfo->GetType();
$is_short = 1;
if ($FieldType == $CQPerlExt::CQ_SHORT_STRING) {
   $FieldType = "Short String";
} elseif ($FieldType == $CQPerlExt::CQ_MULTILINE_STRING) {
   $FieldType = "Multiline String";
   $is_short = 0;
} elseif ($FieldType == $CQPerlExt::CQ_INT) {
   $FieldType = "Integer";
} elseif ($FieldType == $CQPerlExt::CQ_DATE_TIME) {
   $FieldType = "Date Time";
} elseif ($FieldType == $CQPerlExt::CQ_REFERENCE) {
   $FieldType = "Reference";
} elseif ($FieldType == $CQPerlExt::CQ_REFERENCE_LIST) {
   $FieldType = "Reference List";
   $is_short = 0;
} elseif ($FieldType == $CQPerlExt::CQ_ATTACHMENT_LIST) {
   $FieldType = "Attachment List";
   $is_short = 0;
} elseif ($FieldType == $CQPerlExt::CQ_ID) {
   $FieldType = "ID";
} elseif ($FieldType == $CQPerlExt::CQ_STATE) {
   $FieldType = "State";
} elseif ($FieldType == $CQPerlExt::CQ_JOURNAL) {
   $FieldType = "Journal";
   $is_short = 0;
} elseif ($FieldType == $CQPerlExt::CQ_DBID) {
   $FieldType = "DBID";
} else {
   $FieldType = "<UNKNOWN TYPE: " . $FieldType . ">";
   $is_short = 0;
}
$M .= " Type=" . $FieldType . ".";

# Display the results. For the purposes of this example, we
# show values as follows:
```

```
      # 1. Identify whether the field's value has changed or not
      #     during the current action and indicate that in the
      #     output.
      # 2. For single-line fields (integer, short_string, etc.)
      #    show the field's value.
      # 3. For single-line fields whose values have changed during
      #    the current action, show the old and the new values.

      if ($OldFV eq $NewFV) {
         $M .= " Value is unchanged.";
         if ($is_short) {
            $M .= "  Value='".$OldFV."'";
         }
      } else {
         $M .= " Value has changed.";
         if ($is_short) {
            $M .= " Old value='".$OldFV."' New
               value='".$NewFV."'";
         }
      }
      $M .= "\n";
   }
   $M .= "\n'".$actionname."' action's notification script
         (Perl version) exiting.\n";

   # At this point we could write this information to a file,
   # present it in a message box, or write it to the debug window
   # using
$session->OutputDebugString().
   # Here we call a subroutine 'STDOUT' which writes the message
   # out to a file and invokes 'notepad' on it...
   STDOUT($M);
}
# End of Global Script ShowOldNewValues

# Start of Global Script STDOUT
sub STDOUT {
   my($Msg) = shift;
   my($FN) = $ENV{'TEMP'}.'\STDOUT.txt';
   open(DBG, ">>$FN") || die "Failed to open $FN";
   print DBG ($Msg);
   close(DBG);
   system("notepad $FN");
   system("del $FN");
```

```
}
# End of Global Script STDOUT
```

Running a Query and Reporting on its Result Set

ClearQuest client provides powerful reporting capability in a graphical user interface (GUI) environment. The ClearQuest API also supports programmatic reporting.

Sometimes all you need is the raw results rather than a highly formatted report. The following Visual Basic subroutine in an **external application**:

- uses an existing query object to run the query
- prints out the name of the entitydef (record type) that the query runs against
- iterates through all the records in the result set to print the label and value of each field in each record. This subroutines makes use of two other routines, not included here: StdOut, which prints its arguments to a file, and ToStr, which converts its argument to a string.

Example (in VBScript):

```
Sub RunBasicQuery(session As Object, querydef As Object)
' The parameters to this subroutine are a Session object and a
' QueryDef object. It is assumed that the QueryDef is valid (for
' example, BuildField has been used to select one or more fields
' to retrieve).

