Rational Software Corporation®

Rational® ClearQuest®

API Reference

VERSION: 2003.06.00

UNIX/WINDOWS EDITION

Rational®
the software development company

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

Legal Notices

Version Number: 2003.06.00

Third Party Notices, Code, Licenses, and Acknowledgements

Contents

Using the ClearQuest API 1

The ClearQuest API is implemented as a COM library (cqole.dll) for VBScript/Visual Basic, and as a Perl package (CQPerlExt). You can write hook scripts in VBScript or Perl. The API documentation presents both of these collections of interfaces.

- The COM library is presented as properties and methods
- The Perl package is presented as methods. Get and Set methods that perform the same functions as a VB property, are listed with that property. Tables are provided in the methods section to help identify these Perl methods.

Understanding the ClearQuest API

You can use this API to write code that runs within Rational 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.

ClearQuest runs your hooks in VBScript or Perl, but not both at the same time. ClearQuest Designer allows you to switch between scripting languages. For more information, see *Rational ClearQuest Administrator's Guide*.

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

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

- The prefix and syntax are different. For more information, 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. For more information, see 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 Corporation 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 following table outlines the Perl notational conventions of this manual.

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.

Perl Error Handling

When routines in the ClearQuest API encounter unexpected conditions, they throw an exception. If the exception is not caught by the calling program, the language interpreter terminates your program. If there is any possibility that the ClearQuest API call will fail, you should catch and handle exceptions.

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.

Several functions which are expected to commonly fail are exceptions to this. In particular, validate and set field functions return error indications instead of throwing exceptions.

Using VBScript

Notation Conventions for VBScript

The following table outlines VBScript notational conventions used in this manual.

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. (For more information, see "Getting a Session Object" on page 12.
AD	Prefix for VBScript **Enumerated Constants**. For example: AD_ORACLE

VBScript Error Handling

When routines in the ClearQuest API encounter unexpected conditions, they throw an exception. If the exception is not caught by the calling program, the language interpreter terminates your program. If there is any possibility that the ClearQuest API call will fail, you should catch and handle exceptions.

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.

Several functions which are expected to commonly fail are exceptions to this. In particular, validate and set field functions return error indications instead of throwing exceptions.

Handling VARIANT Return Values

For VBScript, some of the properties and methods return a VARIANT value which is supposed to contain an array of objects or Strings. If the array contains zero elements, then the VARIANT value would simply have a value of EMPTY. An empty value is not considered to be an array, and if you try to iterate over something that's not an array, it is considered a type-mismatch. You should check such a return value with the IsEmpty or IsArray functions before applying any array-related function on it. For example:

```
fieldObjs = GetInvalidFieldValues
' Check the return value
If (IsArray(fieldObjs)) Then
   For Each fieldInfo In fieldObjs
      fieldValue = field.GetValue
      fieldName = field.GetName
      currentsession.outputdebugstring "This is the fieldvalue " & fieldvalue
   Next
Else
      currentsession.outputdebugstring "This is not an array or it is empty"
End If
```

Debugging Your Code

You can debug your schema customization effort from within ClearQuest using a number of different utilities. The support is as follows:

- ClearQuest Designer hook compiler

  This utility catches some syntax errors.

- DBWIN32

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

  This utility is available from Visual Basic and Perl hooks and external scripts, on Windows only.

- MsgBox

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

  This function is available on Windows only.

- Internet Explorer 4.0 debugger

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

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

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

- Microsoft Development Studio VBScript debugger

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

Overview of the API Objects

The ClearQuest API includes user database objects, schema repository objects, schema repository collection objects, and additional objects. The primary point of entry into the API is through one of the following objects:

- Session

  Provides access to user database objects

- AdminSession

  Provides access to schema repository objects

Understanding Session Objects

Session and its sub-objects provide services to user database 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. 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 starting a session, the object you will work with most often is the Entity object. The Entity object represents a single user data record in the database 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).

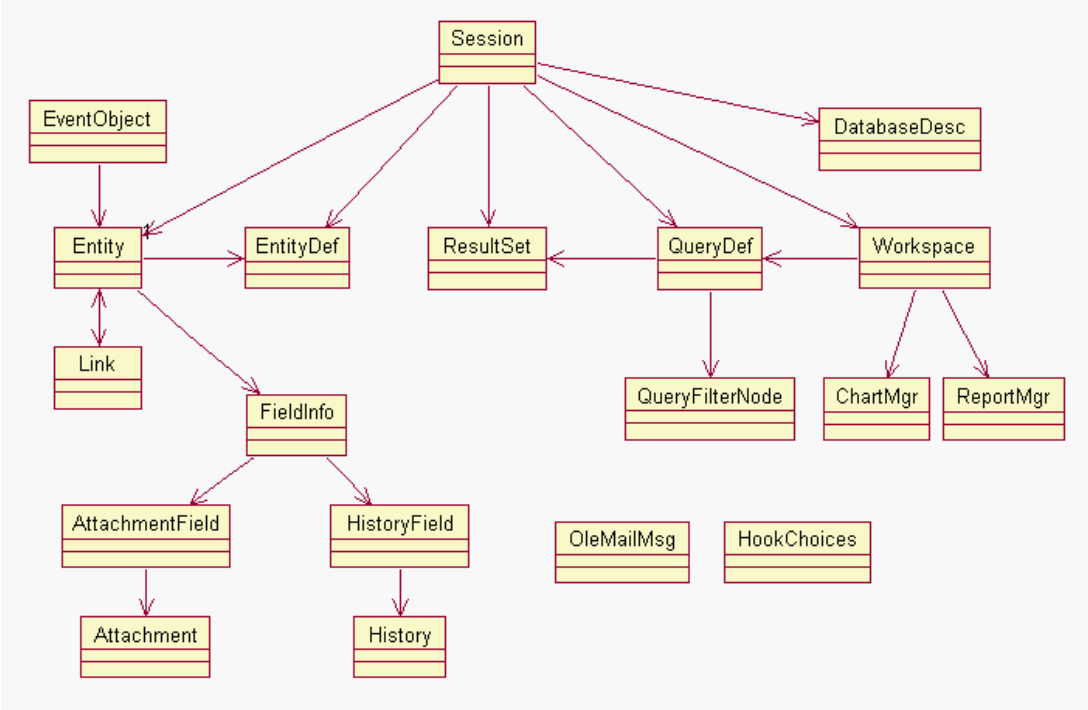
  FieldInfo objects are used to return value information for fields of entities. A FieldInfo object contains the name, value, and validity information for the field. These objects contain snapshots of values; they do not change as the entity is updated.

- Acquire the metadata associated with the Entity object to determine the structure of the record. This information is contained in the associated EntityDef (record type) object.

User Database Objects

User database objects are the objects your code works with the most, from a given Session.

The following diagram illustrates 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 different types of objects such as DatabaseDesc, Entity, EntityDef, QueryDef, and ResultSet directly.



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.

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
EntityDefs Object	Collection of EntityDef (record type) objects
QueryDef Object	Defines the query criteria.
ResultSet Object	Contains the data the query fetches
QueryFilterNode Object	Implements comparison filters for the query

A QueryDef is the definition of a query.

A ResultSet is the result of a query. There are two steps:

- First, the result set is created from a QueryDef (this is like compiling the query).
- Next, the result set is executed to get actual results. If it is a parameterized query, then the ResultSet is used to fill in values for the queries.

Information Objects

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

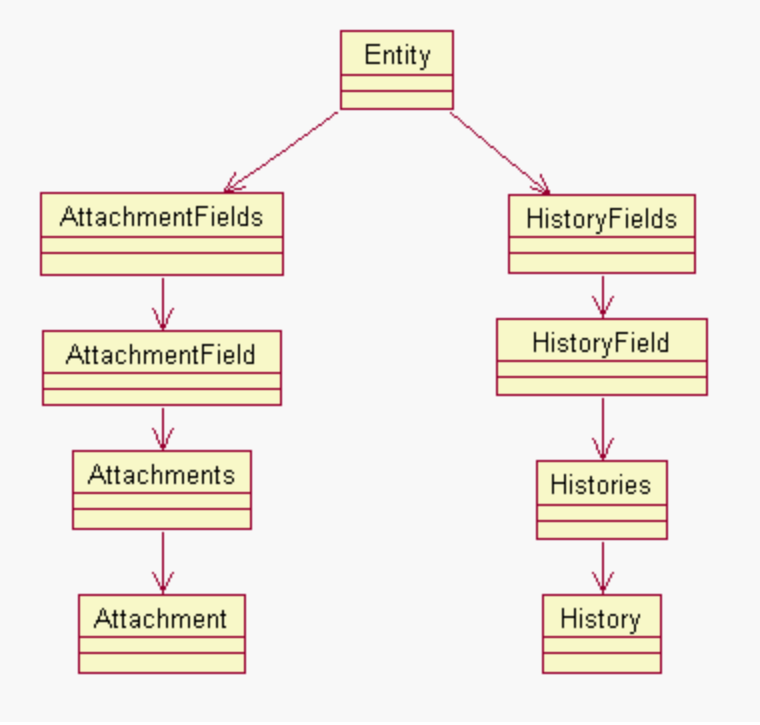
Information Objects	Description
DatabaseDesc Object	Provides information about a given database, including whether it is a schema repository or a user database.
EventObject Object	Provides read-only context information about an Entity's (that is, a record's) invocation method (or named hook).
FieldInfo Object	Provides read-only information about a field in a user database record (for example, what value it currently stores).

Attachments and Histories

Two read-only properties for the Entity object are:

- AttachmentFields (GetAttachmentFields for Perl)
- HistoryFields (GetHistoryFields for Perl)

Collections are used to represent Attachment and History related objects. A collection is a group of objects of same type. The following diagram shows the relationship between object Entity and the attachment and history related objects.



Attachment Objects

In ClearQuest the user can attach files to a record (that is, an Entity object) 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.

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

The AttachmentFields object is a collection of AttachmentFields. 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. This object contains all collections the corresponding attachment field has.

An Attachment object contains a single attached file. An Attachment object contains information about a particular attachment such as its description, size, and provides ways to manipulate the attachment.

For more information, see **`AttachmentFields`** of the **Entity Object.**

History Objects

In ClearQuest a record (that is, an Entity object) has history information associated with it. Each record type (EntityDef) may have 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.

History Objects	Description
Histories Object	Collection object that contains all History objects for a record
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	Collection object that contains all history-related objects for a record

The HistoryFields object is the container object for all of the other objects and is a collection of HistoryField 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. HistoryField contains information about a history field.

The Histories object is a container object that stores one or more History objects. This object contains all collections the corresponding history field has. A Histories object is always associated with a single HistoryField object.

A History object contains a string that describes the modifications to the record. History contains information about a particular history such as its description, size, and provides ways to manipulating the History.

For more information, see **`HistoryFields`** of the **Entity Object.**

Additional Objects

Additional objects provide APIs to list choices, links between records, e-mail 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 e-mail notification hook. A CQMailMsg is the Perl mail message object.
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

A Link object is a link between an original and a duplicate Entity. If you visualize the duplicate relationships as a tree with entities as the nodes, then links are the edges between the nodes.

Understanding AdminSession Objects

The AdminSession object and its sub-objects provide services to the schema repository (master database) for administrative purposes. The ClearQuest schema repository is the *master* database that contains your schemas.

AdminSession is the root object of all master database-related operations such as:

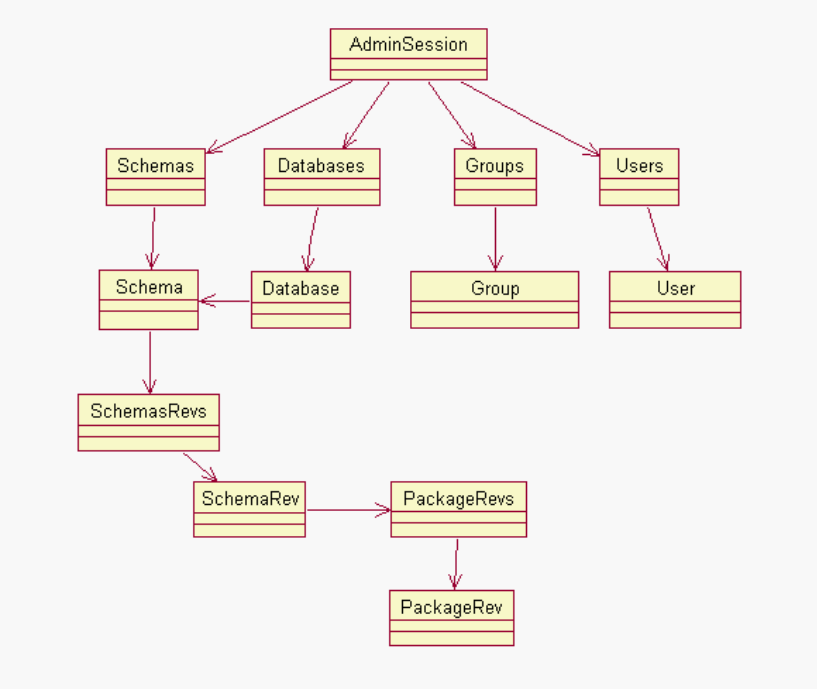
- Creating databases

- Deleting databases
- Subscribing users to a database
- Subscribing groups to a database

The AdminSession object must be created before any operation can be performed. There is no overlap functionality between this object and the Session object.

Schema Repository Objects

Schema repository (master database) objects allow you to get and set certain kinds of metadata. The following diagram illustrates these objects you can work with, from a given AdminSession.



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 Object	Each schema in the schema repository is represented by a **SchemaRev 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.

Schema Repository Object	Description
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.
PackageRev Object	A PackageRev is an object that contains information about a particular version of a Package. A ClearQuest package is a version-based package. Packages are currently not exposed through the API.

A SchemaRev object contains information about a particular version of a Schema. A ClearQuest schema uses a version-based mechanism. A Schema can have multiple versions associated with it.

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
Groups Object	Collection of user database groups
PackageRevs	Collection of package revision objects in the schema repository
Schemas Object	Collection of schemas in the schema repository
SchemaRevs Object	Collection of schema revision objects in the schema repository
Users Object	Collection of user database users

A master database may contain more than one database Schema. A Schemas object can be used to represent a list of schemas.

For information about accessing the schema repository see “Accessing the Schema Repository” on page 20 and “Performing User Administration” on page 21.

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. After 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 stand-alone 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	$currentSession = CQSession::Build(); $currentAdminSession= CQAdminSession::Build(); When you are done with the object, destroy it: CQSession::Unbuild($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. Use methods of the **DatabaseDesc Object** to get specific database information such as the name of a database or the database set (a schema repository and its associated databases) to which a database belongs.

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`** and **`OutputDebugString`**.

Using Sessionwide Variables

Session variables are hook variables that are global to the entire logon session. This means you can set the session variable in any type of hook, and read it later on, again in any type of hook. The value persists for the whole session.

ClearQuest supports the use of sessionwide variables for storing information. After you create sessionwide 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.

In order to:

- Access sessionwide variables, use the **`NameValue`** method of the Session object.
- Create a new variable, pass a new name and value to the **`NameValue`** method. 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.
- 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.

Examples

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);
```

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")
```

Consider the following example in VBScript. If you want to find out the current action name in a field validation hook. You can use the GetActionName method, or use a session variable.

In every action initialization hook, the current action is passed in the parameter, "actionname." So, you can set a session variable, called "ActionName" to the value in actionname with the following code:

```
set session = GetSession
session.NameValue "ActionName", actionname
```

Then, in the field validation hook, you can retrieve the current value of the session variable ActionName into actionname with:

```
set session = GetSession
actionname = session.NameValue("ActionName")
' ...
```

Using VBScript, you can also store objects in a session variable. Note that you use 'set' to store objects. For example:

```
set sessionObj.NameValue ("Obj") = object
```

or

```
set sessionObj.NameValue ("CalendarHandle") = param.ObjectItem
```

In the above example, param is the parameter to a record script hook and contains an object handle. See **`NameValue`**, **`ObjectItem`**, and **`Understanding Record Scripts`** for more information.

Ending a Session

Hooks are attached to events that occur when a user interacts with ClearQuest. Because hooks execute at predefined times during the middle of a session, 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.

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.

- For Perl, you must destroy it using Unbuild. For example:

  ```
  CQSession::Unbuild($currentSession);
  ```

- For VBScript, the session ends when 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 create and run a query to fetch data from the ClearQuest database according to the search criteria that you provide in the query. To build a query:

1. Build a query (QueryDef) to specify what data you want. The QueryDef object contains the definition of a query for a ClearQuest database. After a QueryDef is created, you can use it to get information from the database.
2. Create a result set (ResultSet 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 ResultSet object.

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 you 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. (For more information, see "Navigating Through the Result Set" on page 17.)

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" on page 817.

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 (for more information, see "Creating Queries" on page 15). 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. For more information, see "Working with Records" on page 18.

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 **`BuildField`**.
a SQL statement	the SQL statement (To discover which column has the data you want, use the ResultSet object: **`GetNumberOfColumns`**, **`GetColumnType`**, and **`GetColumnLabel`**.)

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 (EntityDef), 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 any defined field types of the record, including a list of the duplicates of the record, the history of the record, and any files attached to the record (if these types of fields are defined for the record type).

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" on page 15.

Getting Entity Objects

To obtain an existing Entity object whose ID you know, you can use the Session object's **`GetEntity`** or **`GetEntityByDbId`** methods. 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`**, **`GetValue`** **`SetFieldValue`**
Validate and commit the record	**`Validate`**, **`Commit`**
Determine which fields must be filled in by the user	**`GetFieldRequiredness`**
Determine the acceptable values for each field, and which fields have invalid values	**`GetFieldType`**, **`GetInvalidFieldValues`**
Determine which fields have been updated	**`GetFieldsUpdatedThisAction`**, **`GetFieldsUpdatedThisGroup`**, **`GetFieldsUpdatedThisSetValue`**
Find other data records that are considered duplicates of this one	**`GetDuplicates`**
Find the original data record, if this one is a duplicate	**`GetFieldOriginalValue`**

Creating a New Record

To create a new record, call the **`BuildEntity`** method of the Session object. The BuildEntity method creates a new Entity object 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 the default values in the Entity object.

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 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**.

Names	Session Object Method
All record types	**`GetEntityDefNames`**
Record types that have states	**`GetReqEntityDefNames`**
Record types that are stateless	**`GetAuxEntityDefNames`**
Record types that belong to a record type family	**`GetQueryEntityDefNames`**
Record types you can use to create a new record	**`GetSubmitEntityDefNames`**

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

Ensuring that Record Data Is Current

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

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" on page 21, for example, is among such tasks.

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 **`Logon`**.

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. Note that the user or group must be subscribed to the database before doing the upgrade.

Performing User Administration

You can perform user administration and update the database from ClearQuest Designer. You can use either the User administration dialog box 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, e-mail address, and access privileges. You can also subscribe the user to one or more databases.

In order 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`** method.
- Create a new group, call the **`CreateGroup`** method. This method returns a new **Group object**, to which you can add new users.
- Get an existing group, call the **`GetGroup`** method, or iterate through the **`Groups`**.
- 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. After 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
Get the name of the current user	**`GetUserLoginName`**
Get a list of groups to which the user belongs	**`GetUserGroups`**
Get a user's e-mail address	**`GetUserEmail`**
Get a user's full name	**`GetUserFullName`**
Get a user's phone number	**`GetUserPhone`**
Get any additional information about the user	**`GetUserMiscInfo`**

AdminSession Object

2

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:

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

This creates an uninitialized AdminSession object. To use it you have to log into the database using the AdminSession.Logon method. This method logs you into the master database in the specified database set. It takes the following arguments (the argument values are strings):

```
Logon login_name, password, databaseSetName
```

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

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

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

Note: To learn about user administration, see “Performing User Administration” on page 21.

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`** method to find the one you want. You can also disassociate a user database from the schema repository by calling the **`DeleteDatabase`** method.

See Also

"Accessing the Schema Repository" on page 20
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`	Read-only	Returns the collection of databases associated with the schema repository.
`Groups`	Read-only	Returns the collection of groups associated with the schema repository.
`Schemas`	Read-only	Returns the collection of schemas associated with the schema repository.
`Users`	Read-only	Returns the collection of users associated with the schema repository.

Databases

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**.