  Dim rsltset As Object ' This is a ResultSet object
  Dim status As Long
  Dim column As Long
  Dim num_columns As Long
  Dim num_records As Long

  Set rsltset = session.BuildResultSet(querydef)
  rsltset.Execute

  StdOut "primary entity def for query == " & _
            rsltset.LookupPrimaryEntityDefName

  num_columns = rsltset.GetNumberOfColumns
  num_records = 0
  status = rsltset.MoveNext
  Do While status = AD_SUCCESS
     num_records = num_records + 1
     StdOut "Record #" & num_records
```

```
    ' Note: result set indices are based 1..N, not the usual
    '  0..N-1
  column = 1
  Do While column <= num_columns
        StdOut " " & rsltset.GetColumnLabel(column) & "=" & _
           ToStr(rsltset.GetColumnValue(column))
        column = column + 1
    Loop

    StdOut ""
    status = rsltset.MoveNext
  Loop
End Sub
```

Example (in Perl):

```
sub RunBasicQuery {
my($session)=@_[0];
my($querydef)=@_[1];
#  The parameters to this subroutine are a Session object
#  and a QueryDef object. It is assumed that the QueryDef
#  is valid (for example, BuildField has been used to select
#  one or more fields to retrieve).

my ($rsltset);      #  This is a ResultSet object
my ($status);
my ($column);
my ($num_columns);
my ($num_records);

$rsltset = $session->BuildResultSet(querydef);
$rsltset->Execute;

print "primary entity def for query == ",
$rsltset->LookupPrimaryEntityDefName;

   $num_columns = $rsltset->GetNumberOfColumns;
   $num_records = 0;
   $status = $rsltset->MoveNext;
   while ($status == AD_SUCCESS) {
      $num_records = $num_records + 1;
```

```
      print "Record #", $num_records;
     #  Note: result set indices are based 1..N, not the usual
     # 0..N-1
      $column = 1;
      while ($column <= $num_columns) {
         print " ", $rsltset->GetColumnLabel($column), "=",
$rsltset->GetColumnValue($column);
         $column = $column + 1;
      }
      print "";
      $status = $rsltset->MoveNext;
   }
}
```

Getting Session and Database Information

The following Visual Basic **external application** illustrates some of the Session and DatabaseDesc methods. You need a session object to connect to the database. The session object allows you to get information about the database (such as the SQL connect string) and the user that is currently logged on. There are three steps to the process:

1. Create the session object.
2. Log on to the database.
3. Do the tasks you want to do.

For more information, see the **Session Object** and the **DatabaseDesc Object**.

The following code prints out the information stored in the Session's DatabaseDesc object, as well as all the user-related information. This subroutine makes use of another routine (not included here) called StdOut, which prints its arguments to a file.

Example (in VBScript):

```
' Connect via OLE to ClearQuest
Set session = CreateObject("CLEARQUEST.SESSION")

' login_name, password, and dbname are Strings that have
' been set elsewhere
```

```
session.UserLogon login_name, password, dbname,
AD_PRIVATE_SESSION, ""

Set dbDesc = session.GetSessionDatabase
StdOut "DB name = " & dbDesc.GetDatabaseName
StdOut "DB set name = " & dbDesc.GetDatabaseSetName
StdOut "DB connect string = " & dbDesc.GetDatabaseConnectString

StdOut "user login name = " & session.GetUserLoginName
StdOut "user full name = " & session.GetUserFullName
StdOut "user email = " & session.GetUserEmail
StdOut "user phone = " & session.GetUserPhone
StdOut "misc user info = " & session.GetUserMiscInfo

StdOut "user groups:"
Set userGroups = session.GetUserGroups

If IsArray(userGroups) Then
  i = 0
  limit = UBound(userGroups) + 1
  Do While i < limit
     onename = userGroups(i)
     StdOut " group " & onename
     i = i + 1
  Loop
End If
```

Example (in Perl):

```
use lib "E:\\Program Files\\Rational\\common\\lib";

use CQPerlExt;

$CQsession = CQPerlExt::CQSession_Build();

$CQsession->UserLogon("admin", "", "perl2", "");

$dbDesc = $CQsession->GetSessionDatabase();
print "DB name = ", $dbDesc->GetDatabaseName(), "\n";
print "DB set name = ", $dbDesc->GetDatabaseSetName(), "\n";
print "DB connect string = ",
$dbDesc->GetDatabaseConnectString(), "\n";

print "User login name = ", $CQsession->GetUserLoginName(), "\n";
```

```
print "User full name = ", $CQsession->GetUserFullName(), "\n";
print "User email = ", $CQsession->GetUserEmail(), "\n";
print "User phone = ", $CQsession->GetUserPhone(), "\n";
print "Misc user info = ", $CQsession->GetUserMiscInfo(), "\n";

print "User groups: \n";
@userGroups = $CQsession->GetUserGroups();
$size = @userGroups;
if ($size > 0) {
  for ($i=0; $i<=$size; $i++) {
            $onename = $userGroups[$i];
            print "Group ", $i, " ", $onename, "\n";
     }
}
else {
        print "This user does not belong to any group";
}
```

Running a Query Against More than One Record Type

ClearQuest enables you to create a query that retrieves data from more than one record type. A Multitype query fetches data from all the records types that belong to a given "record type family". Here are some possible examples of record type families:

- “change requests” include “defects”, “enhancement requests”, and “documentation requests
- “work orders” include “software fixes” and “hardware fixes”
- “issues” includes “porting,” “features”, and “problem incidents”

To learn about record type families, look up “record type families” in the index of *Administrating ClearQuest*.

This code fragment from an **external application** assumes that

- the schema has one record type family, "TestFamily"
- "TestFamily" contains two record types (for example, "Defect" and "Enhancement Request")

Example (in VBScript):

```
Dim qryDef As OAdQuerydef
Dim resultSet As OAdResultset
Dim familyEntDef As OAdEntityDef

' Insert code here to get the session object and log in to the
' database
families = session.GetEntityDefFamilyNames
If IsArray(families) Then
    Debug.Print UBound(families)
    For i = 0 To UBound(families)
         ' Do something with families(i)
    Next i
    Set qryDef = session.BuildQuery(families(0))
    qryDef.BuildField ("Description")
    Set resultSet = session.BuildResultSet(qryDef)
```

Example (in Perl):

```
# Insert code here to get the session object and log in to the
# database
```

```
$families = $session->GetEntityDefFamilyNames();
foreach $familyName in (@$families)  {
  print ($familyName);
}
if ($qryDef = $session->BuildQuery(@$families[0]))  {
   # do something;
}
$qryDef->BuildField("Description");
$resultSet = $session->BuildResultSet($qryDef);
if ($resultSet->IsMultiType())   {
   # do something;
}
$familyEntDef = $session->GetEntityDefFamily(@$families[0]);
if ($familyEntDef->IsFamily())  {
   # do something;
}
```

Triggering a task with the destination state

To apply some conditional logic, you can determine the destination state of the record currently undergoing an action. Here are some examples:

- Send an email to the Project Manager if a user moves a priority 1 defect into the "postponed" state.
- Allow the user to modify (reapply the "opened" state) to a defect that is currently in the "resolved" state if, and only if, that user belongs to the Manager group.

The following **action notification hook** gets the destination state and sends an email if the current record is being closed.

Note: This action notification hook uses a base action. A base action is an action that occurs with every action. A base action is convenient if you want a hook to fire with more than one action, such as an e-mail notification hook that fires with every action.

Example (in VBScript):

```
Sub Defect_Notification(actionname, actiontype)
  ' actionname As String
  ' actiontype As Long
  ' action = test_base
   set cqSes = GetSession
  ' NOTE: You can also have conditional logic based on the
  ' current action
   set entDef = cqSes.GetEntityDef(GetEntityDefName)
   if entDef.GetActionDestStateName(actionName) = "Closed" then
      ' put send notification message code here
   end if
End Sub
```

Example (in Perl):

```
sub Defect_Notification {
    my($actionName, $actiontype) = @_;
    # $actionName as string scalar
    # $actiontype as long scalar
    # action is test_base
```

```
 $actionName = $entity->GetActionName();
# NOTE: You can also have conditional logic based on the
# current action