Syntax

VBScript

adminSession.**Databases**

Perl

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

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

Example

VBScript

```
set adminSession = CreateObject("ClearQuest.AdminSession")
set SessionObj = CreateObject("ClearQuest.Session")
adminSession.Logon "admin", "admin", ""
set databaseList = adminSession.Databases
For each dbObj in databaseList
   dbName = dbObj.DatabaseName
   SessionObj.OutputDebugString "Found database: " & dbName
Next
```

Perl

```
use CQPerlExt;

#Create a ClearQuest admin session
$adminSession= CQAdminSession::Build();

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

$dbList = $adminsession->GetDatabases();

#Get the number of databases
$numDbs = $databaseList->Count();

#Iterate through the databases
for ( $x=0; $x<$numDbs; $x++ ) {
     #Get the specified item in the collection of databases
     $dbObj = $databaseList->Item( $x );
     #Get the name of the database
     $dbName = $dbObj->GetName();
}

CQAdminSession::Unbuild($adminSession);
```

See Also `GetDatabase`
Database Object
Databases Object

Groups

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**.

Syntax

VBScript

adminSession.**Groups**

Perl

$adminSession->**GetGroups**;

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

Example

VBScript

```
set adminSession = CreateObject("ClearQuest.AdminSession")
adminSession.Logon "admin", "", ""
set groupList = adminSession.Groups
for each groupObj in groupList
   groupName = groupObj.Name
   msgbox groupName
Next
```

Perl

```
use CQPerlExt;

#Create a ClearQuest admin session
$adminSession= CQAdminSession::Build();

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

#Get the list of groups
$groupList = $adminSession->GetGroups();

#Get the number of groups
$numGroups = $groupList->Count();

#Iterate through the groups
for ( $x=0; $x<$numGroups; $x++ ) {
     #Get the specified item in the collection of groups
     $groupObj = $groupList->Item( $x );
     #Get the name of the group
     $groupName = $groupObj->GetName();
}

CQAdminSession::Unbuild($adminSession);
```

See Also `GetGroup`
Group Object
Groups Object
“Adding and Removing Users in a Group” on page 857.

Schemas

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**.

Syntax

VBScript

`adminSession.`**Schemas**

Perl

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

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

Example

VBScript

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

set schemaList = adminSession.Schemas
For each schemaObj in schemaList
     schemaName = schemaObj.Name
     SessionObj.OutputDebugString "Found schema: " & schemaName
Next
```

Perl

```
use CQPerlExt;

#Create a ClearQuest admin session
$adminSession = CQAdminSession::Build();

$SessionObj = CQSession::Build();

#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;
```

```
    $SessionObj->OutputDebugString( $debugString );
}
CQSession::Unbuild($SessionObj);
CQAdminSession::Unbuild($adminSession);
```

See Also

Schema Object
Schemas Object

Users

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**.

Syntax

VBScript

adminSession.**Users**

Perl

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

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

Example

VBScript

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

adminSession.Logon "admin", "admin", ""

set userList = adminSession.Users

For each userObj in userList
  userName = userObj.Name
  SessionObj.OutputDebugString "Found user: " & userName
Next
```

Perl

```
use CQPerlExt;

#Create a ClearQuest admin session
$adminSession= CQAdminSession::Build();

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

#Get the list of users in the repository.
$userList = $adminSession->GetUsers();

#Get the number of users
$numUsers = $userList->Count();

#Iterate through the users
for ( $x=0; $x<$numUsers; $x++ ) {
     #Get the specified item in the collection of users
     $userObj = $userList->Item( $x );
     #Get the name of the user
     $userName = $userObj->GetName();
}

CQAdminSession::Unbuild($adminSession);
```

See Also `GetUser`
User Object
Users Object
“Adding and Removing Users in a Group” on page 857.

AdminSession Object Methods

The following list summarizes the AdminSession object methods:

Note: For all Perl Get and Set methods that map to Visual Basic Properties, see the Properties section of this object.

Method Name	Description
`Build`	(Perl only) Creates an AdminSession object.
`CQDataCodePageIsSet`	Returns whether the ClearQuest data code page has been set, or not.
`CreateDatabase`	Creates a new database and associates it with the schema repository.
`CreateGroup`	Creates a new group and associates it with the schema repository.
`CreateUser`	Creates a new user and associates it with the schema repository.
`DeleteDatabase`	Disassociates the specified database from the schema repository.
`GetClientCodePage`	Returns a String describing the client's code page.
`GetCQDataCodePage`	Returns a String describing the ClearQuest data code page.
`GetDatabase`	Returns the database object with the specified name.
`GetGroup`	Returns the group object with the specified name.
`GetLastSchemaRepoInfo`	Returns schema repository information for the current connection.
`GetLastSchemaRepoInfoByDbSet`	Returns schema repository information for the current connection, given a database set name.
`GetLocalReplicaName`	Returns the name of the local replica.
`GetReplicaNames`	Returns a reference to the replica names.
`GetUser`	Returns the user object with the specified name.
`IsClientCodePageCompatibleWithCQDataCodePage`	Returns whether the client code page is compatible with the ClearQuest data code page, or not.
`IsMultisiteActivated`	Returns a boolean indicating whether the current database has been activated for multisite operations.
`IsReplicated`	Returns a boolean indicating whether the current database has at lease two replicated sites.

Method Name	Description
`IsStringInCQDataCodePage`	Returns whether, or not, the ClearQuest data code page contains a given String.
`IsUnsupportedClientCodePage`	Returns whether the client code page is unsupported, or not.
`Logon`	Logs the specified user into the schema repository.
`RegisterSchemaRepoFromFile`	Creates a new dbset, given a file path.
`RegisterSchemaRepoFromFileByDbSet`	Creates a new dbset, given a database set name.
`Unbuild`	(Perl only) Deletes an AdminSession object, when you are done with it.
`ValidateStringInCQDataCodePage`	Checks to see if a given String is in the ClearQuest data code page for the session's schema-repository.

Additional Perl Get and Set Methods that map to Visual Basic properties:

Method Name	Description
`GetDatabases`	Returns the collection of databases associated with the schema repository.
`GetGroups`	Returns the collection of groups associated with the schema repository.
`GetSchemas`	Returns the collection of schemas associated with the schema repository.
`GetUsers`	Returns the collection of users associated with the schema repository.

Build

Description Creates an AdminSession object.

Note: This method is for Perl only.

Syntax

Perl

```
CQAdminSession::Build();
```

Identifier	Description
CQAdminSession	A static function specifier for a AdminSession object.
Return value	A newly created AdminSession object.

Example

Perl

```
use CQPerlExt;

my $AdminSessionObj = CQAdminSession::Build();

CQAdminSession::Unbuild($AdminSessionObj);
```

See Also

Unbuild
Session Object

CQDataCodePageIsSet

Description Returns whether the ClearQuest data code page has been set, or not.

Syntax

VBScript

*adminSession.***CQDataCodePageIsSet**

Perl

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

Identifier	Description
adminSession	The AdminSession object representing the current schema repository access session.
Return value	Returns True if the ClearQuest data code page has been set, False otherwise.

Examples

VBScript

```
isSet = adminSession.CQDataCodePageIsSet
```

Perl

```
$isSet = $adminSession->CQDataCodePageIsSet();
```

See Also

GetClientCodePage
GetCQDataCodePage
IsClientCodePageCompatibleWithCQDataCodePage
IsStringInCQDataCodePage
IsUnsupportedClientCodePage
ValidateStringInCQDataCodePage
CQDataCodePageIsSet of the **Session Object**

CreateDatabase

Description

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

Note: This method is for Windows only.

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" on page 129 for more information on creating databases.

Syntax

VBScript

*adminSession.***`CreateDatabase`** *(databaseName)*

Perl

*$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.

Examples

VBScript

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

set newDatabaseObj = adminSession.CreateDatabase ("NEWDB")

' set up the database

' ...
```

Perl

```
use CQPerlExt;

#Create a ClearQuest admin session
$adminSession= CQAdminSession::Build();

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

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

# set up the database

#...

CQAdminSession::Unbuild($adminSession);
```

See Also

`DeleteDatabase`
`GetDatabase`
`Databases`
Database Object
`SetInitialSchemaRev` of the **Database Object**

CreateGroup

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.

Syntax

VBScript

*adminSession.***CreateGroup** *(groupName)*

Perl

*$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**.

Examples

VBScript

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

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

Perl

```
use CQPerlExt;

#Create a ClearQuest admin session
$adminSession= CQAdminSession::Build();

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

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

#...

CQAdminSession::Unbuild($adminSession);
```

See Also

GetGroup
Groups
Group Object
ApplyPropertyChanges of the **Database Object**

CreateUser

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**.

Syntax

VBScript

*adminSession.***CreateUser** *(userName)*

Perl

*$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**.

Examples

VBScript

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

Perl

```
use CQPerlExt;

# Create a ClearQuest admin session
my $adminSession = CQAdminSession::Build();

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

# Create the user "jsmith" object
my $newUserObj = $adminSession->CreateUser( "jsmith" );
die "Unable to create the user!\n" unless $newUserObj;

# Set the new user's password to secret
$newUserObj->SetPassword("secret");

# All done.
CQAdminSession::Unbuild($adminSession);
```

See Also

`GetUser`
`Users`
User Object
`Password` of the `User Object`

DeleteDatabase

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.

Syntax

VBScript

adminSession.**DeleteDatabase** *databaseName*

Perl

*$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	None.

Examples

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"
```

Perl

```
use CQPerlExt;

#Create a ClearQuest admin session
$adminSession = 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" );

CQAdminSession::Unbuild($adminSession);
```

See Also

CreateDatabase
GetDatabase
Databases
Database Object

GetClientCodePage

Description Returns a String describing the client's code page (for example, "1252 (ANSI - Latin I)" or "20127 (US-ASCII)"). This method does not require the Session to be logged in.

Syntax

VBScript

*adminSession.*__GetClientCodePage__

Perl

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

Identifier	Description
adminSession	The AdminSession object representing the current schema repository access session.
Return value	Returns a String describing the client's code page.

Examples

VBScript

```
myClientCP = adminSession.GetClientCodePage
```

Perl

```
$myClientCP = $adminSession->GetClientCodePage();
```

See Also

CQDataCodePageIsSet
GetCQDataCodePage
IsClientCodePageCompatibleWithCQDataCodePage
IsStringInCQDataCodePage
IsUnsupportedClientCodePage
ValidateStringInCQDataCodePage
CQDataCodePageIsSet of the **Session Object**

GetCQDataCodePage

Description Returns a String describing the ClearQuest data code page (for example, "1252 (ANSI - Latin I)" or "20127 (US-ASCII)").

Syntax

VBScript

*adminSession.*__GetCQDataCodePage__

Perl

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

Identifier	Description
adminSession	The AdminSession object representing the current schema repository access session.
Return value	Returns a String describing the ClearQuest data code page.

Examples

VBScript

```
myCQDataCP = adminSession.GetCQDataCodePage
```

Perl

```
$myCQDataCP = $adminSession->GetCQDataCodePage();
```

See Also

CQDataCodePageIsSet
GetClientCodePage
IsClientCodePageCompatibleWithCQDataCodePage
IsStringInCQDataCodePage
IsUnsupportedClientCodePage
ValidateStringInCQDataCodePage
CQDataCodePageIsSet of the **Session Object**

GetDatabase

Description

Returns the database object with the specified name.

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

Syntax

VBScript

```
adminSession.GetDatabase (databaseName)
```

Perl

```
$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.

Examples

VBScript

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

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

Perl

```
use CQPerlExt;

#Create a ClearQuest admin session
$adminSession= CQAdminSession::Build();

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

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

#...

CQAdminSession::Unbuild($adminSession);
```

See Also

`CreateDatabase`
`Databases`
Database Object

GetGroup

Description

Returns the group object with the specified name.

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

Syntax

VBScript

adminSession.`GetGroup` *(groupName)*

Perl

$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

VBScript

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

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

Perl

```
use CQPerlExt;

#Create a ClearQuest admin session
$adminSession= CQAdminSession::Build();

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

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

#...

CQAdminSession::Unbuild($adminSession);
```

See Also

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

GetLastSchemaRepoInfo

Description

Returns schema repository information for the current connection.

It can be useful to save the schema repository connection info in a file. This is called a schema repo location file (that is, a tracking file). The name of this file is stored in the schema repo and whenever the schema repo location changes, the file is automatically updated. This method is used to save and retrieve info from the file.

Syntax

VBScript

```
adminSession.GetLastSchemaRepoInfo vendor, server, database, roLogin,
roPassword
```

Perl

```
$adminSession->GetLastSchemaRepoInfo(vendor, server, database, roLogin,
roPassword);
```

Identifier	Description
adminSession	The AdminSession object representing the current schema repository access session.
vendor	A string containing the vendor type of the database.
server	A string containing the server name of the database.
database	A string containing the database name.
roLogin	A string containing the roLogin.
roPassword	A string containing the roPassword.
Return value	A Boolean whose value is True if the operation was successful, otherwise False.

See Also

RegisterSchemaRepoFromFile
Database Object

GetLastSchemaRepoInfoByDbSet

Description

Returns schema repository information for the current connection, given a database set name.

It can be useful to save the schema repository connection info in a file. This is called a schema repo location file (that is, a tracking file). The name of this file is stored in the schema repo and whenever the schema repo location changes, the file is automatically updated. This method is used to save and retrieve info from the file.

Syntax

VBScript

```
adminSession.GetLastSchemaRepoInfoByDbSet dbset, vendor, server, database,
roLogin, roPassword
```

Perl

```
$adminSession->GetLastSchemaRepoInfoByDbSet(dbset, vendor, server, database,
roLogin, roPassword);
```

Identifier	Description
adminSession	The AdminSession object representing the current schema repository access session.
dbset	A string containing the database set name (that is, the product version number).
vendor	A string containing the vendor type of the database.
server	A string containing the server name of the database.
database	A string containing the database name.
roLogin	A string containing the roLogin.
roPassword	A string containing the roPassword.
Return value	A Boolean whose value is True if the operation was successful, otherwise False.

See Also

GetLastSchemaRepoInfo
Database Object

GetLocalReplicaName

Description (Perl only) Returns the name of the local replica.

Syntax

Perl

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

Identifier	Description
adminSession	The AdminSession object representing the current schema repository access session.
Return value	Returns a String containing name of the local replica.

Example

Perl

```
use CQPerlExt;
my $adminSess;
$adminSess = CQAdminSession::Build();
$adminSess->Logon("admin", "", "CQMS.MS_ACCESS.SITEA");

if ($adminSess->IsReplicated()) {
    printf "\nLocal replica is %s.\n", $adminSess->GetLocalReplicaName();
}
#...
CQAdminSession::Unbuild($adminSess);
```

See Also **GetReplicaNames**

GetReplicaNames

Description (Perl only) Returns a reference to the replica names.

Syntax

Perl

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

Identifier	Description
adminSession	The AdminSession object representing the current schema repository access session.
Return value	A reference to an array of strings containing all of the replica names.

Example

Perl

```
use CQPerlExt;
my $adminSess;
$adminSess = CQAdminSession::Build();
$adminSess->Logon("admin", "", "CQMS.MS_ACCESS.SITEA");
if ($adminSess->IsMultisiteActivated()) {
   my $replicNames;
   my $replicName;
   $replicaNames = $adminSess->GetReplicaNames();
   foreach $replicaName (@$replicaNames) {
      printf $replicaName. "\n";
   }
}
#...
CQAdminSession::Unbuild($adminSess);
```

See Also **GetLocalReplicaName**

GetUser

Description

Returns the user object with the specified name.

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

Syntax

VBScript

```
adminSession.GetUser (userName)
```

Perl

```
$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

VBScript

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

set userObj = adminSession.GetUser ("Dev")
```

Perl

```
use CQPerlExt;

#Create a ClearQuest admin session
$adminSession= CQAdminSession::Build();

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

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

#...

CQAdminSession::Unbuild($adminSession);
```

See Also

Users
CreateUser
Users
Name of the **User Object**
Password of the **User Object**

IsClientCodePageCompatibleWithCQDataCodePage

Description

Returns whether the client code page is compatible with the ClearQuest data code page, or not. This returns True if all characters in the ClearQuest data code page are present in the client code page. If this method returns False, ClearQuest will not let you edit anything in the database (that is, you will have read-only access).

Syntax

VBScript

*adminSession.***IsClientCodePageCompatibleWithCQDataCodePage**

Perl

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

Identifier	Description
adminSession	The AdminSession object representing the current schema repository access session.
Return value	Returns True if the client code page is compatible with the ClearQuest data code page, False otherwise.

Examples

VBScript

```
isComp = adminSession.IsClientCodePageCompatibleWithCQDataCodePage
```

Perl

```
$ isComp = $adminSession->IsClientCodePageCompatibleWithCQDataCodePage();
```

See Also

CQDataCodePageIsSet
GetClientCodePage
GetCQDataCodePage
IsStringInCQDataCodePage
IsUnsupportedClientCodePage
ValidateStringInCQDataCodePage
CQDataCodePageIsSet of the **Session Object**

IsMultisiteActivated

Description

Returns True if the current database has been activated for multisite operations. This method returns True even if the current database is the only exisiting replica.

Syntax

VBScript

*adminSession.*__IsMultisteActivated__

Perl

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

Identifier	Description
adminSession	The AdminSession object representing the current schema repository access session.
Return value	Returns True if the current database has been activated for multisite operations.

Examples

Perl

```
use CQPerlExt;
my $adminSess;
$adminSess = CQAdminSession::Build();
$adminSess->Logon("admin", "", "CQMS.MS_ACCESS.SITEA");

if ($adminSess->IsMultisiteActivated()) {
   my $replicNames;
   my $replicName;
   $replicaNames = $adminSess->GetReplicaNames();
   foreach $replicaName (@$replicaNames) {
      printf $replicaName. "\n";
   }
}
CQAdminSession::Unbuild($adminSess);
```

See Also

IsReplicated
GetReplicaNames

IsReplicated

Description Returns True if the current database has at lease two replicated sites, False otherwise.

Syntax

VBScript

*adminSession.*__IsReplicated__

Perl

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

Identifier	Description
adminSession	The AdminSession object representing the current schema repository access session.
Return value	Returns True if the current database has at lease two replicated sites, False otherwise.

Examples

Perl

```
use CQPerlExt;
my $adminSess;
$adminSess = CQAdminSession::Build();
$adminSess->Logon("admin", "", "CQMS.MS_ACCESS.SITEA");

if ($adminSess->IsReplicated()) {
    printf "\nLocal replica is %s.\n", $adminSess->GetLocalReplicaName();
}
CQAdminSession::Unbuild($adminSess);
```

See Also **IsMultisiteActivated**
GetLocalReplicaName

IsStringInCQDataCodePage

Description

Returns whether, or not, the ClearQuest data code page contains a given String.

This method takes a String argument and checks to see if the String is in the ClearQuest data code page for the Session's schema-repository.

Syntax

VBScript

*adminSession.*__IsStringInCQDataCodePage__ *stringToCheck*

Perl

$adminSession->*__IsStringInCQDataCodePage__*($stringToCheck);

Identifier	Description
adminSession	The AdminSession object representing the current schema repository access session.
stringToCheck	A String that specifies what you are checking to see is in the ClearQuest data code page.
Return value	Returns True if the ClearQuest data code page contains a given String, False otherwise.

Examples

VBScript

```
IsInCodePage = adminSession.IsStringInCQDataCodePage stringToCheck
```

Perl

```
$isInCodePage = $adminSession->IsStringInCQDataCodePage($stringToCheck);
```

See Also

CQDataCodePageIsSet
GetClientCodePage
GetCQDataCodePage
IsClientCodePageCompatibleWithCQDataCodePage
IsUnsupportedClientCodePage
ValidateStringInCQDataCodePage
CQDataCodePageIsSet of the **Session Object**

IsUnsupportedClientCodePage

Description

Returns whether the client code page is unsupported, or not. You do not need to login to use this method.

Note: The client code page is separate from the ClearQuest data code page.

Syntax

VBScript

adminSession.**IsUnsupportedClientCodePage**

Perl

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

Identifier	Description
adminSession	The AdminSession object representing the current schema repository access session.
Return value	Returns True if the client code page is unsupported, False otherwise.

Examples

VBScript

```
isUnsupported = adminSession.IsUnsupportedClientCodePage
```

Perl

```
$isUnsupported = $adminSession->IsUnsupportedClientCodePage();
```

See Also

CQDataCodePageIsSet
GetClientCodePage
GetCQDataCodePage
IsClientCodePageCompatibleWithCQDataCodePage
IsStringInCQDataCodePage
ValidateStringInCQDataCodePage
CQDataCodePageIsSet of the **Session Object**

Logon

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 (using ClearQuest Designer not the API). 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.

Note: Any active user can create an AdminSession, but the privileges may be restricted.

Syntax

VBScript

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

Perl

$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.

Examples

VBScript

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

Perl

```
use CQPerlExt;

#Create a ClearQuest admin session
$adminSession= CQAdminSession::Build();

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

#...