 # You can use the $session variable that ClearQuest provides.
  $entDef = session->GetEntityDef($entitiy->GetEntityDefName());
  if ($entDef->GetActionDestStateName($actionName) eq "Closed")
     {# put send notification message code here}
```

Glossary

access control

Access control limits the use or modification of actions to designated users. Access for actions is set through the action's Properties dialog and can be open to all users, limited to a specific group, or controlled by a hook. Access to fields is determined by the Behaviors table.

action

Whenever you modify a record, you invoke an action from the Action menu to register the changes. Actions may result in a state transition or they may simply modify information in the record's fields. In ClearQuest, the Action menu displays only the appropriate legal actions.

ClearQuest allows you to modify a record or to transition a record from one state to another state. For example, you can transition a record from the open state to the closed state.

In ClearQuest Designer: the ClearQuest administrator modifies the Actions table and state transition matrix of each record type to define the legal actions for records of that type. The definition of each action is stored in the ClearQuest schema repository.

administrator

The person responsible for setting up schemas and databases at your company.

The ClearQuest administrator uses ClearQuest Designer to create and modify schemas, databases, and forms. In addition, the administrator performs other tasks, such as creating user groups and establishing permissions, maintaining the database and setting up e-mail notification.

aging chart

Aging charts show how many records have been in the selected states for how long. Use aging charts to answers the questions: "How many defects have been open for one week? For two weeks? For three weeks?"

API

Application Programming Interface.

ClearQuest contains a robust interface that administrators can use to customize the behavior of their databases. The API consists of a set of objects, methods, and functions

that can be called from hook code to perform tasks such as getting or setting the value of a field.

attachment

ClearQuest allows you to associate a file with a particular record. Attachments are stored in the ClearQuest user database, along with other data contained in the record.

attachment field

An attachment field is a field whose type is ATTACHMENT_LIST. An attachment field stores attached files.

Attachment object

In the ClearQuest Designer API: An Attachment object stores information about a single attached file.

Attachments object

In the ClearQuest Designer API: An Attachments object is a collection for Attachment objects. The overall collection contains the attached files for a single field (represented by an AttachmentField object).

AttachmentField object

In the ClearQuest Designer API: An AttachmentField object represents the attached files for a single field. This object stores a reference to an Attachments object.

AttachmentFields object

In the ClearQuest Designer API: An AttachmentFields object is a collection object that represents all of the attached files for a given record. This object stores references to one AttachmentField object for every attachment field in the record.

bar chart

A bar chart illustrates comparisons among individual items. Categories are organized horizontally, values vertically, to focus on comparing values. For example, the bar chart “Defects by Engineer” displays the names of the engineers along the horizontal or x axis and the type of defect along the vertical or y axis.

behavior

The behavior of a field defines the access restrictions for the field. The behavior for a given field can be READONLY, OPTIONAL, MANDATORY, or USE_HOOK. To set the behavior for a field, modify the field's entry in the Behaviors table.

change-state action

A change-state action moves a record from one state to another state.

chart

A chart is a graphical representation of a selected set of records, created usually for the purpose of comparing attributes of those records. There are several different kinds of charts including distribution charts, trend charts, and aging charts. Results can be displayed using several different kinds of graphics, including bar chart and pie chart graphics.

checkout/checkin

The two-part process that allows you to edit a schema and add a new version of the schema to the ClearQuest schema repository. You can modify a schema only after you check it out of the schema repository. A version of the schema can be associated with a database only after you check it in to the schema repository.

A checkout allows you to add fields, record types, forms, states and actions to a schema.

A checkin adds a new version of the schema to the ClearQuest schema repository. Once you check a schema into the schema repository, you can make changes to it only by creating a new version of the schema. You can save intermediate changes to a schema, without checking it into the schema repository and without updating the version of the schema.

control

A graphic element such as a text box, list box, button or picture that you place on a form to display data, enter data, perform an action, or make the form easier to read.

cursor

The cursor is a placeholder used while navigating through a result set. The cursor indicates which row is currently being reviewed.

database

In ClearQuest, the term database refers to the client database that contains all user data and a copy of the schema associated with the database. The ClearQuest database contains all forms, fields, the state transition matrix, and all data entered by users. Compare with production database and test database.

DatabaseDescription object

In the ClearQuest Designer API: A DatabaseDescription object contains information about a particular database. You can use this object to get information about the database.

database set

A database set consists of a schema repository and all of the databases associated with that repository. The databases in the set can be either production databases or test databases.

dependent field

A dependent field is one whose value is affected by the values in other fields. To set up a dependent field, you must create hooks that set the value of the field when the original field (or fields) changes. Typically, you would modify one of the following hooks: the field default value hook, the field value-changed hook, or the field choice-list hook.

Designer toolbar

In ClearQuest Designer: The Designer toolbar provides easy access to some of the more commonly-used menu items.

destination state

When you perform an action that causes a state transition, the destination state is the state to which the record is sent. The record originates from the source state.

distribution chart

Distribution charts are used to measure how many records fall into defined categories or match the values you indicate.