CQAdminSession::Unbuild($adminSession);
```

See Also

CreateUser
Users
UserLogon of the **Session Object**

RegisterSchemaRepoFromFile

Description Creates a new dbset using the file path argument. Returns an error message if an error occurred.

Note: Session doesn't need to be logged in.

Syntax

VBScript

adminSession.`RegisterSchemaRepoFromFile` *filePath*

Perl

$adminSession->`RegisterSchemaRepoFromFile` *(filePath);*

Identifier	Description
adminSession	The AdminSession object representing the current schema repository access session.
filePath	A String containing the file path to the connection information.
Return value	A Boolean whose value is True if the operation was successful, otherwise False.

See Also `Schemas`

`GetLastSchemaRepoInfo`

RegisterSchemaRepoFromFileByDbSet

Description

Creates a new dbset using the databaseset name argument. Returns an error message if an error occurred.

Same as RegisterSchemaRepoLocationFile, except a new database set is created using the name in the *dbset* input parameter.

Note: Session does not need to be logged in.

Syntax

VBScript

*adminSession.*__RegisterSchemaRepoFromFileByDbSet__ *dbset, filePath*

Perl

*$adminSession->*__RegisterSchemaRepoFromFileByDbSet__*(dbset, filePath);*

Identifier	Description
adminSession	The AdminSession object representing the current schema repository access session.
dbset	A String containing the name of the new dbset.
filePath	A String containing the file path to the connection information.
Return value	A Boolean whose value is True if the operation was successful, otherwise False.

See Also

`RegisterSchemaRepoFromFile`

Unbuild

Description Deletes the AdminSession object you explicitly created with the Build method, when you do not need it anymore.

Note: This method is for Perl only.

Syntax

Perl

CQAdminSession::**Unbuild**(AdminSessionObj);

Identifier	Description
CQAdminSession	A static function specifier for the AdminSession object.
AdminSessionObj	The AdminSession object that you are deleting.
Return value	None.

Example

Perl

```
use CQPerlExt;

my $AdminSessionObj = CQAdminSession::Build();

CQAdminSession::Unbuild($AdminSessionObj);
```

See Also

Build
Session Object

ValidateStringInCQDataCodePage

Description

Checks to see if a given String is in the ClearQuest data code page for the session's schema-repository. If the String is not in the code page, it returns an error message for display to the user. The error message includes which characters (up to the first five characters) were not in the ClearQuest data code page.

Syntax

VBScript

adminSession. **ValidateStringInCQDataCodePage** *stringToCheck*

Perl

$adminSession-> **ValidateStringInCQDataCodePage** *($stringToCheck);*

Identifier	Description
adminSession	The AdminSession object representing the current schema repository access session.
stringToCheck	A String that specifies what you are checking to see is in the ClearQuest data code page.
Return value	Returns an empty String if the *stringToCheck* is valid, otherwise, it returns a displayable error message

Examples

VBScript

```
validation = adminSession.ValidateStringInCQDataCodePage stringToCheck
```

Perl

```
$validation = $adminSession->ValidateStringInCQDataCodePage($stringToCheck);
```

See Also

CQDataCodePageIsSet
GetClientCodePage
GetCQDataCodePage
IsClientCodePageCompatibleWithCQDataCodePage
IsStringInCQDataCodePage
IsUnsupportedClientCodePage
CQDataCodePageIsSet of the **Session Object**
Validate of the **Entity Object**

Attachment Object

3

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

In order to:

- Attach files to a database, use the **`Add`** method of the **Attachments Object**. (You never create instances of Attachment directly.)
- Retrieve an Attachment object, use the **`Item`** method of the Attachments object.
- Delete an Attachment object, use the **`Delete`** method of the **Attachments Object**.
- Copy an existing attachment to a new file, use the **`Load`** method.

See Also

"Attachments and Histories" on page 7
AttachmentField Object
AttachmentFields Object
Attachments Object
"Getting and Setting Attachment Information" on page 810.

Attachment Object Properties

The following list summarizes the Attachment object properties:

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

Description

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.

Syntax

VBScript

attachment.**Description**

attachment.**Description** *description*

Perl

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

*$attachment->***SetDescription***(description);*

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

Example

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
For each attachment in attachments
     description = attachment.Description
     key = attachment.DisplayName
     currentSession.OutputDebugString "Unique key: " & key & " - _
     description: " & description
Next
```

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`

"Getting and Setting Attachment Information" on page 810.

DisplayName

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.

Syntax

VBScript

attachment.**DisplayName**

Perl

*$attachment->***GetDisplayName***()*

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

Example

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
For each attachment in attachments
     description = attachment.Description
     key = attachment.DisplayName
     currentSession.OutputDebugString "Unique key: " & key & " - _
           description: " & description & description
Next
```

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`

`FileName`

"Getting and Setting Attachment Information" on page 810.

FileName

Description

Returns the path name of the attached file.

This is 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, after 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.

Syntax

VBScript

attachment.**FileName**

Perl

$attachment->.**GetFileName()**

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

Example

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
For each attachment in attachments
     fileName = attachment.FileName
     fileSize = attachment.FileSize
     currentSession.OutputDebugString "Attached file: " & fileName &
                " - size: " & fileSize
Next
```

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`
`Add` of the **Attachments Object**
`Commit` of the **Entity Object**
"Getting and Setting Attachment Information" on page 810.

FileSize

Description

Returns the size of the attached file in bytes.

This is 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.

Syntax

VBScript

attachment.**FileSize**

Perl

*$attachment->*GetFileSize();

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

Example

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
For each attachment in attachments
     set fileName = attachment.FileName
     set fileSize = attachment.FileSize
     OutputDebugString "Attached file: " & fileName & " - size: " _
           & fileSize
Next
```

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`**

"Getting and Setting Attachment Information" on page 810.

Attachment Object Methods

The following list summarizes the Attachment object methods:

Note: For all Perl Get and Set methods that map to Visual Basic Properties, see the Properties section of this object.

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

Additional Perl Get and Set Methods that map to Visual Basic properties:

Method Name	Description
GetDescription	Returns the description of the attached file.
GetDisplayName	Returns the unique key that identifies the attached file.
GetFileName	Returns the name of the attached file in bytes.
GetFileSize	Returns the size of the attached file in bytes.
SetDescription	Adds a description for the attached file.

Load

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 be 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.

Syntax

VBScript

attachment.**Load** *filename*

Perl

$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 value	A Boolean whose value is True if the operation was successful, otherwise False.

Examples

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
For each attachment in attachments
     fileName = "C:\attach" & x & ".txt"

' Write the file

   status = attachment.Load (filename)
Next
```

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
 $status = $attachment->Load($filename);
 }
```

See Also **`Add`** of the **Attachment Object**
Attachments Object
"Getting and Setting Attachment Information" on page 810.

AttachmentField Object

4

An AttachmentField object represents one attachment field in a record. A record can have more than one field of type attachment list (that is, a FieldType enumeration of 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

"Attachments and Histories" on page 7
AttachmentFields Object
Attachments Object
"Getting and Setting Attachment Information" on page 810.

AttachmentField Object Properties

The following list summarizes the AttachmentField object properties:

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

Attachments

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`** method of the Attachments object contains the value zero.

Syntax

VBScript

`attachmentField``.`**Attachments**

Perl

`$attachmentField->``Get`**Attachments();**

Identifier	Description
attachmentField	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

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

For each attachment in attachments
     'Do something with each attachment
Next
```

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
FieldName
"Getting and Setting Attachment Information" on page 810.
Attachment Object

DisplayNameHeader

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 Rational ClearQuest Designer, not the ClearQuest API.

Syntax

VBScript

attachmentField.**DisplayNameHeader**

Perl

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

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

Example

VBScript

```
' This example assumes there is at least 1 attachment field
' associated with the record.
set sessionObj = GetSession

set attachFields = entity.AttachmentFields
set attachField1 = attachFields.Item(0)

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

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

$session = $entity->GetSession();

# 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`
`FieldName`
`DisplayName` of the Attachment object
Attachment Object
“Getting and Setting Attachment Information” on page 810.

FieldName

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.

Syntax

VBScript

name = attachmentField.**FieldName**

Perl

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

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

Example

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
```

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 field
$attachfield1 = $attachfields->Item(0);


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

See Also

Attachments
DisplayNameHeader
"Getting and Setting Attachment Information" on page 810.

AttachmentField Object Methods

The following list summarizes Perl AttachmentField object methods:

Note: For all Perl Get and Set methods that map to Visual Basic Properties, see the Properties section of this object.

Method Name	Access	Description
GetAttachments	Read-only	Returns this attachment field's collection of attachments.
GetDisplayNameHeader	Read-only	Returns the unique keys of the attachments in this field.
GetFieldName	Read-only	Returns the name of the attachment field.

AttachmentFields Object

5

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 Rational 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`**.

See Also

"Attachments and Histories" on page 7
Attachment Object
AttachmentField Object
Attachments Object
"Getting and Setting Attachment Information" on page 810.

AttachmentFields Object Properties

The following list summarizes the AttachmentFields object properties:

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

Count

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

Syntax

VBScript

collection.**Count**

Perl

*$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

VBScript

```
set attachFields = entity.AttachmentFields
numFields = attachFields.Count
For x = 0 to numFields - 1
     set oneField = attachFields.Item x
     ' ...
Next
```

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

"Getting and Setting Attachment Information" on page 810.

AttachmentFields Object Methods

The following list summarizes the AttachmentFields object methods:

Method Name	Description
`Item`	Returns the specified item in the collection.
`ItemByName`	(Perl only) Returns the specified item in the collection.

Note: For Perl methods that map to Visual Basic Properties, see the Properties section of this object.

The following list summarizes additional Perl AttachmentFields object methods:

Method Name	Access	Description
`Count`	Read-only	Returns the number of items in the collection.

Item

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

VBScript

```
collection.Item(itemNum)
collection.Item(name)
```

Perl

```
$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`** of the desired AttachmentField.
Return value	The AttachmentField object at the specified location in the collection.

Example

VBScript

```
set attachFields = entity.AttachmentFields
numFields = attachFields.Count
For x = 0 to numFields - 1
     set oneField = attachFields.Item x
     ' ...
Next
```

See Also

`Count`

"Getting and Setting Attachment Information" on page 810.

Attachments Object

6

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`** method.

See Also

"Attachments and Histories" on page 7
Attachment Object
AttachmentField Object
"Getting and Setting Attachment Information" on page 810

Attachments Object Properties

The following list summarizes the Attachments object properties:

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

Count

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

Syntax

VBScript

collection.**Count**

Perl

*$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

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 - 1
     set attachment = attachments.Item(x)

     ' Do something with the attachments
Next
```

Perl

```
#Get a list of attachments

$attachfields = $entity->GetAttachmentFields();

# Get the first attachments field

$attachfield1 = $attachfields->Item(0)

# Get the collection of attachments from the attachments field

$attachments = $attachfield1->GetAttachments();


#Get the number of attachments
numAttachments = $attachments->Count();
```

See Also

Item

"Getting and Setting Attachment Information" on page 810

Attachments Object Methods

The following list summarizes the Attachments object methods:

Method Name	Description
Add	Adds an Attachment object to the collection.
AddAttachment	Adds an Attachment object to the collection.
Delete	Deletes an attached file from the collection.
Exists	Checks if a file with the given name has already been attached.
Item	Returns the specified item in the collection.
ItemByName	(Perl only) Returns the specified item in the collection.

Note: For Perl methods that map to Visual Basic Properties, see the Properties section of this object.

The following list summarizes additional Perl Attachments object methods:

Method Name	Access	Description
Count	Read-only	Returns the number of items in the collection.

Add

Description

Adds an Attachment object to the collection.

Note: This method is for Visual Basic only.

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.

Syntax

VBScript

`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

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
```

See Also

`Count`
`AddAttachment`
`Delete`
`Item`
"Getting and Setting Attachment Information" on page 810

AddAttachment

Description

Adds an Attachment object to the collection.

Note: This method is for Perl only.

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.

Syntax

Perl

*`$attachments->`***AddAttachment**`(`*`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

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.AddAttachment("c:\\attach1.txt","attachment description"
 {

$session->OutputDebugString("Error adding attachment to record.\n");
 }
```

See Also

`Count`
`Add`
`Delete`
`Item`
"Getting and Setting Attachment Information" on page 810

Delete

Description

Deletes an attached file from the collection.

For VBScript, the argument to this method can be either a numeric index (itemNum) or a display name (displayName). For Perl, the argument must be a numeric index.

You can use the **`Count`** and **`Item`** methods to locate the correct Attachment object before calling this method.

Syntax

VBScript

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

Perl

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

Identifier	Description
attachments	An Attachments collection object, representing the set of attachments in one field of a record.
itemNum	For VBScript, a Variant that is an index into the collection. This index is 0-based and points to the file that you want to delete. For Perl, 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	For VBScript a Variant that is a display name of an item in the collection.
Return value	A Boolean that is True if the file was deleted successfully, otherwise False.

Examples

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
```

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 (!$attachmentsDelete(0))

 {
 $session->OutputDebugString("Error deleting attachment from
     record.\n");

 }
```

See Also

Count
Add
Item
“Getting and Setting Attachment Information” on page 810

Exists

Description Checks if a file with the given name has already been attached. This is used during synchronization.

Syntax

VBScript

```
attachments.Exists filename
```

Perl

```
$attachments->Exists(filename);
```

Identifier	Description
attachments	An Attachments collection object, representing the set of attachments in one field of a record.
filename	A Long that is an index into the collection. This index is 0-based and points to the file that you want to delete.
Return value	Returns True if the file has been attached to the collection, otherwise False.

See Also `Attachment Object`

Item

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

VBScript

collection.**Item** (*itemNum*)

collection.**Item** (*name*)

Perl

$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 value of the desired History object.
Return value	The Attachment object at the specified location in the collection.

Example

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)
```

See Also

Count

"Getting and Setting Attachment Information" on page 810

ChartMgr Object

7

The ChartMgr object provides an interface for creating charts.

Note: The ChartMgr object is for Windows only.

You can use this object to write external applications to execute charts defined in the Rational 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`	Read/Write	Gets or sets the Bool that indicates whether or not the chart should be created as a grayscale image.
`Height`	Read/Write	Sets or gets the height of the image.
`Interlaced`	Read/Write	Sets or gets whether or not PNG images are interlaced.
`OptimizeCompression`	Read/Write	Sets or gets whether or not the image compression is optimized.
`Progressive`	Read/Write	Sets or gets whether or not to create progressive JPEG images.
`Quality`	Read/Write	Sets or gets the quality factor used to generate the image.
`Width`	Read/Write	Sets or gets the width of the image.

Note: These properties are for Windows only.

GrayScale

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.

Syntax

VBScript

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

Perl

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

*$chartMgr->***SetGrayScale**(*isGrayScale*);

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	Returns 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
MakePNG

Height

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.

Syntax

VBScript

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

Perl

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

*$chartMgr->***SetHeight(newHeight);**

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	Returns an INT indicating the height of the image in pixels

See Also

Width
MakeJPEG
MakePNG

Interlaced

Description

Sets or returns whether or not PNG images are interlaced.

This property is used when producing PNG images with the **MakePNG** method. By default, this property is set to True.

Syntax

VBScript

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

Perl

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

*$chartMgr->***SetInterlaced(isInterlaced);**

Identifier	Description
chartMgr	The CHARTMGR object associated with the current session.
isInterlaced	A Bool indicating whether or not PNG images should be created in multiple passes.
Return value	Returns True if the MakePNG method will create an interlaced PNG image, otherwise False.

See Also

Progressive
MakePNG

OptimizeCompression

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

By default, this property is set to True.

Syntax

VBScript

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

Perl

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

*$chartMgr->***SetOptimizeCompression(compression);**

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	Returns True if the image compression should be optimized, otherwise False.

See Also `Quality`

Progressive

Description

Sets or gets whether or not to create a progressive JPEG image.

This property is used when producing JPEG images with the **`MakeJPEG`** method. By default, this property is set to False.

Syntax

VBScript

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

Perl

`$chartMgr->``GetProgressive();`

`$chartMgr->``SetProgressive(`*`isProgressive`*`);`

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	Returns True if the MakeJPEG method will create a progressive JPEG image, otherwise False.

See Also

`Interlaced`
`MakeJPEG`

Quality

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

VBScript

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

Perl

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

*$chartMgr->***SetQuality(***newValue***);**

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	Returns an INT between 1 and 100 indicating the compression factor of the image.

See Also

`OptimizeCompression`

Width

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.

Syntax

VBScript

chartMgr.**Width**
chartMgr.**Width** *newWidth*

Perl

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

*$chartMgr->***SetWidth(***newWidth***);**

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	Returns an INT indicating the new width of the image in pixels.

See Also

Height
MakeJPEG
MakePNG

ChartMgr Object Methods

The following list summarizes the ChartMgr object methods:

Note: For all Perl Get and Set methods that map to Visual Basic Properties, see the Properties section of this object.

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

Additional Perl Get and Set Methods that map to Visual Basic properties:

Method Name	Description
GetGrayScale	Returns the current gray scale setting.
GetHeight	Returns the current height setting.
GetInterlaced	Returns the current interlace setting.
GetOptimizeCompression	Returns the current compression setting.
GetProgressive	Returns the current progressive setting.
GetQuality	Returns the current quality setting.
GetWidth	Returns the current width setting.
SetGrayScale	Sets the image gray scale.
SetHeight	Sets the image height.
SetInterlaced	Sets the image interlace.
SetOptimizeCompression	Sets the image compression.
SetProgressive	Sets the image progressive.
SetQuality	Sets the image quality.
SetWidth	Sets the image width.

Note: These methods are for Windows only.

MakeJPEG

Description

Creates a JPEG file containing the image of a chart.

This image takes the data in the current result set and generates a JPEG image using the current settings.

Syntax

VBScript

chartMgr.**MakeJPEG** *pathName*

Perl

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

Identifier	Description
chartMgr	The CHARTMGR object associated with the current session.
pathName	A String containing the pathname, including the file name and location, to use when generating the image.
Return value	Returns True if the image was created successfully, otherwise False.

Example

Perl

```
use CQPerlExt;
   $session = CQSession::Build();
   $user = "admin";
   $pass = "";
   $db = "SAMPL";
   $session->UserLogon($user, $pass, $db, "");
   $wkSpc = $session->GetWorkSpace();
   $chartDef = $wkSpc->GetChartDef("Personal Queries/Sample_Chart");
   $resultSet = $session->BuildResultSet($chartDef);
   $resultSet->SetMaxRowsInMemory(2000);
   $resultSet->Execute();
   $chartMgr = $wkSpc->GetChartMgr();
   $chartMgr->SetResultSet($resultSet);
   $chartMgr->MakeJPEG("C:\\temp\\BBChart.jpg");
CQSession::Unbuild($session);
```

See Also

GetChartMgr of the **Workspace Object**
Height
Interlaced
OptimizeCompression
Progressive

Quality
Width
SetResultSet

MakePNG

Description

Creates a PNG file containing the image of a chart.

This image takes the data in the current result set and generates a PNG image using the current settings.

Syntax

VBScript

chartMgr.**MakePNG** *pathName*

Perl

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

Identifier	Description
chartMgr	The CHARTMGR object associated with the current session.
pathName	A String containing the pathname, including the file name and location, to use when generating the image.
Return value	Returns True if the image was created successfully, otherwise False.

Example

Perl

```
use CQPerlExt;
   $session = CQSession::Build();
   $user = "admin";
   $pass = "";
   $db = "SAMPL";
   $session->UserLogon($user, $pass, $db, "");
   $wkSpc = $session->GetWorkSpace();
   $chartDef = $wkSpc->GetChartDef("Personal Queries/Sample_Chart");
   $resultSet = $session->BuildResultSet($chartDef);
   $resultSet->SetMaxRowsInMemory(2000);
   $resultSet->Execute();
   $chartMgr = $wkSpc->GetChartMgr();
   $chartMgr->SetResultSet($resultSet);
   $chartMgr->MakePNG("C:\\temp\\BBChart.png");
CQSession::Unbuild($session);
```

See Also

GetChartMgr of the **Workspace Object**
Height
Interlaced
OptimizeCompression
Progressive

Quality
Width
SetResultSet

SetResultSet

Description

Sets the result set used to generate the chart.

After creating the chart manager and before creating the image file, you must specify the result set with this call. 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. The image is generated by charting the data in the given result set.

Syntax

VBScript

chartMgr.**SetResultSet** *resultSet*

Perl

$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.

See Also

GetChartMgr of the **Workspace Object**
MakeJPEG
MakePNG
Execute of the **ResultSet object**
ResultSet Object

ClearQuest Object

8

The CQClearQuest object is a top-level creatable object for Perl API. It serves as a "factory" for some of the other API objects, such as Session and AdminSession.

Note: The CQClearQuest object and its methods are for usage with Perl only.

A CQClearQuest object provides access as an application object, or point of entry into the Perl API. You can create a CQClearQuest object to directly:

- Create a Session object
- Create an AdminSession object
- Create a CQProductInfo object

To create a CQClearQuest object, you can do the following:

```
use CQPerlExt;
my $cqobject = CQClearQuest::Build();
# Do something with the ClearQuest object...
# Delete any objects that you explicitly create and do not need anymore
CQClearQuest::Unbuild($cqobject);
```

See Also

Session Object
AdminSession Object
ProductInfo Object

ClearQuest Object Methods

The following list summarizes the CQClearQuest object methods:

Method name	Description
`Build`	Creates a CQClearQuest object.
`CreateAdminSession`	Creates an AdminSession object.
`CreateProductInfo`	Creates a CQProductInfo object.
`CreateUserSession`	Creates a Session object.
`IsUnix`	Returns True if the CQClearQuest object is created in a UNIX platform.
`IsWindows`	Returns True if the CQClearQuest object is created in a Windows platform.
`SessionLogoff`	(UNIX only) Logoff a given session.
`SessionLogon`	(UNIX only) Create and login into a user session.
`Unbuild`	Deletes the CQClearQuest object, when you are done with it.

Build

Description Creates a CQClearQuest object from which you can create a Session or AdminSession object.

Syntax

Perl

```
CQClearQuest::Build();
```

Identifier	Description
CQClearQuest	A static function specifier for a CQClearQuest object.
Return value	A newly created CQClearQuest object.

Example

Perl

```
use CQPerlExt;

my $cqobject = CQClearQuest::Build();

CQClearQuest::Unbuild($cqobject);
```

See Also

Unbuild
AdminSession Object
Session Object

CreateAdminSession

Description Creates an AdminSession object.

Syntax

Perl

$ClearQuestObj->**CreateAdminSession**();

Identifier	Description
ClearQuestObj	A CQClearQuest object.
Return value	Returns an AdminSession object.

Example

Perl

```
use CQPerlExt;

my $cqobject = CQClearQuest::Build();
my $AdminSessionObj = $cqobject->CreateAdminSession();
# login to the AdminSession object
# do something with the AdminSession object...
CQClearQuest::Unbuild($cqobject);
```

See Also **AdminSession Object**
Logon method of the **AdminSession Object**

CreateProductInfo

Description Creates a CQProductInfo object. You can then use this method to retrieve product information.

Syntax

Perl

```
$ClearQuestObj->CreateProductInfo();
```

Identifier	Description
ClearQuestObj	A CQClearQuest object.
Return value	Returns a CQProductInfo object.

Example

Perl

```
use CQPerlExt;

my $cqobject = CQClearQuest::Build();
my $prodinfo = $cqobject->CreateProductInfo();
print $prodinfo->GetProductVersion(),"\n";
CQClearQuest::Unbuild($cqobject);
```

See Also **ProductInfo Object**

CreateUserSession

Description Create a Session object. After the object is created, you must login to the Session.

Syntax

Perl

*$ClearQuestObj->***CreateUserSession**();

Identifier	Description
ClearQuestObj	A CQClearQuest object.
Return value	Returns a Session object.

Example

Perl

```
use CQPerlExt;

my $cqobject = CQClearQuest::Build();
my $SessionObj = $cqobject->CreateUserSession();
# login to the Session object
# do something with the Session object...
CQClearQuest::Unbuild($cqobject);
```

See Also **Session Object**
UserLogon of the **Session Object**

IsUnix

Description Returns True if the CQClearQuest object is created in a UNIX platform.

Syntax

Perl

```
$ClearQuestObj->IsUnix();
```

Identifier	Description
ClearQuestObj	A CQClearQuest object.
Return value	Returns True if the object is created in a Unix platform, False otherwise.

Example

Perl

```
use CQPerlExt;

my $cqobject = CQClearQuest::Build();
$is_Windows_flag = $cqobject->IsUnix();

CQClearQuest::Unbuild($cqobject);
```

See Also **IsWindows**

IsWindows

Description Returns True if the CQClearQuest object is created in a Windows platform.

Syntax

Perl

```
$ClearQuestObj->IsWindows();
```

Identifier	Description
ClearQuestObj	A CQClearQuest object.
Return value	Returns True if the object is created in a Windows platform, False otherwise.

Example

Perl

```
use CQPerlExt;

my $cqobject = CQClearQuest::Build();
$is_Windows_flag = $cqobject->IsWindows();

CQClearQuest::Unbuild($cqobject);
```

See Also **IsUnix**

SessionLogoff

Description Logoff a given Session.

Note: This method is for UNIX only and is for ClearQuest integration server use only.

Syntax

Perl

```
$ClearQuestObj->SessionLogoff(SessionObj, flags);
```

Identifier	Description
ClearQuestObj	A CQClearQuest object.
SessionObj	A Session object that specifies the session you are logging off.
flags	A LONG that is reserved and not used in the current implementation.
Return value	None.

See Also `SessionLogon`

SessionLogon

Description Create and login into a user Session object.

Note: This method is for UNIX only and is for ClearQuest integration server use only.

Syntax

Perl

```
$ClearQuestObj->SessionLogon(clientID, userID, password, db, dbSet, flags);
```

Identifier	Description
ClearQuestObj	A CQClearQuest object.
clientID	A String containing a client ID.
userID	A String containing a user ID.
password	A String containing a user password.
db	A String containing a database name.
dbSet	A String containing a database set name.
flags	A LONG that is reserved and not used in the current implementation.
Return value	A Session object.

See Also `SessionLogoff`

Unbuild

Description Deletes the CQClearQuest object you explicitly created with the Build method, when you do not need it anymore.

Syntax

Perl

```
CQClearQuest::Unbuild(ClearQuestObj);
```

Identifier	Description
CQClearQuest	A static function specifier for the CQClearQuest object.
ClearQuestObj	The CQClearQuest object that you are deleting.
Return value	None.

Example

Perl

```
use CQPerlExt;

my $cqobject = CQClearQuest::Build();

CQClearQuest::Unbuild($cqobject);
```

See Also **AdminSession Object**
Session Object

Database Object

9

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, Rational 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 examples show you how to create a database and set its initial schema revision.

Examples

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)
```

Perl

```
use CQPerlExt;
$adminSession = CQAdminSession::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(0);

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

#...
CQAdminSession::Unbuild($adminSession);
```

See Also

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

Database Object Properties

The following list summarizes the Database object properties:

Property Name	Access	Description
CheckTimeoutInterval	Read/Write	Sets or returns the interval at which to check for user timeouts.
ConnectHosts	Read/Write	Sets or returns the host name list for the physical location of a database server.
ConnectProtocols	Read/Write	Sets or returns the network protocol list for the database server.
DatabaseFeatureLevel	Read	Gets the feature level of your session database.
DatabaseName	Read/Write	Sets or returns the physical name of the database.
DBOLogin	Read/Write	Sets or returns the database owner's login name.
DBOPassword	Read/Write	Sets or returns the database owner's password.
Description	Read/Write	Sets or returns the descriptive comment associated with the database.
Name	Read/Write	Sets or returns the logical database name.
ROLogin	Read/Write	Sets or returns the login name for users who have read-only access to the database.
ROPassword	Read/Write	Sets or returns the password for users who have read-only access to the database.
RWLogin	Read/Write	Sets or returns the login name for users who have read/write access to the database.
RWPassword	Read/Write	Sets or returns the password for users who have read/write access to the database.
SchemaRev	Read-only	Returns the schema revision currently in use by the database.
Server	Read/Write	Sets or returns the name of the server on which the database resides.
SubscribedGroups	Read-only	Returns the groups that are explicitly subscribed to this database.
SubscribedUsers	Read-only	Returns the users that are explicitly subscribed to this database.
TimeoutInterval	Read/Write	Sets or returns the user timeout interval.
Vendor	Read/Write	Sets or returns the vendor type of the database.

CheckTimeoutInterval

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.

Note: Setting a new value does not take effect until the ApplyPropertyChanges method is called.

Syntax

VBScript

database.**CheckTimeoutInterval**

database.**CheckTimeoutInterval** *setValue*

Perl

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

*$database->***SetCheckTimeoutInterval(***setValue***);**

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

See Also

TimeoutInterval
ApplyPropertyChanges

ConnectHosts

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 whose Vendor property is SQL_ANYWHERE. This property corresponds to the SQL Anywhere `HOST` database server option.

Note: Setting a new value does not take effect until the ApplyPropertyChanges method is called.

Syntax

VBScript

database.**ConnectHosts**

database.**ConnectHosts** *newHosts*

Perl

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

*$database->***SetConnectHosts(***newHosts***);**

Identifier	Description
database	A Database object.
newHosts	For Visual Basic, an array of strings that sets the host name list for a physical location of a database server. For Perl, a reference to an array of strings. Each String in the array contains the host name list for a physical location of a database server.
Return value	For Visual Basic, an array of strings containing the host name list for a physical location of a database server. For Perl, a reference to an array of strings. Each String in the array contains the host name list for a physical location of a database server.

See Also

`ConnectProtocols`
`ApplyPropertyChanges`

ConnectProtocols

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

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

Note: Setting a new value does not take effect until the ApplyPropertyChanges method is called.

Syntax

VBScript

database.**ConnectProtocols**

database.**ConnectProtocols** *newProtocol*

Perl

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

*$database->***SetConnectProtocols**(*newProtocol*)**;**

Identifier	Description
database	A Database object.
newProtocol	For Visual Basic, an array of strings setting the name of a new network protocol for the database server. For Perl, reference to an array of strings. Each String in the array contains the name of a network protocol for the database server.
Return value	For Visual Basic, an array of strings containing the name of a network protocol for the database server. For Perl, 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**
ApplyPropertyChanges

DatabaseFeatureLevel

Description

Gets the feature level of your session database.

You can use **GetMinCompatibleFeatureLevel** and **GetMaxCompatibleFeatureLevel** of the Session object to get the feature level information you need for determining the valid releases for upgrades and for backward compatability.

The feature level is read-only.

Syntax

VBScript

database.**DatabaseFeatureLevel**

Perl

database->**GetDatabaseFeatureLevel**();

Identifier	Description
database	A Database object.
Return value	A Long containing the database feature level.

See Also

DatabaseFeatureLevel

When you want to find what database formats your ClearQuest client supports, you will also want to use:
GetMaxCompatibleFeatureLevel
GetMinCompatibleFeatureLevel
GetSessionFeatureLevel
ApplyPropertyChanges

DatabaseName

Description

Sets or returns the physical name of the current database.

Setting the name changes the information ClearQuest uses to connect to the physical database, not the actual database itself.

Note: Setting a new value does not take effect until the ApplyPropertyChanges method is called.

Syntax

VBScript

database.**DatabaseName**

database.**DatabaseName** *newDbName*

Perl

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

*$database->***SetDatabaseName**(new*DbName*);

Identifier	Description
database	A Database object.
newDbName	A String containing the new physical name of the database, including any associated path information (for example, "C:\temp\NewDb.mdb").
Return value	A String containing the current physical name of the database, including any associated path information.

Examples

VBScript

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


' Get the list of databases in the
' master database set.
databases = sessionObj.GetAccessibleDatabases("MASTR","admin","")
' Login to each database successively.
For Each db in databases
   dbName = db.GetDatabaseName
   sessionObj.UserLogon "admin", "", dbName, AD_PRIVATE_SESSION, ""
' Access the database
   ' ...
Next
```

Perl

```
use CQPerlExt;
#Start a ClearQuest session
$sessionObj = CQSession::Build();
#Get a list of accessible databases
```

```
$databases = $sessionObj->GetAccessibleDatabases("MASTR", "admin", "");
$count = $databases->Count();

# Login to each database successively.
for($x=0;$x<$count;$x++){
   $db = $databases->Item($x);
   $dbName = $db->GetDatabaseName();
   # Logon to the database
   $sessionObj->UserLogon( "admin", "", $dbName, "" );
   #...
   }
CQSession::Unbuild($sessionObj);
```

See Also

Name
ApplyPropertyChanges

DBOLogin

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.

Note: Setting a new value does not take effect until the ApplyPropertyChanges method is called.

Syntax

VBScript

database.**DBOLogin**

database.**DBOLogin** *dbOwnerName*

Perl

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

$database->**Set**DBOLogin(*dbOwnerName*);

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

See Also

DBOPassword
ApplyPropertyChanges

DBOPassword

Description

Sets or returns the database owner's password.

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

Note: Setting a new value does not take effect until the ApplyPropertyChanges method is called.

Syntax

VBScript

database.**DBOPassword**

database.**DBOPassword** *dbOwnerPassword*

Perl

$database->**Get**DBOPassword();
$database->**Set**DBOPassword(*dbOwnerPassword*);

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

See Also

ROLogin
DBOLogin
ApplyPropertyChanges

Description

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.

Note: Setting a new value does not take effect until the ApplyPropertyChanges method is called.

Syntax

VBScript

database.**Description**

database.**Description** *dbOwnerName*

Perl

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

Identifier	Description
database	A Database object.
dbDescription	A String containing the database's comment text.
Return value	A String containing the descriptive comment associated with the database.

See Also

Name
ApplyPropertyChanges

Name

Description

Sets or returns the logical database name.

Setting the Name changes the information ClearQuest uses to connect to the physical database, not the actual database itself.

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 five characters.

Note: Setting a new value does not take effect until the ApplyPropertyChanges method is called.

Syntax

VBScript

database.**Name**

database.**Name** *dbName*

Perl

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

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

See Also

DatabaseName
ApplyPropertyChanges

ROLogin

Description

Sets or returns the login name for users who have read-only access to the master database (schema repository).

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

Note: This setting can be different from **DBOLogin** in SQL Server 6.5 installations. For newer installations, it is the same as **DBOLogin**.

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.

Note: Setting a new value does not take effect until the ApplyPropertyChanges method is called.

Syntax

VBScript

database.**ROLogin**

database.**ROLogin** *loginName*

Perl

*$database->***Get****ROLogin();**
*$database->***Set****ROLogin(***loginName***);**

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

See Also

ROPassword
RWLogin
Vendor
"Notation Conventions for VBScript" on page 3
"Notation Conventions for Perl" on page 2
ApplyPropertyChanges

ROPassword

Description

Sets or returns the password for users who have read-only access to the master database (schema repository).

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

Note: This setting can be different from **DBOPassword** in SQL Server 6.5 installations. For newer installations, it is the same as **DBOPassword**.

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.

Note: Setting a new value does not take effect until the ApplyPropertyChanges method is called.

Syntax

VBScript

database.**ROPassword**

database.**ROPassword** *password*

Perl

*$database->***Get****ROPassword();**
*$database->***Set****ROPassword(***password***);**

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

See Also

ROLogin
RWPassword
Vendor
"Notation Conventions for Perl" on page 2
"Notation Conventions for VBScript" on page 3
ApplyPropertyChanges

RWLogin

Description

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

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

Note: This setting can be different from **DBOLogin** in SQL Server 6.5 installations. For newer installations, it is the same as **DBOLogin**.

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

Note: Setting a new value does not take effect until the ApplyPropertyChanges method is called.

Syntax

VBScript

database.**RWLogin**

database.**RWLogin** *password*

Perl

*$database->***Get****RWLogin();**
*$database->***Set****RWLogin(***password***);**

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

See Also

ROLogin
RWPassword
Vendor
ApplyPropertyChanges

RWPassword

Description

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

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

Note: This setting can be different from **DBOPassword** in SQL Server 6.5 installations. For newer installations, it is the same as **DBOPassword**.

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

Note: Setting a new value does not take effect until the ApplyPropertyChanges method is called.

Syntax

VBScript

database.**RWPassword**

database.**RWPassword** *password*

Perl

*$database->***Get****RWPassword();**
*$database->***Set****RWPassword(***password***);**

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

See Also

ROPassword
RWLogin
Vendor
ApplyPropertyChanges

SchemaRev

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.

Syntax

VBScript

database.**SchemaRev**

Perl

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

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

See Also

SetInitialSchemaRev
Upgrade
SchemaRev Object

Server

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

Note: Setting a new value does not take effect until the ApplyPropertyChanges method is called.

Syntax

VBScript

database.**Server**

database.**Server** *serverName*

Perl

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

Identifier	Description
database	A Database object.
serverName	A String containing the name of the server.
Return value	A String containing the name of the server on which the database resides.

See Also **DatabaseName**
ApplyPropertyChanges

SubscribedGroups

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.

Syntax

VBScript

database.**SubscribedGroups**

Perl

*$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

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.

Syntax

VBScript

database.**SubscribedUsers**

Perl

*$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

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.

Note: Setting a new value does not take effect until the ApplyPropertyChanges method is called.

Syntax

VBScript

database.**TimeoutInterval**

database.**TimeoutInterval** *theTimeoutInterval*

Perl

*$database->***GetTimeoutInterval();**
*$database->***SetTimeoutInterval(***theTimeoutInterval***);**

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

See Also

CheckTimeoutInterval
ApplyPropertyChanges

Vendor

Description Sets or returns the vendor type of the database.

Note: Setting a new value does not take effect until the ApplyPropertyChanges method is called.

Syntax

VBScript

database.**Vendor**

database.**Vendor** *database_vendor_constant*

Perl

*$database->***GetVendor();**
*$database->***SetVendor(***database_vendor_constant***);**

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

See Also **DatabaseVendor Constants**
Enumerated Constants
ApplyPropertyChanges

Database Object Methods

The following list summarizes the Database object methods:

Note: For all Perl Get and Set methods that map to Visual Basic Properties, see the Properties section of this object.

Method Name	Description
ApplyPropertyChanges	Updates the database's writable properties with any recent changes.
GetConnectOptions	Returns the connect options for a database.
SetConnectOptions	Sets the connect options for a database.
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.

Additional Perl Get and Set Methods that map to Visual Basic properties:

Method Name	Description
GetCheckTimeoutInterval	Returns the interval at which to check for user timeouts.
GetConnectHosts	Returns the host name list for the physical location of the database server.
GetConnectProtocols	Returns the network protocol list for the database server.
GetDatabaseFeatureLevel	Gets the feature level of your session database.
GetDatabaseName	Returns the physical name of the database.
GetDBOLogin	Returns the database owner's login name.
GetDBOPassword	Returns the database owner's password.
GetDescription	Returns the descriptive comment associated with the database.
GetName	Returns the logical database name.
GetROLogin	Returns the login name for users who have read-only access to the database.
GetROPassword	Returns the password for users who have read-only access to the database.
GetRWLogin	Returns the login name for users who have read/write access to the database.
GetRWPassword	Returns the password for users who have read/write access to the database.

Method Name	Description
GetSchemaRev	Returns the schema revision currently in use by the database.
GetServer	Returns the name of the server on which the database resides.
GetSubscribedGroups	Returns the groups explicitly subscribed to this database.
GetSubscribedUsers	Returns the users that are explicitly subscribed to this database.
GetTimeoutInterval	Returns the user timeout interval.
GetVendor	Returns the vendor type of the database.
SetCheckTimeoutInterval	Sets the interval at which to check for user timeouts.
SetConnectHosts	Sets the host name list for the physical location of the database server.
SetConnectProtocols	Sets the network protocol list for the database server.
SetDatabaseName	Sets the physical name of the database.
SetDBOLogin	Sets the database owner's login name.
SetDBOPassword	Sets the database owner's password.
SetDescription	Sets the descriptive comment associated with the database.
SetName	Sets the logical database name.
SetROLogin	Sets the login name for users who have read-only access to the database.
SetROPassword	Sets the password for users who have read-only access to the database.
SetRWLogin	Sets the login name for users who have read/write access to the database.
SetRWPassword	Sets the password for users who have read/write access to the database.
SetServer	Sets the name of the server on which the database resides.
SetTimeoutInterval	Sets the user timeout interval.
SetVendor	Sets the vendor type of the database.

ApplyPropertyChanges

Description

Updates the writable properties of the user database with any recent property 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.

Syntax

VBScript

database.**ApplyPropertyChanges** *forceEmpty*

Perl

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

Identifier	Description
database	A Database object.
forceEmpty	Reserved. Must be False. For VB, a Variant. This argument is optional. The default value is False. For Perl, a Boolean. Must be False.
Return value	Returns an empty String if the property changes are valid. Returns a String containing an error message if there are invalid value changes to the database properties.

Example

VBScript

```
set adminSession = CreateObject("ClearQuest.AdminSession)
' Create a new database
set db = adminSession.CreateDatabase("newDb")
db.Vendor = AD_SQL_ANYWHERE
db.DatabaseName = "path SQL-Anywhere db file"
db.Desciption = "This is a sample database"
db.Server = "machine name of the server"
db.SetInitialSchemaRev =  "some schema revision"
db.ApplyPropertyChanges
```

Perl

```
# Create a new database object
my($DB);
$DB = $CQAdminSession->CreateDatabase("NEWDB");

# Set some properties
$DB->SetName("NEWDB");
$DB->SetDescription("My Cool Database");
# Set all the physical characteristics...
```

```
$DB->SetVendor($CQPerlExt::CQ_SQL_SERVER);
# Store the database in SQL Server, on machine, MySQLServer
$DB->SetServer("MySQLServer");
$DB->SetDatabaseName("CQ_NEWDB");
$DB->SetDBOLogin("CQ_NEWDB_DBO");
$DB->SetDBOPassword("SECRET");
$DB->SetRWLogin("CQ_NEWDB_DBO");
$DB->SetRWPassword("SECRET");
$DB->SetROLogin("CQ_NEWDB_DBO");
$DB->SetROPassword("SECRET");
$DB->SetTimeoutInterval(0);
$DB->SetConnectOptions("");  # Not needed, for SQL Server
# Set the initial schema rev of the user database...
$DB->SetInitialSchemaRev($DesiredSchemaRev);

# Apply the property changes
$DB->ApplyPropertyChanges(0);
```

See Also

CreateDatabase of the **AdminSession Object**
SetInitialSchemaRev
Upgrade

GetConnectOptions

Description Returns the connect options for a database. This method is for retrieving Oracle connect options for the database.

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

Syntax

Perl

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

Identifier	Description
database	A Database object.
Return value	A String containing the connect options for the database.

Example

Perl

```
# Get the database
# ...
# Get the connect options for the db
$DB->SetConnectOptions("CLIENT_VER=8.0;SERVER_VER=8.1;
                        HOST=MyOracleServerHost;SID=ORCL;LOB_TYPE=LONG");
# Then get the connect options
$CO = $DB->GetConnectOptions();
print $CO, "\n";
```

See Also

SetConnectOptions
Vendor
Upgrade

SetConnectOptions

Description

Sets the connect options for a database. This method is for setting Oracle connect options for the database.

This method does not check validity of the connect options; that is done when the settings are used to connect to a database.

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

Syntax

Perl

$database->**SetConnectOptions**(new*Value*);

Identifier	Description
database	A Database object.
newValue	A String that specifies the connect options for the database.
Return value	None.

Example

Perl

```
# Get the database
# ...
# Set the connect options for the db
$DB->SetConnectOptions("CLIENT_VER=8.0;SERVER_VER=8.1;
                        HOST=MyOracleServerHost;SID=ORCL;LOB_TYPE=LONG");
# Then get the connect options
$CO = $DB->GetConnectOptions();
print $CO, "\n";
```

See Also

GetConnectOptions
Vendor
Upgrade

SetInitialSchemaRev

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.

Syntax

VBScript

```
database.SetInitialSchemaRev schemaRev
```

Perl

```
$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
SchemaRev
ApplyPropertyChanges

Upgrade

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.