For example, use a distribution chart to see the current status of a group of records, or see who has been assigned the most/least change requests. Another example is a chart that details which records have the highest priority.

duplicate

In ClearQuest, the term duplicate refers to the nature of a record, action or field in the state transition matrix.

A record that contains data already recorded in a previous entry is a duplicate. In the case of defect tracking, a duplicate identifies a defect that has previously been reported.

duplicate action

A duplicate action marks a given record as a duplicate of another record.

entity object

In ClearQuest Designer: An entity object is a runtime object that represents a record in the database.

entitydef object

In ClearQuest Designer: An entitydef object is a runtime object that represents the metadata for a record. This metadata describes the structure of the record, including the number of fields, their names, what data types they must contain, the names of the permitted actions and states for this record type, and so on.

external application

You can write an external application in VBScript or Perl to perform tasks against a ClearQuest database. An external application must begin by creating a session object and logging in to a ClearQuest database. Among the tasks you can then perform are: create a query, execute a saved query, create new records, perform actions on existing records.

expression

Any combination of an operator, value, and field name that evaluate a single value. Filters use expressions to define query criteria.

field

A field represents a singular piece of data in a record. Fields can contain simple data types such as numbers and strings or they can contain more complex information such as references to other fields, dates, or the current state of a record.

FieldInfo object

In the ClearQuest Designer API: A FieldInfo object contains information about a particular field. You can use this object to obtain the field's value and other attributes.

filter

In ClearQuest, filters are restrictions you place on a query to limit the number of records returned. Typically, a filter specifies which field values are needed to identify the specific records you want to work with. For example, you can set up a filter that limits the query to records submitted after a certain date. Records that were submitted before the given date are not returned in the query results.

Filter dialog

In ClearQuest, use the Filter dialog to edit the criteria of a given filter.

form

A form provides a visual interface for entering data into a new record, for modifying the data in an existing record, or for specifying query information. You can create record forms or submit forms for your schema.

Form Layout toolbar

In ClearQuest Designer: The Form Layout toolbar allows you to adjust the alignment and size of controls in a form.

hook

Hooks are entry points, like triggers, for pieces of code that ClearQuest executes at specified times to more fully customize the product. Actions can have hooks for access control, initialization, notification, committal, and validation. Fields can have hooks for specifying default values, choice lists, and permissions and for handling tasks associated with the field when it is validated or its value changes. Records can use record scripts that allow you to trigger actions that are specific to a record type. Global scripts allow

you write a subroutine, such as an e-mail notification, that can be called from any hook in any record type.

history

ClearQuest allows you to track all modifications of each record. The history of a record includes the creation date and each modification made to the record, such as assigning a defect to an engineer, adding details to the description field and resolving a defect.

history field

In ClearQuest Designer: A History field stores information about the actions that have taken place on a record. Every record has an implicit history field associated with it. You cannot create new history fields. You can place a history control on a form to allow the user to view the history of a record.

History object

In the ClearQuest Designer API: A History object stores a text string describing an action that was initiated on a record.

Histories object

In the ClearQuest Designer API: A Histories object is a collection that stores the History objects associated with a single history field.

HistoryField object

In the ClearQuest Designer API: A HistoryField object represents the history entries displayed in a single history field.

HistoryFields object

In the ClearQuest Designer API: A HistoryFields object is a collection that stores all of the history information for a given record. This object stores references to one HistoryField object for every history field used on the record type's form.

HookChoices object

In the ClearQuest Designer API: A HookChoices object represents the choices stored for a given field.

import action

An import action is used when records are imported from another database. During an import action, the new records are added to the database with only a limited amount of validation. In particular, records are not validated to determine whether or not they could have legally reached their current state.

initialization hook

In ClearQuest Designer: An initialization hook can be associated with an action to initialize the fields of a record to some default values. Because this hook provides access to all the fields of the record, you should use it primarily for complex initialization. Compare with the default value hook for fields.

line chart

A line chart shows trends in data at equal intervals. Generally time elements are displayed along one axis and values displayed along the alternate axis.

Link object

In the ClearQuest Designer API: A Link object represents a link between a duplicate and its original record. You cannot create Link objects directly.

metadata

Metadata is information that describes other information. In ClearQuest, metadata is used to specify the structure of records. Databases use metadata to perform searches.

modify action

A modify action allows users to modify the fields of a record without changing the record's state.

notification hook

In ClearQuest Designer: A notification hook can be associated with an action to send notifications or to trigger other actions. For example, a notification hook can send an e-mail message to a group of people to alert them to changes in a particular record.

Operator

Operators act on field values to create a filter expression. Valid operators are:

IN	Looks for single or multiple values (that is, several different states).
EQUAL	Looks for one value (that is, a specific record description or date.)
CONTAINS	Lets you look for text within a value (that is, words or words that might exist in the records you're looking for).
IS NULL	Looks for fields that have no value entered. Tip: To look for fields that have any value, select the Not check box with the IS NULL operator.
BETWEEN	Lets you look for a range of numeric values such as dates.
GREATER THAN	Lets you look for values greater than the value specified (that is, records entered after a certain date).