Syntax

VBScript

```
database.Upgrade schemaRev
```

Perl

```
$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
UpgradeMasterUserInfo
SchemaRev

UpgradeMasterUserInfo

Description

Upgrade this database's user information. This method copies the changes from the schema repository to the user database.

You should call this function to upgrade the appropriate databases after making changes to the users and groups of the master database.

This method is used to update a user database's "copy" of user-related information from the master database. Attempts to call this method using a schema repository (master database) results in an error stating that the user information cannot be loaded into the database MASTR because that is the master database, not a user database.

Syntax

VBScript

database.**UpgradeMasterUserInfo**

Perl

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

Identifier	Description
database	A Database object.
Return value	None.

See Also

Upgrade

DatabaseDesc Object

10

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 log on to the database. However, suppose that you want to have a logon dialog box 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`** (ODBC experts can use this string to log on to the database) method.

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` of the Session object
Session Object

DatabaseDesc Object Methods

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

GetDatabaseConnectString

Description

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 Rational ClearQuest database.

Note: You must log in with superuser privilege or an error will be generated by GetDatabaseConnectString

Syntax

VBScript

dbDesc.**GetDatabaseConnectString**

Perl

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

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

Examples

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.
databases = sessionObj.GetAccessibleDatabases("MASTR","","")
For Each db in databases
   dbName = db.GetDatabaseName
   sessionObj.UserLogon "admin", "", dbName, AD_PRIVATE_SESSION, ""
   dbConnectString = db.GetDatabaseConnectString
Next
```

Perl

```
use CQPerlExt;
#Start a ClearQuest session
$sessionObj = CQSession::Build();

#Get a list of accessible databases
$databases = $sessionObj->GetAccessibleDatabases("MASTR", "", "");
$count = $databases->Count();
#Foreach accessible database, login as joe with password gh36ak3
#joe must be a superuser
for($x=0;$x<$count;$x++){
```

```
   $db = $databases->Item($x);
   $dbName = $db->GetDatabaseName();
   # Logon to the database
   $sessionObj->UserLogon( "joe", "gh36ak3", $dbName, "" );
#Get a "direct connect" string that ODBC experts
#can use to logon to the database
   $dbConnectString = $db->GetDatabaseConnectString();
}
CQSession::Unbuild($sessionObj);
```

See Also

`UserLogon` of the Session object

Session Object

"Getting Session and Database Information" on page 850.

GetDatabaseName

Description

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.

Syntax

VBScript

dbDesc.**GetDatabaseName**

Perl

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

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

Examples

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.
databases = sessionObj.GetAccessibleDatabases("MASTR","","")
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
```

Perl

```
use CQPerlExt;

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


#Get a list of accessible databases

$databases = $sessionObj->GetAccessibleDatabases("MASTR", "", "");

#Get the number of databases

$count = $databases->Count();

# Foreach accessible database, get the dbName and

# login as joe with password gh36ak3
```

```
for($x=0;$x<$count;$x++){
   $db = $databases->Item($x);
   $dbName = $db->GetDatabaseName();
      # Logon to the database
   $sessionObj->UserLogon( "joe", "gh36ak3", $dbName, "" );
   #...
   }
CQSession::Unbuild($sessionObj);
```

See Also

`GetDatabaseSetName`
`GetAccessibleDatabases` of the Session object
Session Object
"Getting Session and Database Information" on page 850
"Notation Conventions for VBScript" on page 3
"Notation Conventions for Perl" on page 2

GetDatabaseSetName

Description

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.

Syntax

VBScript

dbDesc.**GetDatabaseSetName**

Perl

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

Identifier	Description
dbDesc	A DatabaseDesc object containing information about one of the installed databases.
Return value	A String containing the name of the database set.

Examples

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.
databases = sessionObj.GetAccessibleDatabases("MASTR","","")
For Each db in databases
   If Not db.GetIsMaster Then
      dbSetName = db.GetDatabaseSetName
      dbName = db.GetDatabaseName
      ' Logon to the database
      sessionObj.UserLogon "tom", "gh36ak3", dbName,
          AD_PRIVATE_SESSION, dbSetName
   End If
   ' ...
Next
```

Perl

```
use CQPerlExt;

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

#Get a list of accessible database description objects

$databases = $sessionObj->GetAccessibleDatabases("MASTR", "", "");

#Get the number of databases

$count = $databases->Count();
```

```
#Foreach accessible database that is not the master database, login as
#user "tom" with password "gh36ak3"
for($x=0;$x<$count;$x++){
   $db = $databases->Item($x);
   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, $dbSetName );
   }
#...
}
CQSession::Unbuild($sessionObj);
```

See Also

`GetDatabaseName`
`GetAccessibleDatabases` of the **Session Object**
"Getting Session and Database Information" on page 850
"Notation Conventions for VBScript" on page 3
"Notation Conventions for Perl" on page 2

GetDescription

Description

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.

Syntax

VBScript

dbDesc.**GetDescription**

Perl

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

Identifier	Description
dbDesc	A DatabaseDesc object containing information about one of the installed databases.
Return value	A String containing descriptive comments about the database.

Examples

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.
databases = sessionObj.GetAccessibleDatabases("MASTR","","")
For Each db in databases
   dbDescription = db.GetDescription
   ' ...
Next
```

Perl

```
use CQPerlExt;

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

#Get a list of accessible database description objects

$databases = $sessionObj->GetAccessibleDatabases("MASTR", "", "");

#Get the number of databases

$count = $databases->Count();

#For each accessible database, get the description

#of the contents of the database

for($x=0;$x<$count;$x++){

   $db = $databases->Item($x);

   $dbDescription = $db->GetDescription();

#...
```

```
}
CQSession::Unbuild($sessionObj);
```

See Also **`Description`** of the Database object
Database Object

GetIsMaster

Description

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.

Syntax

VBScript

dbDesc.**GetIsMaster**

Perl

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

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

Examples

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.
databases = sessionObj.GetAccessibleDatabases("MASTR","","")
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
```

Perl

```
use CQPerlExt;

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

#Get a list of accessible database description objects

$databases = $sessionObj->GetAccessibleDatabases("MASTR", "", "");

#Get the number of databases

$count = $databases->Count();

#Foreach accessible database that is the master database

for($x=0;$x<$count;$x++){

   $db = $databases->Item($x);
```

```
   if ( $db->GetIsMaster() ) {
      #Create an AdminSession and logon to the schema repository
      #...
   }
   else {
      #Logon to the database using the regular Session object
      #...
   }
}
CQSession::Unbuild($sessionObj);
```

See Also **Logon** of the AdminSession object

GetLogin

Description Returns the database login associated with the current user.

Syntax

VBScript

loginvalue dbDesc.**GetLogin**

Perl

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

Identifier	Description
dbDesc	A DatabaseDesc object containing information about one of the installed databases.
Return value	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 stand-alone 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.

Examples

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.
databases = sessionObj.GetAccessibleDatabases("MASTR", "", "")
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
```

Perl

```
use CQPerlExt;

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

#Get a list of accessible database description objects
$databases = $sessionObj->GetAccessibleDatabases("MASTR", "", "");
#Get the number of databases
$count = $databases->Count();
#Foreach accessible database that is not the master database
for($x=0;$x<$count;$x++){
   $db = $databases->Item($x);
   if ( ! $db->GetIsMaster() ) {
      $dbName = $db->GetDatabaseName();
      #Logon to the database as "tom" with password "gh36ak3"
      $sessionObj->UserLogon( "tom", "gh36ak3", $dbName, $dbSetName );
      #Get the database login and password for "tom"
      $dbLogin = $db->GetLogin();
      $dbPassword = $db->GetPassword();
      #...
   }
}
CQSession::Unbuild($sessionObj);
```

See Also

GetLogin
DBOLogin of the Database object
ROLogin of the Database object
RWLogin of the Database object
UserLogon of the Session object
Database Object
Session Object
"Notation Conventions for VBScript" on page 3
"Notation Conventions for Perl" on page 2

GetPassword

Description

Returns the database password (of the **`Vendor`** database) associated with the current user.

The database password is not the same as the user's ClearQuest password. See **`GetLogin`** for more information.

Note: You must have superuser privileges for GetPassword to return the database login password, or an exception is thrown.

Syntax

VBScript

`dbDesc``.`**GetPassword**

Perl

`$dbDesc->``GetPassword()`*`;`*

Identifier	Description
dbDesc	A DatabaseDesc object containing information about one of the installed databases.
Return value	A String containing the database password associated with the current user.

See Also

`GetLogin`
`DBOPassword` of the Database object
`ROPassword` of the Database object
`RWPassword` of the Database object
`UserLogon` of the Session object
Database Object
Session Object
"Notation Conventions for VBScript" on page 3
"Notation Conventions for Perl" on page 2

DatabaseDescs Object

11

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`** method. Use the **`Item`** method to retrieve items from the collection.

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

See Also **DatabaseDesc Object**

DatabaseDescs Object Methods

The following list summarizes the DatabaseDescs object methods:

Method Name	Description
`Add`	Adds an Attachment object to the collection.
`Count`	Returns the number of items in the collection.
`Item`	Returns the specified item in the collection.
`ItemByName`	Returns the specified item in the collection.

Add

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.

Syntax

Perl

*`$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`
`Item`

Count

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

Syntax

Perl

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

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

See Also `Item`
`Add`

Item

Description

Returns the specified item in the DatabaseDescs collection.

The argument to this method is a numeric index (itemNum).

Syntax

Perl

```
$dbDescObj = $databasedescs->Item(itemNum);
```

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.
Return value	The DatabaseDesc object at the specified location in the collection.

See Also

Count
ItemByName
Add

ItemByName

Description

Returns the specified item in the DatabaseDescs collection.

The argument to this method is a String (name).

Syntax

Perl

```
$dbDescObj = $databasedescs->ItemByName(name);
```

Identifier	Description
databasedescs	A DatabaseDescs collection object.
name	A String that serves as a key into the collection. This string corresponds to the **GetDatabaseName** of the desired DatabaseDescs.
Return value	The DatabaseDesc object at the specified location in the collection.

See Also

Count
Item
Add

Databases Object

12

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`** method. Use the **`Item`** method to retrieve items from the collection.

See Also **Database Object**

Databases Object Properties

The following list summarizes the Databases object properties:

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

Count

Description

Returns the number of items in the collection.

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

Syntax

VBScript

db_collection.**Count**

Perl

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

Identifier	Description
db_collection	A Databases collection object, representing the set of databases 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`

Databases Object Methods

The following list summarizes the Databases object methods:

Method name	Description
`Item`	Returns the item at the specified index in the collection.
`ItemByName`	(Perl only) Returns the specified item in the collection.

Note: For Perl methods that map to VB Properties, see the Properties section of this object.

The following list summarizes additional Perl Databases object methods:

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

Item

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

VBScript

```
db_collection.Item(itemNum)
db_collection.Item (name)
```

Perl

```
$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.
Return value	The Database object at the specified location in the collection.

See Also

`Count`

Entity Object

13

An Entity object represents a record in the database.

Entity objects are some of the most important objects in Rational 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 **`Entity 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 **`SetFieldRequirednessForCurrentAction`** method of Entity to set the field's behavior.

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

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

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` and `GetFieldsUpdatedThisGroup` methods.

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** 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** (to build a new record) or **GetEntity** (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" on page 15 or the methods of the current **Session Object**.

Task-Oriented Entity Methods

- Static information (metadata) about the entity:
  GetEntityDefName
  GetType

- Information about the fields associated with the entity:
  GetFieldNames
  GetFieldType
  GetFieldRequiredness
  GetFieldChoiceList
  SetFieldChoiceList
  InvalidateFieldChoiceList
  GetFieldChoiceType
  FireNamedHook
- Information about the entity's duplicates:
  IsDuplicate
  GetOriginal
  GetOriginalID
  HasDuplicates
  IsOriginal
  GetDuplicates
  GetAllDuplicates
- Special handling for Attachments:
  GetAttachmentDisplayNameHeader
  AddAttachmentFieldValue
  DeleteAttachmentFieldValue
  EditAttachmentFieldDescription
  LoadAttachment
- Changing an Entity:
  IsEditable
  Validate
  Commit
  Revert

  After BuildEntity or EditEntity has run, the Entity may be modified. The changes must be successfully validated before they may be committed -- a commit actually updates the database. They may also be cancelled, using Revert. After committed or reverted, the entity is no longer editable.
- Changing an entity's field values:
  SetFieldValue
  AddFieldValue
  DeleteFieldValue
- Fetching an entity's field values:
  GetFieldValue
  GetFieldOriginalValue
  GetAllFieldValues
  GetInvalidFieldValues
  GetFieldsUpdatedThisSetValue
  GetFieldsUpdatedThisGroup
  BeginNewFieldUpdateGroup
  GetFieldsUpdatedThisAction

See Also "Working with Records" on page 18
`BuildEntity` of the **Session Object**
`EditEntity` of the **Session Object**
`GetEntity` of the **Session Object**
`GetEntityByDbId` 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`	Read-only	Returns the AttachmentFields collection object containing this Entity object's attachment fields.
`HistoryFields`	Read-only	Returns the HistoryFields collection object containing this Entity object's history fields.

AttachmentFields

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, after 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 **`Attachments and Histories`**.

See also "Getting and Setting Attachment Information" on page 810.

Syntax

VBScript

`entity``.`**AttachmentFields**

Perl

`$`*`entity`*`->Get`**AttachmentFields();**

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.

Examples

VBScript

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

   ' ...
Next
```

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

Attachment Object
AttachmentField Object
AttachmentFields Object
Attachments Object
“Getting and Setting Attachment Information” on page 810

HistoryFields

Description

Returns the HistoryFields collection 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**.

Syntax

VBScript

entity.**HistoryFields**

Perl

$*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.

Examples

VBScript

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

   ' ...
Next
```

Perl

```
# Get the list of history fields
$historyfields = $entity->GetHistoryFields();


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


for ($x = 0; $x < $numfields ; $x++)
 {
 # Get each history field
 $onefield = $historyfields->Item($x);


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

See Also

Histories Object
History Object
HistoryField Object
HistoryFields Object

Entity Object Methods

The following list summarizes the Entity object methods:

Note: For all Perl Get and Set methods that map to Visual Basic Properties, see the Properties section of this object.

Method Name	Description
`AddAttachmentFieldValue`	Inserts the file and description into the list of attachments.
`AddFieldValue`	Adds the specified value to the list of values in the named field.
`BeginNewFieldUpdateGroup`	Marks the beginning of a series of SetFieldValue calls.
`Commit`	Updates the database with the changes made to the Entity object.
`DeleteAttachmentFieldValue`	Deletes the attachment.
`DeleteFieldValue`	Removes the specified value from the field's list of values.
`EditAttachmentFieldDescription`	Updates the description field of the given attachment.
`EditEntity`	Performs the specified action on a record and makes the record available for editing.
`FireNamedHook`	Executes a named hook (record script) of this record's **EntityDef Object**.
`GetActionName`	Returns the name of the action associated with the current Entity object.
`GetActionType`	Returns the type of the action associated with the current Entity object.
`GetAllDuplicates`	Returns links to all of the duplicates of this Entity, including duplicates of duplicates.
`GetAllFieldValues`	Returns an array of FieldInfo objects corresponding to all of the Entity object's fields.
`GetAttachmentDisplayNameHeader`	Returns the attachment display names, given the attachment fieldname.
`GetDbId`	Returns the Entity object's database ID number.
`GetDefaultActionName`	Returns the default action associated with the current state.
`GetDisplayName`	Returns the unique key associated with the Entity.

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

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

Additional Perl Get and Set Methods that map to Visual Basic properties:

Method Name	Description
GetAttachmentFields	Returns the AttachmentFields collection object containing this Entity object's attachment fields.
GetHistoryFields	Returns the HistoryFields collection object containing this Entity object's history fields.

AddAttachmentFieldValue

Description Inserts the file and description into the list of attachments. Adds a new attachment to an entity to the specified attachment field, with the given filename and description.

Syntax

VBScript

entity.`AddAttachmentFieldValue` *attachment_fieldname, filename, description*

Perl

$entity->`AddAttachmentFieldValue`*(attachment_fieldname, filename, description);*

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).
attachment_fieldname	A String containing the attachment field name you are adding this attachment to.
filename	A String containing the file name of the attachment.
description	A String containing a description of the attachment.
Return value	Returns a String. The String returned is empty if the update is permitted, or an explanation of the error.

See Also `AttachmentFields`
`AttachmentField Object`

AddFieldValue

Description

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

This method is similar to **`SetFieldValue`**, 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.

Note: The AddFieldValue method is designed to work with list fields. It is not designed to work with scalar fields (such as string, multiline string, and reference).

To determine whether a field contains a valid value, obtain the **FieldInfo Object** for that field and call **`ValidityChangedThisSetValue`** 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 **`EditEntity`** of the Session object.

Syntax

VBScript

`entity`.**AddFieldValue** *`field_name, new_value`*

Perl

`$`*`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 Visual Basic, 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.

Examples

VBScript

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

Perl

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

See Also

`DeleteFieldValue`
`GetFieldValue`
`SetFieldValue`

`ValidityChangedThisSetValue` in the **FieldInfo Object**
`ValueChangedThisSetValue` in the **FieldInfo Object**
`EditEntity` of the **Session Object**

BeginNewFieldUpdateGroup

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`** You can later call **`GetFieldsUpdatedThisGroup`** 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.

Syntax

VBScript

*`entity.`***BeginNewFieldUpdateGroup**

Perl

`$`*`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

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`
`GetFieldsUpdatedThisGroup`
`GetFieldsUpdatedThisSetValue`
`SetFieldValue`
`ValidityChangedThisSetValue` of the **FieldInfo Object**

Commit

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.

Syntax

VBScript

*entity.***Commit**

Perl

*$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	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.

Examples

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

status = entityObj.Validate
     if status = "" then
        status = entityObj.Commit
        if status = "" then
           ' successful commit
        else
           'check error message
        end if
     else
        entityObj.Revert
     end if
' The Entity object is no longer editable
```

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

$status = $entityObj->Validate();
   if ($status == ""){
      $status = $entityObj->Commit();
      if ($status == ""){
         # successful commit
      }
      else {
         # check error message
         }
   else {
         $entityObj->Revert();
      }
# The entity object is no longer editable.
```

See Also

`IsEditable`
`Revert`
`Validate`
`BuildEntity` of the **Session Object**
`EditEntity` of the **Session Object**
"Updating Duplicate Records to Match the Parent Record" on page 820
"Getting a List of Defects and Modifying a Record" on page 847

DeleteAttachmentFieldValue

Description Deletes the attachment. Deletes an attachment from an entity from the given attachment field using the given display name. The display name is the attachment filename.

Syntax

VBScript

entity.`DeleteAttachmentFieldValue` *attachment_fieldname, element_displayname*

Perl

$entity->`DeleteAttachmentFieldValue`*(attachment_fieldname, element_displayname);*

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.
attachment_fieldname	A String containing the attachment field name you are adding this attachment to.
element_displayname	A String containing the attachment file name.
Return value	Returns a String. The String returned is empty if the update is permitted, or an explanation of the error.

See Also `AttachmentFields`
`AttachmentField Object`

DeleteFieldValue

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.

Syntax

VBScript

entity.`DeleteFieldValue` *field_name, old_value*

Perl

`$`*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.

Examples

VBScript

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

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`
`GetFieldValue`
`SetFieldValue`

`ValidityChangedThisSetValue` in the **FieldInfo Object**
`ValueChangedThisSetValue` in the **FieldInfo Object**
`EditEntity` of the **Session Object**

EditAttachmentFieldDescription

Description Updates the description field of the given attachment. Changes the attachment's description to the new value.

Syntax

VBScript

```
entity.EditAttachmentFieldDescription attachment_fieldname,
element_displayname, new_description
```

Perl

```
$entity->EditAttachmentFieldDescription( attachment_fieldname,
element_displayname, new_description);
```

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.
attachment_fieldname	A String containing the attachment field name for which you are editing the field description.
element_displayname	A String containing the attachment file name.
new_description	A String containing the new description of the attachment.
Return value	Returns a String. The String returned is empty if the update is permitted, or an explanation of the error.

See Also **`Description`** of the **`Attachment Object`**
`AttachmentFields`
`AttachmentField Object`

EditEntity

Description

Performs the specified action on a record and makes the record 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 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 **`Validate`** and **`Commit`** methods, respectively.

Syntax

VBScript

`entity`.**EditEntity** *`edit_action_name`*

Perl

$*`entity`***->EditEntity(***`edit_action_name`***);**

Identifier	Description
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.

See Also

`EditEntity` of the **`Session Object`**
`BuildEntity` of the **`Session Object`**
`GetEntity` of the **`Session Object`**
`GetEntityByDbId` of the **`Session Object`**
`Validate`
`Commit`
`GetActionDefNames`
"ActionType Constants" on page 783
"Updating Duplicate Records to Match the Parent Record" on page 820
"Managing Records (Entities) that are Stateless and Stateful" on page 822

FireNamedHook

Description

Executes a named hook (record script) of this record's **EntityDef Object**.

You can use this method to execute a record script at runtime. Record scripts are routines you define and are specific to a particular record type. You can use record scripts in conjunction with form controls or you can call them from other hooks. You define record hooks using ClearQuest Designer. The syntax for record scripts is as follows:

```
Function EntityDefName_RecordScriptName(param)
   ' param as Variant
   ' EntityDefName_RecordScriptName as Variant

   ' Hook program body
End Function
```

You cannot use this method to execute a field or action hook of a record. Neither can you execute a global hook, except indirectly from the record script.

You can call this method on an Entity object regardless of whether or not it is editable. However, if your script 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 script accepts any parameters, put all of the parameters in a single Variant (for Visual Basic) and specify that Variant in `param`. The script must be able to interpret the parameters passed into it. Upon return, the script can similarly return a Variant with any appropriate return values.

For Perl, you can include multiple parameters by concatenating simple string values with a non-printing character as a separator (such as newline). This String can then be decoded with the built-in `split` operator.

Syntax

VBScript

*`entity.`***FireNamedHook** *`scriptName, param`*

Perl

`$`*`entity->`***FireNamedHook**`(`*`scriptName, param`*`);`

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).
scriptName	A String containing the name of the hook to execute.
param	For Visual Basic, a Variant containing the parameters you want to pass to the hook. For Perl, a 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 executes successfully, this method returns an empty string (""), otherwise the returned string contains a description of the error.

Examples

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)
```

Perl

```
# Execute the hook "MyHook" with the specified parameters
$params = "option 1\noption 2";
$returnValue = $entity->FireNamedHook("MyHook",$params);

# In the hook, split them like this:
my $param = shift;
my @params = split '\n', $param;
```

See Also

`EditEntity` of the **Session Object**
`GetHookDefNames` of the **EntityDef Object**
Understanding Record Scripts on page 321

GetActionName

Description Returns the name of the current action associated with the current entity. Used in base action hooks.

Syntax

VBScript

entity.**GetActionName**

Perl

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

Identifier	Description
entity	An Entity object corresponding to a record in a schema.
Return value	A String containing the name of the current action associated with the current entity.

Examples

VBScript

```
actionName = entityObj.GetActionName
```

Perl

```
$actionName = $entityObj->GetActionName();
```

See Also

See "Showing Changes to a FieldInfo (Field)" on page 836.
See "Triggering a Task with the Destination State" on page 856.
GetActionType
ActionType Constants

GetActionType

Description Returns the type of the current action associated with the current entity. Typically used in base action hooks.

Syntax

VBScript

entity.**GetActionType**

Perl

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

Identifier	Description
entity	An Entity object corresponding to a record in a schema.
Return value	A String containing the type of the current action associated with the current entity.

Examples

VBScript

```
actionType = entity.GetActionType
```

Perl

```
$actionType = $entity->GetActionType();
```

See Also **GetActionName**
ActionType Constants

GetAllDuplicates

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.

Syntax

VBScript

`entity``.`**GetAllDuplicates**

Perl

`$`*`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 Visual Basic, 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.

Examples

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 three **Link Object**s:

- A link between entity1 and entity2
- A link between entity1 and entity3
- A link between entity3 and entity4

Perl

```
$linkobjs = $entity1->GetAllDuplicates();

# Find out how many duplicates there
# are so the for loop can iterate them
$numLinks = $linkobjs->Count();

for ($x = 0; $x < $numLinks ; $x++)
   {
   $linkobj = $linkobjs->Item($x);
   $childentity = $linkobj->GetChildEntity();
```

```
}
```

See Also

`GetDuplicates`
`GetOriginal`
`GetOriginalID`
`HasDuplicates`
`IsDuplicate`
`IsOriginal`
`MarkEntityAsDuplicate` of the **Session Object**
`UnmarkEntityAsDuplicate` of the **Session Object**
Link Object
"Updating Duplicate Records to Match the Parent Record" on page 820

GetAllFieldValues

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.

Syntax

VBScript

*entity.***GetAllFieldValues**

Perl

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

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 Visual Basic, 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.

Examples

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
```

Perl

```
# Get the list of field values
$fieldvalues = $entity->GetAllFieldValues();


$numfields = $fieldvalues->Count();


for ($x = 0; $x < $numfields ; $x++)
   {
   $field = $fieldvalues->Item($x);
   $fieldvalue = $field->GetValue();
   $fieldname = $field->GetName();
   # ... other field commands
   }
```

See Also **"Getting and Setting Attachment Information" on page 810**
`GetFieldValue`
`GetInvalidFieldValues`
FieldInfo Object

GetAttachmentDisplayNameHeader

Description Returns the column headers for the subfields of an attachment's display name. Returns the attachment display names, given the attachment fieldname. Each attachment field contains one or more attachments.

Syntax

VBScript

entity.`GetAttachmentDisplayNameHeader` *attachment_fieldname*

Perl

$entity->`GetAttachmentDisplayNameHeader`*(attachment_fieldname);*

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.
attachment_fieldname	A String containing the attachment field name.
Return value	For Visual Basic, a Variant 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 an array of strings.

See Also `DisplayName` of the `Attachment Object`

GetDbId

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 **`GetDisplayName`**.

Syntax

VBScript

entity.**GetDbId**

Perl

$*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.

Examples

VBScript

```
set session = GetSession
set record1 = session.GetEntity("defect", "test00000001")
dbid = record1.Getdbid
set record1 = session.GetEntityByDbId("defect", dbid)
```

Perl

```
#Assume you have $entityObj, an Entity Object
$sessionObj = $entityObj->GetSession();
$dbid = $entityObj->GetDbId();
#...
#Later, to get the record again:
$entityObj = $sessionObj->GetEntityByDbId("defect",$dbid);
```

See Also

`GetDisplayName`

"Extracting Data About an EntityDef (Record Type)" on page 828

GetDefaultActionName

Description

Returns the default action name associated with the current state.

This method allows you to get the default action for the specified record.

Whereas this method returns the default action name associated with the current state, **`GetActionDestStateName`** returns the destination state name associated with the current action.

Syntax

VBScript

*`entity.`***GetDefaultActionName**

Perl

*`$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.

Examples

VBScript

```
DefaultActionName = entity.GetDefaultActionName
```

Perl

```
$defaultactionname = $entity->GetDefaultActionName();
```

See Also

`GetActionDestStateName`

GetDisplayName

Description

Returns the display name (a 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.

Syntax

VBScript

*`entity.`***GetDisplayName**

Perl

*`$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.

Examples

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
```

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`
`GetFieldValue`
`GetValue` of the FieldInfo Object
"Updating Duplicate Records to Match the Parent Record" on page 820

GetDuplicates

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.

Syntax

VBScript

entity.**GetDuplicates**

Perl

$*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 Visual Basic, 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.

Examples

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

Perl

```
$dups = $entity->GetDuplicates();

# Find out how many duplicates there
# are so the for loop can iterate them
$numdups = $dups->Count();

for ($x = 0; $x < $numdups ; $x++)
   {
   $dupvar = $dups->Item($x);
   $childentity = $dupvar->GetChildEntity();
```

```
}
```

See Also

`GetAllDuplicates`
`GetOriginal`
`GetOriginalID`
`HasDuplicates`
`IsDuplicate`
`IsOriginal`
`MarkEntityAsDuplicate` of the **Session Object**
`UnmarkEntityAsDuplicate` of the **Session Object**
"Updating Duplicate Records to Match the Parent Record" on page 820
Link Object

GetEntityDefName

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.

Syntax

VBScript

*entity.***GetEntityDefName**

Perl

*$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.

Examples

VBScript

```
set sessionObj = GetSession

' Get the EntityDef of the record using GetEntityDefName
entityDefName = GetEntityDefName
set entityDefObj = sessionObj.GetEntityDef(entityDefName)
```

Perl

```
$sessionobj = $entity->GetSession();


# Get the EntityDef of the record using GetEntityDefName


$entitydefname = $entity->GetEntityDefName();
$entitydefobj = $sessionobj->GetEntityDef($entitydefname);
```

See Also

`GetEntityDef`
EntityDef Object

GetFieldChoiceList

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** (Visual Basic only). You can use that object to retrieve the values.

If you have a choice list hook, which generates the set of choices for a field, it must return its results by filling in a collection that is passed into the hook procedure.

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 the entity is not editable.

Syntax

VBScript

*`entity.`***GetFieldChoiceList** *`field_name`*

Perl

*`$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 Visual Basic, 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 an array of strings is returned.

Examples

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
```

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]);
 }
#Example 2:
sub Dyn_choice_get_values
{
   my $session;
   my $fieldchoicelist;
   $session = $entity->GetSession();
   $fieldchoicelist = $entity->GetFieldChoiceList("Dyn_List_Example");
   $session->OutputDebugString(" CHOICELIST @$fieldchoicelist \n");
   return 0;
}
```

See Also

GetFieldChoiceType
GetFieldNames
HookChoices Object
“Creating a Dependent Choice List” on page 854
ChoiceType Constants

GetFieldChoiceType

Description

Returns the choice list type for the given field.

The return value is a ChoiceType constant, 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.

Syntax

VBScript

*entity.*__GetFieldChoiceType__ *field_name*

Perl

*$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. This value is one of the **ChoiceType Constants**.

Examples

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
```

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`
`GetFieldNames`
`ChoiceType Constants`
HookChoices Object
"Notation Conventions for Perl" on page 2
"Notation Conventions for VBScript" on page 3

GetFieldMaxLength

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.

Syntax

VBScript

*entity.***GetFieldMaxLength** *field_name*

Perl

$*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.

Examples

VBScript

```
' Check the maximum length of a string field.
fieldType = GetFieldType("field1")
If fieldType = AD_SHORT_STRING Then
   maxLength = GetFieldMaxLength("field1")
End If
```

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
EventType Constants
"Notation Conventions for VBScript" on page 3
"Notation Conventions for Perl" on page 2

GetFieldNames

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.

Syntax

VBScript

entity.**GetFieldNames**

Perl

$*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 Visual Basic, a Variant containing an Array whose elements are strings is returned. Each String contains the name of one field. For Perl, a reference to an array of strings.

Examples

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
```

Perl

```
# get session object
$sessionobj = $entity->GetSession();


# get a reference to an array of strings
$fieldNameList = $entity->GetFieldNames();


foreach $fieldname (@$fieldNameList)
   {
    $fieldinfoobj = $entity->GetFieldValue($fieldname);
    $fieldtype = $fieldinfoobj->GetType();
```

```
    $fieldvalue = $fieldinfoobj->GetValue();

    $sessionobj->OutputDebugString(
        "Field name: ".$fieldname.", type=".$fieldtype.",
        value=".$fieldvalue);
}
```

See Also

GetFieldChoiceList
GetFieldDefNames
GetFieldRequiredness
GetFieldType
GetFieldValue
“Getting and Setting Attachment Information” on page 810
“Showing Changes to an Entity (Record)” on page 838

GetFieldOriginalValue

Description

Returns a FieldInfo object 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.

Syntax

VBScript

entity.**GetFieldOriginalValue** (*field_name*)

Perl

$*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

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
```

Perl

```
my($FieldNamesRef) = $entity->GetFieldNames();
   foreach $FN (@$FieldNamesRef) {
      # Get the field's original value...
      $FieldInfo = $entity->GetFieldOriginalValue($FN);
      #...
   }
```

See Also `GetFieldValue`
FieldInfo Object
"Showing Changes to a FieldInfo (Field)" on page 836
"Showing Changes to an Entity (Record)" on page 838

GetFieldRequiredness

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.

Syntax

VBScript

`entity`.**GetFieldRequiredness** *`field_name`*

Perl

`$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`**.

Examples

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
```

Perl

```
# Change all mandatory fields to optional
$fieldnamelist = $entity->GetFieldNames();


foreach $fieldname (@$fieldnamelist)
 {
```

```
$fieldreq = $entity->GetFieldRequiredness($fieldname);
if ($fieldreq eq $CQPerlExt::CQ_MANDATORY)
 {
    $entity->SetFieldRequirednessForCurrentAction($fieldname,
        $CQPerlExt::CQ_OPTIONAL);
 }
}
```

See Also

`GetFieldNames`
`GetRequiredness` of the **FieldInfo Object**
"Notation Conventions for Perl" on page 2
"Notation Conventions for VBScript" on page 3

GetFieldsUpdatedThisAction

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.

Syntax

VBScript

entity.**GetFieldsUpdatedThisAction**

Perl

$*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 Visual Basic, 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.

Examples

VBScript

```
set sessionObj = GetSession

' Report any fields that changed during the recent action
fieldList = GetFieldsUpdatedThisAction
For Each field in fieldList
     ' Report the fields to the user
     sessionObj.OutputDebugString "Field " & field.GetName & "
        changed."
Next
```

Perl

```
$sessionobj = $entity->GetSession();

# Report any fields that changed during the recent action

$fieldlist = $entity->GetFieldsUpdatedThisAction();
```

```
# Find out how many duplicates there
# are so the for loop can iterate them
$updatedfields = $fieldlist->Count();
for ($x = 0; $x < $updatedfields ; $x++)
{
 # Report the fields to the user
 $sessionobj->OutputDebugString("Field ".$fieldlist->Item($x)->GetName."
      changed." )
}
```

See Also **"Showing Changes to a FieldInfo (Field)" on page 836**
`BeginNewFieldUpdateGroup`
`GetFieldsUpdatedThisAction`
`GetFieldsUpdatedThisSetValue`
`SetFieldValue`
`ValidityChangedThisSetValue` of the **FieldInfo Object**

GetFieldsUpdatedThisGroup

Description

Returns a **FieldInfo Object** for each field that was modified since the most recent call to **`BeginNewFieldUpdateGroup`**.

Use this method to mark the end of a group of calls to **`SetFieldValue`** (You must have previously called **`BeginNewFieldUpdateGroup`** 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.

Syntax

VBScript

`entity``.`**GetFieldsUpdatedThisGroup**

Perl

`$`*`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 Visual Basic, 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.

Examples

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
```

Perl

```
$entity->BeginNewFieldUpdateGroup()
$entity->SetFieldValue("field1", "1" );
$entity->SetFieldValue("field2", "submitted");
$entity->SetFieldValue("field3", "done");
$updatedFields = $entity->GetFieldsUpdatedThisGroup ();
$count = $updatedFields->Count();
# Iterate over all the fields that changed
for ($x = 0; $x < $count ; $x++)
```

```
  {
  $field = $updatedFields->Item($x);
  # do other tasks...
 }
```

See Also **BeginNewFieldUpdateGroup**
GetFieldsUpdatedThisAction
GetFieldsUpdatedThisSetValue
SetFieldValue
ValidityChangedThisSetValue of the **FieldInfo Object**

GetFieldsUpdatedThisSetValue

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`**. 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.

Syntax

VBScript

entity.**GetFieldsUpdatedThisSetValue**

Perl

$*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 Visual Basic, 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.

Examples

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
```

Perl

```
$entity->SetFieldValue("field1", "100");


$modifiedfields = $entity->GetFieldsUpdatedThisSetValue();
$numfields = $modifiedfields->Count();


if ($numfields > 1)
 {
 $session->OutputDebugString("Changing field1 resulted in changes
    to ".$numfields." other fields");
 }
```

See Also

`BeginNewFieldUpdateGroup`
`GetFieldsUpdatedThisAction`
`GetFieldsUpdatedThisGroup`
`SetFieldValue`
`ValidityChangedThisSetValue` in the **FieldInfo Object**
FieldInfo Object

GetFieldType

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.

Syntax

VBScript

*`entity.`***GetFieldType** *`field_name`*

Perl

`$`*`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`**.

Examples

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
```

Perl

```
$sessionobj = $entity->GetSession();


# Iterate through the fields and output
# the field name and type.
```

```
$fieldnamelist = $entity->GetFieldNames();


foreach $fieldname (@$fieldnamelist)
{
 $fieldtype = $entity->GetFieldType($fieldname);
 $sessionobj->OutputDebugString("Field name: ".$fieldname. ",
         type=".$fieldtype);
 }
```

See Also **`GetFieldNames`**
`GetType` of the **FieldInfo Object**
"Getting and Setting Attachment Information" on page 810

GetFieldValue

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`** or **`GetValueAsList`** methods.

You can use a field path name as the argument to this method. For more information, see "Using Field Path Names to Retrieve Field Values" on page 834.

Syntax

VBScript

*entity.***GetFieldValue** *(field_name)*

Perl

*$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. You can also use a **`FieldPathName`**.
Return value	The FieldInfo object corresponding to the specified field.

Examples

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
```

Perl

```
$sessionobj = $entity->GetSession();


# Iterate through the fields and output

# the field name and type.


$fieldnamelist = $entity->GetFieldNames();

foreach $fieldname (@$fieldnamelist)

   {
```

```
$fieldinfoobj = $entity->GetFieldValue($fieldname);
$fieldvalue = $fieldinfoobj->GetValue();
   $sessionobj->OutputDebugString("Field name: ".$fieldname. ",
         value=".$fieldvalue);
 }
```

See Also

`AddFieldValue`
`DeleteFieldValue`
`GetAllFieldValues`
`SetFieldValue`
`GetValue` of the **FieldInfo Object**
`GetValueAsList` of the **FieldInfo Object**
"Showing Changes to an Entity (Record)" on page 838
"Showing Changes to a FieldInfo (Field)" on page 836
"Getting a List of Defects and Modifying a Record" on page 847
`FieldPathName` of the **`QueryFieldDef Object`**

GetInvalidFieldValues

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.

Syntax

VBScript

entity.**GetInvalidFieldValues**

Perl

$*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 Visual Basic, 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.