GREATER THAN	Lets you look for values less than the value specified (that is, records entered before a certain date).

original

An original record is a record that has one or more duplicate records associated with it. ClearQuest updates duplicate records using information in the original record.

Note: An original object can itself be a duplicate of another object.

parent

A parent record is the original record among two or more duplicate records. All other records are children (or duplicates) of the parent and should draw their state information from the parent.

permission

Users must be granted permission to access a database or to access the fields of a record. The ClearQuest administrator defines the permissions for each user using ClearQuest Designer.

pie chart

A pie chart shows the relationship of items to the sum of the items, or as a percentage of the whole. It always displays only one data series and is useful when you want to emphasize a significant element.

poll interval

The poll interval for a database is the amount of time a database waits before checking to see if a user’s connection is still valid.

production database

The production database is the database used by users to submit defects, run queries, and modify records. The data in this database is used by the company to track defects from the time they are found to the time they are fixed.

property

In VBScript, a property is a data member of an object. Properties contain readable (and occasionally writable) values associated with the object.

query

A query is a request to the system to return a set of records that match the specified search criteria. Queries use filters to set up the search criteria.

QueryDef object

In the ClearQuest Designer API: A QueryDef object contains the information for a query, including the fields to display and the search criteria. You must use this object in conjunction with a ResultSet object to initiate a query.

QueryFilterNode object

In the ClearQuest Designer API: A QueryFilterNode object contains information about the search criteria in a query. This object represents a single condition in the search

criteria. Multiple QueryFilterNode objects can be created and grouped to perform complex searches.

record form

A record form is a form that can be used to display the contents of a record or to submit new records. Every record type must have at least one record form, which ClearQuest displays by default when you query the ClearQuest database. If a record type also has a submit form, ClearQuest uses that form when submitting new records instead of the record form.

record type

A record type is a template that defines the actions, fields, forms, behaviors, and state information associated with a record. The state information associated with record types defines the rules for how a record moves from state to state in the database. Schemas can also contain stateless record types which do not move from state to state.

record type family

ClearQuest enables you to define a "family" of record types that have related characteristics so that one query can be defined which will return the results from one of these "families" of types. This will, among other things, enable you to run "to do" lists across multiple record types, such as defects, enhancement requests, and tasks with a single query.

result set

A result set contains the data returned from a database search (query). This data is organized into rows and columns where each row represents a single record and each column represents a designated field of the record.

ResultSet object

In the ClearQuest Designer API: A ResultSet object initiates a query and provides methods to allow you to navigate through the search results.

schema

In ClearQuest, the term schema refers to all the attributes associated with a database. This includes field definitions, field behaviors, the state transition table, actions, report formats, and forms.

The ClearQuest administrator creates and modifies schemas in ClearQuest Designer. ClearQuest supports multiple schemas and multiple versions of each schema. Each version of a schema can be associated with multiple databases.

ClearQuest allows you to update and delete a schema. There must always be at least one schema in the ClearQuest schema repository .

schema repository

The schema repository is a master database that contains all the data associated with existing schemas. No user data is stored in the schema repository.

Session object

In the ClearQuest Designer API: A Session object represents the context in which users access a database. The Session object provides methods to allow the creation and modification of records and queries.

source state

When you perform an action that causes a state transition, the source state is the state from which the record originated. The record is sent to the destination state .

SQL

SQL stands for Standard Query Language and is a language supported by most databases for specifying queries.

SQL editor

In ClearQuest, use the SQL editor to edit SQL expressions.

state

The state of a record refers to the record's location in the record lifecycle. The ClearQuest administrator defines the possible states in which a record can exist. For example, a record is usually given the Submit state when it is first entered into the system. From there, it might proceed to the Open state while the defect is being examined, and then to the Fixed state when the defect has been corrected.

state transition

A state transition occurs when a record moves from one state to a different state. Actions trigger state transitions based on the rules set up in your system's state transition matrix.

state transition matrix

The state transition matrix defines the rules for moving from one state to another state. For each state, the administrator decides the appropriate set of state transitions for that state and enters them into the matrix.

submit action

A submit action allows users to create new records in the database.

submit form

A submit form is a specialized type of form that is used only for adding new records to the ClearQuest database. If a submit form is available, it is used instead of the default record form when adding new records. ClearQuest still uses the record form to display existing records in the database.

test database

A test database is a database used by the administrator to verify the correctness of the schema associated with the database. Typically, a test database contains a set of artificial records whose contents are created solely for the purpose of testing.

trend chart

Trend charts show how many records were transitioned into the selected states by day, week or month. In other words, they show you the rate at which new records are being submitted, resolved or moved into other states.

UNC Pathname

A Uniform Naming Convention pathname allows you to fully specify the location of a file. A UNC Pathname includes the host machine and directory information and is of the format:

```
\\machine_name\directory\file.ext
```

undo checkout

Cancels a schema checkout. ClearQuest cancels all edits to a schema and reverts to the previously saved version of the schema when you undo a checkout. You can save intermediate changes to a schema, without checking it into the schema repository or updating the version of the schema. Once you check a schema into the schema repository, you can only make changes to it by creating a new version of the schema.

unduplicate action

An unduplicate action removes the mark from a record that identifies it as a duplicate of another record.

unique key

The database needs to know which column or combination of columns always have a unique value. For record types that contain states, the unique key is the ID. For stateless record types, the administrator must assign a unique key. For example, in a project table, the Project Name could be the unique key. In the case that there are multiple versions of the project, the Project Name and the Version can be the unique key.

upgrade database

The process of applying recent changes to a user database. The changes are created using ClearQuest Designer and stored in the master database.

user

A ClearQuest user is someone who submits records to a database using ClearQuest or who modifies existing records in a database. Users can also create their own custom forms to use when creating queries but cannot modify the public forms provided with the database. Compare with administrator.

user group

A user group is a list of users with similar privileges and access permissions. ClearQuest uses user groups to limit access to certain actions. When access to an action is limited to a user group, only members of that group may perform the action.

validate

ClearQuest stores all schemas in the schema repository. Before checking in changes to a schema ClearQuest validates all changes, verifying that field types and behavior are valid. Some of the tests ClearQuest performs during the validation process are:

- Validates that you have not used SQL reserved words incorrectly.
- Validates that you have entered unique labels and names for fields and actions.
- Validates that you have assigned a type to each field and a behavior for each state of each field.
- Validates that you have supplied a reference_to record type for each reference field.
- Validates that you have defined a source state and a destination state for all state transitions.
- Validates that you have defined a unique key for all stateless record types.

validation hook

A validation hook verifies that the fields in a record do not contain illegal values. Validation hooks can be associated with fields to verify the contents of the field immediately or with actions to verify the fields in an entire record.

version

ClearQuest allows you to modify or update a schema. Each time that you checkout a schema, ClearQuest creates a new version, or revision, of the schema. ClearQuest stores each version of the schema in the schema repository. You can associate any version of a schema to a database.

ClearQuest allows you to delete either the last version of the schema or the entire schema.

Workspace

The Workspace displays the currently available elements in the left pane of the ClearQuest component. Elements in the Workspace are displayed as a series of navigable folders that can be expanded and collapsed as needed.

In ClearQuest, the Workspace displays your personal and system queries, charts, and reports.

In ClearQuest Designer, the Workspace displays the elements of the currently selected schema. Schema elements include field and behavior tables, states and the state transition matrix, forms and stateless record types, such as user and project tables.