Examples

VBScript

```
' Iterate through the fields and examine the field names and values
fieldObjs = GetInvalidFieldValues
For Each field In fieldObjs
   fieldValue = field.GetValue
   fieldName = field.GetName
   ' ...
Next
```

Perl

```
# Get the list of field values
$fieldvalues = $entity->GetInvalidFieldValues();


$numfields = $fieldvalues->Count();


for ($x = 0; $x < $numfields ; $x++)
   {
   $field = $fieldvalues->Item($x);
   $fieldvalue = $field->GetValue();
   $fieldname = $field->GetName();
   # ... other field commands
}
```

See Also

`GetAllFieldValues`
`GetFieldValue`
`Validate`
`ValidityChangedThisSetValue` of the **FieldInfo Object**

GetLegalActionDefNames

Description

Returns a list of accessible actions for a given record (entity). Returns the names of the actions that can be used on this Entity object. These are the actions that are permissible by the user group access control.

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.

The accessible list is computed based on the name access list for the actions and base actions. If the method of access control is by hooks either for the action itself or any base actions, they are included in the list but are not executed. Users may get a "permission denied" error message when they execute one of those actions, as they do if the access control hooks return not permissible.

Syntax

VBScript

entity.**GetLegalActionDefNames**

Perl

$*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 Visual Basic, 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 an array of strings.

Examples

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
```

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();

foreach $actionname(@$actiondeflist)
   {
 $actiondeftype = $entitydefobj->GetActionDefType($actionname);
 if ($actiondeftype eq $CQPerlExt::CQ_MODIFY)
  {
  $sessionobj->EditEntity($entity,$actionname);
  }
 }
```

See Also

`GetActionDefNames`
`EditEntity` of the **Session Object**
"Notation Conventions for Perl" on page 2
"Notation Conventions for VBScript" on page 3

GetOriginal

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.

Syntax

VBScript

entity.**GetOriginal**

Perl

$*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.

Examples

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
```

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
GetDuplicates
GetOriginalID
HasDuplicates
IsDuplicate
IsOriginal
MarkEntityAsDuplicate of the **Session Object**
UnmarkEntityAsDuplicate of the **Session Object**
"Updating Duplicate Records to Match the Parent Record" on page 820

GetOriginalID

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.

Syntax

VBScript

*entity.***GetOriginalID**

Perl

*$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.

Examples

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
```

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`
`GetDuplicates`
`GetOriginal`
`HasDuplicates`
`IsDuplicate`
`IsOriginal`
`MarkEntityAsDuplicate` of the **Session Object**
`UnmarkEntityAsDuplicate` of the **Session Object**
"Updating Duplicate Records to Match the Parent Record" on page 820

GetSession

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 stand-alone application, you cannot call this method to obtain a Session object. You must create your own Session object and pass it to any stand-alone 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.

Syntax

VBScript

*entity.***GetSession**

Perl

*$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.

Examples

VBScript

```
set sessionObj = entity.GetSession
```

Perl

```
$sessionobj = $entity->GetSession();
```

See Also

`UserLogon` of the **Session Object**
"Updating Duplicate Records to Match the Parent Record" on page 820

GetType

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.

Syntax

VBScript

*entity.*__GetType__

Perl

*$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.

Examples

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
```

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" on page 2
"Notation Conventions for VBScript" on page 3

HasDuplicates

Description

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`** and **`IsOriginal`** methods for details.

Syntax

VBScript

entity.**HasDuplicates**

Perl

$*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.

Examples

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
```

Perl

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

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

See Also

`GetAllDuplicates`
`GetDuplicates`
`GetOriginal`
`GetOriginalID`
`IsDuplicate`
`IsOriginal`
`MarkEntityAsDuplicate` of the **Session Object**
`UnmarkEntityAsDuplicate` of the **Session Object**
"Updating Duplicate Records to Match the Parent Record" on page 820

InvalidateFieldChoiceList

Description

Erases the values in a choice list, which can then be reset with **`SetFieldChoiceList`**.

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.

Syntax

VBScript

`entity``.`**InvalidateFieldChoiceList** *field_name*

Perl

`$`*`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.

See Also

`SetFieldChoiceList`

IsDuplicate

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.

Syntax

VBScript

*`entity.`***IsDuplicate**

Perl

*`$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.

Examples

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
```

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
GetDuplicates
GetOriginal
HasDuplicates
IsOriginal
MarkEntityAsDuplicate of the **Session Object**
UnmarkEntityAsDuplicate of the **Session Object**
"Updating Duplicate Records to Match the Parent Record" on page 820

IsEditable

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 **`BuildEntity`** or open an existing object for editing with **`EditEntity`**. An Entity object remains editable until you either commit your changes with the **`Commit`** method or revert the Entity object with the **`Revert`** method.

Syntax

VBScript

*`entity.`***IsEditable**

Perl

*`$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.

Examples

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
```

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`
`Revert`
`BuildEntity` of the **Session Object**
`EditEntity` of the **Session Object**

IsOriginal

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.

Syntax

VBScript

*`entity.`***IsOriginal**

Perl

*`$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.

Examples

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
```

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
GetDuplicates
GetOriginal
GetOriginalID
HasDuplicates
IsDuplicate
MarkEntityAsDuplicate of the **Session Object**
UnmarkEntityAsDuplicate of the **Session Object**
"Updating Duplicate Records to Match the Parent Record" on page 820

LoadAttachment

Description Loads an attachment to the specified destination filename.

Syntax

VBScript

```
entity.LoadAttachment attachment_fieldname element_displayname
destination_filename
```

Perl

```
$entity->LoadAttachment(attachment_fieldname, element_displayname,
destination_filename);
```

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).
attachment_fieldname	A String containing the attachment field name.
element_displayname	A String containing the attachment file name.
destination_filename	A String containing the destination filename.
Return value	Returns a Long. The String returned is empty if the update is permitted, or an explanation of the error. .

See Also `Attachment Object`

LookupStateName

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.

Syntax

VBScript

*entity.***LookupStateName**

Perl

*$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 ('').

Examples

VBScript

```
' Find the entity's current state name
currentState = LookupStateName
```

Perl

```
# Find the entity's current state name
$currentstate = $entity->LookupStateName();
```

See Also

`GetFieldValue`
`EditEntity` of the **Session Object**

Revert

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.

Syntax

VBScript

*entity.*__Revert__

Perl

*$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.

Examples

VBScript

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

' Revert the changes to the record
entityToEdity.Revert
```

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`
`IsEditable`
`Validate`
`EditEntity` of the **Session Object**
“Extracting Data About an EntityDef (Record Type)” on page 828

SetFieldChoiceList

Description

Sets a list of acceptable values for the field. Resets a dynamic choice list. Can be used with **`InvalidateFieldChoiceList`** to empty any values already stored.

Use this function to force the ClearQuest client to fetch the new choice list values for the field.

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`** 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`** 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.

Syntax

VBScript

*`entity.`***SetFieldChoiceList** *`fieldName, (choiceList)`*

Perl

`$`*`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	For VB, a Variant containing an array of strings. For Perl, a reference to an array of strings.
Return value	None.

Examples

VBScript

```
fieldchoicelist3 = array("hello", "world", "goodbye")
SetFieldChoiceList "severity",  (fieldchoicelist3)
```

Perl

```
$entity->SetFieldChoiceList($fieldname, \@choiceList);
# Add choices by adding strings to the array of field choices
```

See Also

`GetFieldChoiceList`
`InvalidateFieldChoiceList`
"Creating a Dependent Choice List" on page 854

SetFieldRequiredness ForCurrentAction

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. After 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.

Syntax

VBScript

entity.`SetFieldRequiredness ForCurrentAction` *field_name, newValue*

Perl

$entity->`SetFieldRequiredness ForCurrentAction(`*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.
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.

Examples

VBScript

```
' Change all mandatory fields to optional
fieldNameList = GetFieldNames
For Each fieldName in fieldNameList
   fieldReq = GetFieldRequiredness(fieldName)
   if fieldReq = AD_MANDATORY Then
      SetFieldRequiredness ForCurrentAction fieldName, AD_OPTIONAL
   End If
Next
```

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 eq $CQPerlExt::CQ_MANDATORY)
   {
   # Since it is, make it optional
   $entity->SetFieldRequiredness ForCurrentAction($fieldname,
                                    $CQPerlExt::CQ_OPTIONAL);
```

```
    }
}
```

See Also

`GetFieldRequiredness`
`EditEntity` of the **Session Object**
`Behavior Constants`
"Notation Conventions for Perl" on page 2
"Notation Conventions for VBScript" on page 3

SetFieldValue

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.

Syntax

VBScript

entity.**SetFieldValue** *field_name, new_value*

Perl

$*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 Visual Basic, a Variant containing the new value 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.

Examples

VBScript

```
' Set two field values, but only check errors for
' the second field.
entity.SetFieldValue "field1", "new value"
returnVal = SetFieldValue("field2", "100")
```

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` of the **Session Object**
`EditEntity` of the **Session Object**
`GetFieldType`
`AddFieldValue`
`GetFieldOriginalValue`
`GetFieldValue`
`ValidityChangedThisSetValue`
`ValueChangedThisSetValue`
FieldInfo Object
"Updating Duplicate Records to Match the Parent Record" on page 820
"Extracting Data About an EntityDef (Record Type)" on page 828
"Getting a List of Defects and Modifying a Record" on page 847

SiteHasMastership

Description

Tests whether this object is mastered in the local, session database and returns True if it is mastered by the local site and otherwise returns False.

This method supports MultiSite operations.

An object can be modified or deleted only in its master database. An object's initial master database is the database in which it is first created, but the master database can be changed by using the MultiUtil tool.

Syntax

VBScript

entity.**SiteHasMastership**

Perl

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

Identifier	Description
entity	An Entity object representing a user data record.
Return value	The return value is Boolean True if this object is mastered in the session database, and otherwise returns False.

Examples

VBScript

```
is_mastered_locally = entity.SiteHasMastership
```

Perl

```
$is_mastered_locally = $entity->SiteHasMastership();
```

See Also

SiteHasMastership in Group
SiteHasMastership in User
SiteHasMastership in Workspace

Validate

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.

Syntax

VBScript

*`entity.`***Validate**

Perl

*`$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.

Examples

VBScript

```
set sessionObj = GetSession

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

' modify the Entity object


status = entityObj.Validate
     if status = "" then
        entityObj.Commit
     else
        entityObj.Revert
     End If
' The Entity object is no longer editable
```

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

$status = $entityobj->Validate();
   if ($status == ""){
      $entityobj->Commit();
      }
   else {
         $entityobj->Revert();
      }
# At this point, the entity object is no longer modifiable
```

See Also

Commit
GetInvalidFieldValues
Revert
FieldInfo Object
"Updating Duplicate Records to Match the Parent Record" on page 820
"Getting a List of Defects and Modifying a Record" on page 847

EntityDef Object

14

An EntityDef object represents one of the *record types* 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 Rational 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`**, **`GetActionDefNames`**, and **`GetStateDefNames`** methods to obtain the names of the record type's fields, actions, and states, respectively. You can also use the **`GetFieldDefType`** or **`GetActionDefType`** methods 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)" on page 828

EntityDef Object Methods

The following list summarizes the EntityDef object methods:

Method Name	Description
`CanBeSecurityContext`	Tests whether access to a record type can be controlled with security privileges.
`CanBeSecurityContextField`	Tests whether access to a field can be controlled with security privileges.
`DoesTransitionExist`	Returns the list of transitions that exist between two states.
`GetActionDefNames`	Returns the action names defined in the EntityDef object.
`GetActionDefType`	Identifies the type of the specified action.
`GetActionDestStateName`	Returns the name of the destination state of a given action def.
`GetFieldDefNames`	Returns the field names defined in the EntityDef object.
`GetFieldDefType`	Identifies the type of data that can be stored in the specified field.
`GetFieldReferenceEntityDef`	Returns the type of record referenced by the specified field.
`GetHookDefNames`	Returns the list of named hooks associated with records of this type.
`GetLocalFieldPathNames`	Returns the path names of local fields.
`GetName`	Returns the name of the EntityDef object's corresponding record type.
`GetStateDefNames`	Returns the state names defined in the EntityDef object.
`GetType`	Returns the type (state-based or stateless) of the EntityDef.
`IsActionDefName`	Identifies whether the EntityDef object contains an action with the specified name.
`IsSecurityContext`	Tests whether this record type is used as a security context.
`IsSecurityContextField`	Tests whether a field is used as a security context.
`IsFamily`	Returns true of a given entitydef defines a family.
`IsFieldDefName`	Identifies whether the EntityDef object contains a field with the specified name.
`IsStateDefName`	Identifies whether the EntityDef object contains a state with the specified name.
`IsSystemOwnedFieldDefName`	Returns a Bool indicating whether the specified field is owned by the system.

CanBeSecurityContext

Description

Tests whether access to a record type can be controlled with access and action security privileges.

Syntax

VBScript

entitydef.**CanBeSecurityContext**

Perl

$*entitydef*->**CanBeSecurityContext()**;

Identifier	Description
entitydef	An EntityDef object corresponding to a record type in a schema.
Return value	The return value is Boolean True if the EntityDef (record type) object can be used as a security context; otherwise the return value is Boolean False.

Security can be managed at the database, record type, and field levels. This method tests security properties at the record type level.

Security cannot be managed by ClearQuest APIs. To manage security, you need to use the ClearQuest designer and ClearQuest client.

Examples

VBScript

can_be_secure = entitydef.**CanBeSecurityContext**

Perl

$*can_be_secure* = $*entitydef*->**CanBeSecurityContext**();

See Also

CanBeSecurityContextField,
IsSecurityContext,
IsSecurityContextField.

CanBeSecurityContextField

Description Tests whether a field can be used as a security context field.

Syntax

VBScript

```
entitydef.CanBeSecurityContextField("field_name")
```

Perl

```
$entitydef->CanBeSecurityContextField($field_name);
```

Identifier	Description
entitydef	An EntityDef object corresponding to a record type in a schema.
field_name	A String containing the name of the field to be tested.
Return value	The return value is Boolean True if the field can be used as a security context field; otherwise the return value is Boolean False.

Security can be managed at the database, record type, and field levels. This method tests security properties at the field level.

Security cannot be managed by ClearQuest APIs. To manage security, you need to use the ClearQuest designer and ClearQuest client.

Examples

VBScript

```
can_be_security_field = entitydef.CanBeSecurityContextField("field_name")
```

Perl

```
$can_be_security_field = $entitydef->CanBeSecurityContextField($field_name);
```

See Also

CanBeSecurityContext
IsSecurityContext
IsSecurityContextField

DoesTransitionExist

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.

Syntax

VBScript

entitydef.`DoesTransitionExist` *sourceState, destState*

Perl

$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 Visual Basic, 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 an array of strings.

Examples

VBScript

```
set sessionObj  = GetSession
set entityDefObj = sessionObj.GetEntityDef(GetEntityDefName())

transitions = entityDefObj.DoesTransitionExist("open", "resolved")
If transitions <> Empty Then
   ' Simply initiate an action using the first entry.
   sessionObj.EditEntity entity, transitions(0)

   ' ...
End If
```

Perl

```
$sessionObj = $entity->GetSession();
$entityDefObj = $sessionObj->GetEntityDef($entity->GetEntityDefName());


$transitions = $entityDefObj->DoesTransitionExist("open",
      "resolved");


if (@$transitions)
 {
```

```
# Simply initiate an action using the first entry.
$sessionObj->EditEntity($entity, @$transitions[0]);
}
```

See Also

GetActionDefNames
IsActionDefName

GetActionDefNames

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.

Syntax

VBScript

entitydef.**GetActionDefNames**

Perl

$*entitydef*->**GetActionDefNames();**

Identifier	Description
entitydef	An EntityDef object corresponding to a record type in a schema.
Return value	For Visual Basic, 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 an array of strings.

Examples

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
```

Perl

```
$sessionObj = $entity->GetSession();
$sessionObj->GetEntityDef($entity->GetEntityDefName());


$sessionObj.OutputDebugString("Action names for "$entityDefObj->GetName());


$nameList = $entityDefObj->GetActionDefNames();

foreach $actionName(@$nameList)
   {
 $sessionObj->OutputDebugString($actionName);

 }
```

See Also **`GetActionDefType`**
`IsActionDefName`
`ActionType Constants`
“Extracting Data About an EntityDef (Record Type)” on page 828

GetActionDefType

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.

Syntax

VBScript

entitydef.**GetActionDefType** *action_def_name*

Perl

$*entitydef*->**GetActionDefType(***action_def_name***);**

Identifier	Description
entitydef	An EntityDef object corresponding to a record type in a schema.
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

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
```

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();

foreach $actionname (@$actiondeflist)
 {
  $actiondeftype = $entitydefobj->GetActionDefType($actionname);
  if ($actiondedftype eq $CQPerlExt::CQ_MODIFY)
   {
   $sessionobj->EditEntity($entity,$actionname);
   }
 }
```

See Also

GetActionDefNames
IsActionDefName
"Extracting Data About an EntityDef (Record Type)" on page 828
"Notation Conventions for VBScript" on page 3
"Notation Conventions for Perl" on page 2

GetActionDestStateName

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`** returns the default action name associated with the current state, this method returns the destination state name associated with the current action.

Syntax

VBScript

`entitydef``.`**GetActionDestStateName** *`actionDefName`*

Perl

`$`*`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`

GetFieldDefNames

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.

Syntax

VBScript

entitydef.**GetFieldDefNames**

Perl

$*entitydef*->**GetFieldDefNames();**

Identifier	Description
entitydef	An EntityDef object corresponding to a record type in a schema.
Return value	For Visual Basic, 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 an array of strings.

Examples

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
```

Perl

```
$sessionObj = $entity->GetSession();
$entityDefObj =
     $sessionObj->GetEntityDef($entity->GetEntityDefName());


$sessionObj.OutputDebugString("Field names for "$entityDefObj->GetName());


$nameList = $entityDefObj->GetFieldDefNames());

foreach $fieldName (@$nameList)
   {
 $sessionObj->OutputDebugString($fieldName);

 }
```

See Also **`GetFieldDefType`**
`IsFieldDefName`
“Extracting Data About an EntityDef (Record Type)” on page 828

GetFieldDefType

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.

Syntax

VBScript

`entitydef`.**GetFieldDefType** *`field_def_name`*

Perl

$*`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`**.

Examples

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
```

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();
```

```
foreach $fieldName (@$nameList)
   {
 $fieldType = $entityDefObj->GetFieldDefType($fieldName);
 if ($fieldType eq $CQPerlExt::CQ_INT)
  {
  $sessionObj->OutputDebugString($fieldName);
  }
 }
```

See Also

GetFieldDefNames
IsFieldDefName
“Extracting Data About an EntityDef (Record Type)” on page 828
“Notation Conventions for VBScript” on page 3
“Notation Conventions for Perl” on page 2

GetFieldReferenceEntityDef

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.

Syntax

VBScript

entitydef.**GetFieldReferenceEntityDef** *field_name*

Perl

$*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.

Examples

VBScript

```
set sessionObj  = GetSession
set entityDefObj = sessionObj.GetEntityDef(GetEntityDefName())

' List the type of reference 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
```

Perl

```
$sessionObj = $entity->GetSession();
$entityDefObj = $sessionObj->GetEntityDef($entity->GetEntityDefName());


# List the type of reference fields
$nameList = $entityDefObj->GetFieldDefNames();
foreach $fieldName (@$nameList)
   {
 $fieldType = $entityDefObj->GetFieldDefType($fieldName);
 if ($fieldType eq $CQPerlExt::CQ_REFERENCE)
  {
  $refEDefObj = $entityDefObj->GetFieldReferenceEntityDef($fieldName);
  $sessionObj->OutputDebugString($refEDefObj->GetName());
```

```
    }
}
```

See Also `GetFieldDefType`
"Notation Conventions for VBScript" on page 3
"Notation Conventions for Perl" on page 2

GetHookDefNames

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.

Syntax

VBScript

entitydef.**GetHookDefNames** *field_def_name*

Perl

$*entitydef*->**GetHookDefNames(***field_def_name***);**

Identifier	Description
entitydef	An EntityDef object corresponding to a record type in a schema.
Return value	For Visual Basic, 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 an array of strings.

Examples

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
```

Perl

```
$sessionObj = $entity->GetSession();
$entityDefObj = $sessionObj->GetEntityDef($entity->GetEntityDefName());


$sessionObj->OutputDebugString("Hooks of
      ".$entityDefObj->GetName());


# List the record type's hooks
$nameList = $entityDefObj->GetHookDefNames();
foreach $hookName (@$nameList)
   {
 $sessionObj->OutputDebugString($hookName);
 }
```

See Also `GetActionDefNames`
`GetFieldDefNames`

GetLocalFieldPathNames

Description

Returns the path names of local fields.

Each string in the returned Variant contains the path name of a single field. This might be a "dotted name" if it is from within a reference (for example, owner.login_name).

Syntax

VBScript

entitydef.**GetLocalFieldPathNames** *visible_only*

Perl

$*entitydef*->**GetLocalFieldPathNames(***visible_only***);**

Identifier	Description
entitydef	An EntityDef object corresponding to a record type in a schema.
visible_only	A Boolean, which if True restricts the list of returned fields to only those that are visible.
Return value	For Visual Basic, a Variant containing a list of strings is returned. For Perl, a reference to an array of strings is returned.

Examples

VBScript

```
set sessionObj  = GetSession
set entityDefObj = sessionObj.GetEntityDef(GetEntityDefName())

pathNames = entityDefObj.GetLocalFieldPathNames(False)
For Each name in pathNames
   sessionObj.OutputDebugString "Path name: " & name
Next
```

Perl

```
$sessionobj = $entity->GetSession();


$entitydefobj = $sessionobj->GetEntityDef($entity->GetEntityDefName());


$pathnames = $entitydefobj->GetLocalFieldPathNames(0);


foreach $name (@$pathnames)
   {
 $sessionobj->OutputDebugString("Path name: ".$name);
 }
```

See Also

FieldPathName of the **QueryFieldDef Object**
"Using Field Path Names to Retrieve Field Values" on page 834
GetFieldDefNames
IsFieldDefName

GetName

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.

Syntax

VBScript

entitydef.**GetName**

Perl

$*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.

Examples

VBScript

```
set sessionObj  = GetSession
set entityDefObj = sessionObj.GetEntityDef(GetEntityDefName())
sessionObj.OutputDebugString "Name of record type: " & _
        entityDefObj.GetName()
```

Perl

```
$sessionobj = $entity->GetSession();
$entitydefobj = $sessionobj->GetEntityDef($entity->GetEntityDefName());


$sessionobj.OutputDebugString("Name of record type:
      ".$entitydefobj->GetName());
```

See Also

GetType

"Extracting Data About an EntityDef (Record Type)" on page 828

GetStateDefNames

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.

Syntax

VBScript

entitydef.**GetStateDefNames**

Perl

$*entitydef*->**GetStateDefNames();**

Identifier	Description
entitydef	An EntityDef object corresponding to a record type in a schema.
Return value	For Visual Basic, 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 an array of strings.

Examples

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
```

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();
```

```
foreach $stateName (@$nameList)
   {
  $sessionObj->OutputDebugString($stateName)
  }
 }
```

See Also

GetType
IsStateDefName
“Extracting Data About an EntityDef (Record Type)” on page 828
“Notation Conventions for VBScript” on page 3
“Notation Conventions for Perl” on page 2

GetType

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.

Syntax

VBScript

entitydef.**GetType**

Perl

$*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.

Examples

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
```

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();
foreach $statename (@$nameList)
   {
  $sessionobj->OutputDebugString($statename);
```

```
    }
}
```

See Also `GetName`

"Extracting Data About an EntityDef (Record Type)" on page 828
"Notation Conventions for VBScript" on page 3
"Notation Conventions for Perl" on page 2

IsActionDefName

Description Identifies whether the EntityDef object contains an action with the specified name.

Syntax

VBScript

entitydef.**IsActionDefName** *name*

Perl

$*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.

Examples

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
```

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**
GetActionDefType

IsFamily

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.

Syntax

VBScript

entitydef.**IsFamily**

Perl

$*entitydef*->**IsFamily();**

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
GetEntityDefFamilyNames

IsFieldDefName

Description Identifies whether the EntityDef object contains a field with the specified name.

Syntax

VBScript

entitydef.**IsFieldDefName** *name*

Perl

$*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**
GetFieldDefType

IsSecurityContext

Description

Tests whether a record type (an EntityDef object) is used as a security context, which means access to it requires that the user is in one of the groups in the `ratl_context_groups` field.

Syntax

VBScript

entitydef.**IsSecurityContext**

Perl

*$entitydef->***IsSecurityContext()**;

Identifier	Description
entitydef	An EntityDef object corresponding to a record type in a schema.
Return value	The return value is Boolean True if this EntityDef object is used as a security context record type; otherwise the return value is Boolean False.

Security can be managed at the database, record type, and field levels. This method tests security properties at the record type level.

Security cannot be managed by ClearQuest APIs. To manage security, you need to use the ClearQuest designer and ClearQuest client.

Examples

VBScript

is_secure = entitydef.**IsSecurityContext**

Perl

*$is_secure = $entitydef->***IsSecurityContext()**;

See Also

CanBeSecurityContext
CanBeSecurityContextField
IsSecurityContextField

IsSecurityContextField

Description

Tests whether a field is used as a security context, which means access to it requires that the user is in one of the groups in the `ratl_context_groups` field.

Syntax

VBScript

entitydef.**IsSecurityContextField** *fieldname*

Perl

$entitydef*->**IsSecurityContext**(*$fieldname);

Identifier	Description
entitydef	An EntityDef object corresponding to a record type in a schema.
fieldname	A String containing the name of the field to be tested.
Return value	The return value is Boolean True if this field is used as a security context field; otherwise the return value is Boolean False.

Security can be managed at the database, record type, and field levels. This method tests security properties at the field level.

Security cannot be managed by ClearQuest APIs. To manage security, you need to use the ClearQuest designer and ClearQuest client.

Examples

VBScript

is_secure = entitydef.**IsSecurityContextField** *fieldname*

Perl

$is_secure = $entitydef*->**IsSecurityContext**(*$fieldname);

See Also

CanBeSecurityContext
CanBeSecurityContextField
IsSecurityContext

IsStateDefName

Description Identifies whether the EntityDef object contains a state with the specified name.

Syntax

VBScript

entitydef.**IsStateDefName** *name*

Perl

$*entitydef*->**IsStateDefName(***name***);**

Identifier	Description
entitydef	An EntityDef object corresponding to a record type in a schema.
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`

"Extracting Data About an EntityDef (Record Type)" on page 828

IsSystemOwnedFieldDefName

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.

Syntax

VBScript

entitydef.**IsSystemOwnedFieldDefName** *field_name*

Perl

$*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 the EntityDef.
Return value	True if the field is owned by the system, otherwise False.

See Also `GetFieldDefNames`

EntityDefs Object

15

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`** method. Use the **`Item`** method to retrieve items from the collection.

See Also **EntityDef Object**

EntityDefs Object Properties

The following list summarizes the EntityDefs object properties:

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

Count

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

Syntax

VBScript

collection.**Count**

Perl

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

Identifier	Description
collection	An EntityDefs collection object, representing the set of EntityDefs available for fetching as a collection.
Return value	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`**
EntityDef Object

EntityDefs Object Methods

The following list summarizes the EntityDefs object methods:

Method Name	Description
`Item`	Returns the specified item in the collection.
`ItemByName`	(Perl only) Returns the specified item in the collection.

Note: For Perl methods that map to Visual Basic Properties, see the Properties section of this object.

The following list summarizes additional Perl EntityDefs object methods:

Method Name	Access	Description
`Count`	Read-only	Returns the number of items in the collection.

Item

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

VB

collection.**Item** (*itemNum*)

collection.**Item** (*name*)

Perl

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

$*collection*->**ItemByName**(*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`** of the desired EntityDefs.
Return value	The EntityDef object at the specified location in the collection.

See Also **`Count`**

EventObject Object

16

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.

Understanding Record Scripts

Record scripts are a generic form of hook that are called in response to an event on a Rational ClearQuest form or from other hooks. Typically, record scripts are used to implement an action that you want to perform in response to a click event on a push button or on a context menu item associated with a particular field on a ClearQuest form. Record scripts are scripts that can be executed within the context of one record type.

If you associate a record script to a push button, when a user pushes the button, the script is executed.

All record scripts have the following syntax:

```
Function RecordTypeName_ScriptName (param)
   ' input param As Variant
   'output RecordTypeName As Variant
   'The content of the script...
End Function
```

A record script:

- Can be called upon in field hooks, action hooks and other record scripts of the same record type.
- Is usually triggered by a form control to perform specific tasks at runtime.

When associated with a form control, the parameter passed into the method contains an instance of the EventObject class. This instance contains information about the event that caused the hook to be called. (See Form Control Events for information on these events.)

When calling a record script from another hook, the parameter you pass into the method is a Variant containing whatever data is appropriate. If the record script returns data to the calling hook, that information is returned as a Variant as well.

You can associate one or more record scripts to a form control.

Form Control Events

When a record script is triggered by a form control, ClearQuest passes the record script an EventObject object as its parameter. This object contains information about the type of event that occurred. Different controls can generate different types of events, including button clicks, item selections, and so on. You must use the information in the EventObject object to determine how to handle the event.

ClearQuest generates the following types of events for form controls:

- Button-click — Indicates that the user clicked a push button control.
- Shortcut menu item-selection — Indicates that the user invoked the hook from a shortcut menu.

The following table lists the supported type of event for each control and the extra information provided by the EventObject. The constants listed under the supported event type column are part of the **`EventType Constants`** enumerated type.

Control Type	Supported Event Type	Additional Information
Push button	_BUTTON_CLICK	Button name
Combo box	_CONTEXMENU_ITEM_SELECTION	Null string
Drop-down list box	_CONTEXMENU_ITEM_SELECTION	Null string
List box	_CONTEXMENU_ITEM_SELECTION	Current field value selection
List view	_CONTEXMENU_ITEM_SELECTION	Current field value selected
Text box	_CONTEXMENU_ITEM_SELECTION	Current field value selected
Drop-down combo box	_CONTEXMENU_ITEM_SELECTION	Null string

See Also

`EventType`

`EventType Constants`

Entity Object

EventObject Object Properties

The following list summarizes the EventObject object properties (Visual Basic) and methods (Perl):

Property Name	Access	Description
`CheckState`	Read-only	Returns the check state of an associated check box.
`EditText`	Read-only	Returns the check state of an associated edit control.
`EventType`	Read-only	Returns the type of event that caused the hook invocation.
`ItemName`	Read-only	Returns the name of the form item that caused the hook to be invoked.
`ListSelection`	Read-only	Returns the primary key of the selected object.
`ObjectItem`	Read-only	(Visual Basic only) Returns the entity object associated with the current selection.
`StringItem`	Read-only	Returns the name of the menu item hook.

CheckState

Description Returns the check state of an associated CheckBox .

Syntax

VBScript

eventObject.**CheckState**

Perl

$*eventObject*->**CheckState();**

Identifier	Description
eventObject	An instance of EventObject.
Return value	A Bool containing the check state of an associated CheckBox . Returns True, if checked, and False if unchecked.

See Also `ObjectItem`

EditText

Description Returns the text in an associated Edit control.

Syntax

VBScript

eventObject.**EditText**

Perl

$*eventObject*->**EditText();**

Identifier	Description
eventObject	An instance of EventObject.
Return value	A String containing the text in an associated Edit control.

See Also **ObjectItem**

EventType

Description Returns the type of event that caused the hook invocation. This is a read-only property; it can be viewed but not set.

Syntax

VBScript

eventObject.**EventType**

Perl

$*eventObject*->**EventType();**

Identifier	Description
eventObject	An instance of EventObject.
Return value	A Long whose value is one of the EventType enumerated constants.

Examples

VBScript

```
Dim eventType
eventType = param.EventType
if eventType = AD_BUTTON_CLICK Then
    SetFieldValue “KeyCustomer”, “Company A”
elseif eventType = AD_CONTEXMENU_ITEM_SELECTION Then
    SetFieldValue “KeyCustomer”, “Company B”
end if
```

Perl

```
my $eventType;
$eventType = $param->EventType();
if ($eventType == $CQPerlExt::CQ_BUTTON_CLICK) {
  $entity->SetFieldValue("KeyCustomer", "Company A");
} else {
           if ($eventType == $CQPerlExt::CQ_CONTEXMENU_ITEM_SELECTION) {
          $entity->SetFieldValue("KeyCustomer", "Company B");
     }
}
```

See Also **EventType Constants**

ItemName

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.

Syntax

VBScript

eventObject.**ItemName**

Perl

$*eventObject*->**ItemName();**

Identifier	Description
eventObject	An instance of EventObject.
Return value	A String containing the name of the form item from which the hook was invoked.

See Also `ObjectItem`

ListSelection

Description

Returns the database id of a highlighted record within your listview control.

Note: This function is for COM only. It is not available for Perl. It is not available within the web interface.

You can use this property in response to a button click event (that is, the AD_BUTTON_CLICK event type) to find out what value is selected in a parent / child list box. The methods returns the primary key of the referenced record type.

In order to get a list selection, you must associate your button with the list control (such as a parent child control) that you want to be able to select an item from. You must also select the List View type Other. Then, when you press the button the value returned is the key of the referenced record (parts of multipart keys are separated by spaces).

Syntax

VBScript

eventObject.**ListSelection**

Identifier	Description
eventObject	An instance of EventObject.
Return value	Returns a Variant array containing a single String value (or an empty array if no selection is made). The value returned contains the key of the referenced record (parts of multipart keys are separated by spaces).

Examples

VBScript

```
' The following script is invoked  when a user presses a button named "Select"
' that is associated with a ListView control and performs an action of type
' "Other" (on the extended properties tab)):
   Function Defect_Cust_Sel(param)
      ' param As Variant
      dim ListSel, Sel
      On Error Resume Next
      ListSel = param.ListSelection
      Sel = ListSel(0)
      SetFieldValue "Customer", Sel
      End Function


' The following example checks for event type, session type, and whether or
' not something is selected:
   function MyRecordHook(param)
    ' param As Variant
    ' record type name isMyRecord
      Dim ListSel
```

```
Dim Item
' Check its an event which we can have a selection for
if param.eventtype = AD_BUTTON_CLICK then
' Make sure we're not on the web since ListSelection doesn't work there
   if not GetSession.HasValue("_CQ_WEB_SESSION") then
      ' OK we're not on the web. Now check to see if anything is selected
      ListSel = param.ListSelection
      if ubound(ListSel) < lbound(ListSel) then
         ' Nothing is selected
      else
         Item = ListSel(0)
         ' ListSel is an array of strings with one element when
         ' something is selected
         ' and no elements when nothing is selected
         ' Put your code here to do what you need to do
         msgbox "Selected item was:" & Item
      end if
   else
      ' Web interface, ListSelection API call doesn't work here
   end if
else
   ' Its not a button click event, listselection only works with
   ' button click events
end if
end function
```

See Also **EventType Constants**
ObjectItem

ObjectItem

Description

(Visual Basic only) 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 CONTEXMENU_ITEM_SELECTION or BUTTON_CLICK.

Syntax

VBScript

eventObject.**ObjectItem**

Identifier	Description
eventObject	An instance of EventObject.
Return value	The Entity object associated with the current selection.

See Also

EventType
Entity Object

StringItem

Description Returns the name of the menu item hook. This is a read-only property; it can be viewed but not set.

Syntax

VBScript

eventObject.**StringItem**

Perl

$*eventObject*->**StringItem();**

Identifier	Description
eventObject	An instance of EventObject.
Return value	A String whose value indicates the menu item hook name when the EventType property contains the value CONTEXMENU_ITEM_CONDITION; otherwise, this property contains an empty value for all other event types.

See Also **EventType**

"Notation Conventions for VBScript" on page 3
"Notation Conventions for Perl" on page 2

FieldInfo Object

17

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`** and the **`GetFieldChoiceList`** methods.

See Also

`GetFieldsUpdatedThisAction`
`GetFieldsUpdatedThisGroup`
`GetFieldsUpdatedThisSetValue`
Entity Object
"Extracting Data About a Field in a Record" on page 831
"Showing Changes to an Entity (Record)" on page 838

FieldInfo Object Methods

The following list summarizes the FieldInfo object methods:

Method Name	Description
`GetMessageText`	Returns a String explaining why the value stored in the field is invalid.
`GetName`	Returns the name of the field.
`GetRequiredness`	Identifies the *behavior* of the specified field.
`GetType`	Identifies the type of data that can be stored in this field.
`GetValidationStatus`	Identifies whether the field's value is valid.
`GetValue`	Returns the field's value as a single String.
`GetValueAsList`	Returns an array of strings containing the values stored in the field.
`GetValueStatus`	Identifies whether the field currently has a value.
`ValidityChangedThisAction`	Returns True if the field's validity was changed by the current action.
`ValidityChangedThisGroup`	Returns True if the field's validity was changed by the most recent group of SetFieldValue calls.
`ValidityChangedThisSetValue`	Returns True if the field's validity was changed by the most recent **`SetFieldValue`** call.
`ValueChangedThisAction`	Returns True if this field's value was modified by the current action.
`ValueChangedThisGroup`	Returns True if the field's value was modified by the most recent group of SetFieldValue calls.
`ValueChangedThisSetValue`	Returns True if the field's value was modified by the most recent **`SetFieldValue`** call.

GetMessageText

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.

Syntax

VBScript

`fieldInfo``.`**GetMessageText**

Perl

`$`*`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`

"Extracting Data About a Field in a Record" on page 831
"Notation Conventions for VBScript" on page 3
"Notation Conventions for Perl" on page 2

GetName

Description Returns the name of the field. Field names are used by various methods to identify a specific field in an Entity object.

Syntax

VBScript

fieldInfo.**GetName**

Perl

$*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** of the **Entity Object**
"Extracting Data About a Field in a Record" on page 831

GetRequiredness

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, Rational ClearQuest will have already executed that hook and cached the resulting value. This method then returns the cached value.

Syntax

VBScript

fieldInfo.**GetRequiredness**

Perl

$*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 of the **Entity Object**
"Notation Conventions for VBScript" on page 3
"Notation Conventions for Perl" on page 2

GetType

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.)

Syntax

VBScript

`fieldInfo``.`**GetType**

Perl

`$`*`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` of the **Entity Object**
"Showing Changes to an Entity (Record)" on page 838
"Showing Changes to a FieldInfo (Field)" on page 836

GetValidationStatus

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.

Syntax

VBScript

fieldInfo.**GetValidationStatus**

Perl

$*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” on page 797.

See Also

`GetMessageText`

“Extracting Data About a Field in a Record” on page 831
“Notation Conventions for VBScript” on page 3
“Notation Conventions for Perl” on page 2

GetValue

Description

Returns the field's list of values 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.

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.

Fields whose type is either `ATTACHMENT_LIST` or `JOURNAL` cannot be modified programmatically.

Syntax

VBScript

`fieldInfo``.`**GetValue**

Perl

`$`*`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.

Example

Perl

```
my($FieldNamesRef) = $entity->GetFieldNames();
   # Loop through the fields, showing name, type, old/new value...
   foreach $FN (@$FieldNamesRef) {
      # Get the field's original value...
      $FieldInfo = $entity->GetFieldOriginalValue($FN);
      $FieldValueStatus = $FieldInfo->GetValueStatus();
      if ($FieldValueStatus == $CQPerlExt::CQ_HAS_NO_VALUE) {
         $OldFV = "<no value>";
      } elsif ($FieldValueStatus == $CQPerlExt::CQ_VALUE_NOT_AVAILABLE) {
         $OldFV = "<value not available>";
      } elsif ($FieldValueStatus == $CQPerlExt::CQ_HAS_VALUE) {
         $OldFV = $FieldInfo->GetValue();
      } else {
         $OldFV = "<Invalid value status: " . $FieldValueStatus . ">";
      }
```

See Also

`GetValueAsList`
`GetFieldValue` of the Entity Object
"Extracting Data About a Field in a Record" on page 831

“Showing Changes to a FieldInfo (Field)” on page 836
“Showing Changes to an Entity (Record)” on page 838
“Getting a List of Defects and Modifying a Record” on page 847

GetValueAsList

Description

Returns a list of string values for the field associated with FieldInfo. This is useful for fields that contain more than one value, including `MULTILINE_STRING` field types and parent/child controls for reference list types (`REFERENCE_LIST`).

It is legal to use this method for a scalar field (that is, one that contains a single value). When used on a scalar field, 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.

Syntax

VBScript

`value = fieldInfo``.`**GetValueAsList**

Perl

`$value = $`*`fieldInfo->`*`GetValueAsList();`

Identifier	Description
fieldInfo	A FieldInfo object, which contains information about one field of a user data record.
Return value	For Visual Basic, a 1-element Variant Array is returned. The Variant contains the list of values, separated by vbLF. If the field contains no values, this method returns an Empty Variant. In Perl, a reference to an array of strings containing the values in the list.

Examples

VBScript

```
MyList = MyField.GetValueAsList
if not IsEmpty (MyList) then
   for each listItem in MyList
   '...
   next
end if
' You can separate the single variant that is returned into an array of
' string list elements by using the Split function:
av = GetFieldValue("multiline_string_field").GetValueAsList
if not IsEmpty(av) then
  array = Split(Cstr(av(0)),vbLF)
  u = UBound(array)
  for i = 0 to u
    ' ...
```

```
  next
end if
```

Perl

```
$asgs = $entity->GetFieldValue("Assignments")->GetValueAsList();
foreach my $asg (@$asgs) {
   # ...
   }
```

See Also

`GetValue`
`AddFieldValue` of the **Entity Object**
`GetFieldValue` of the **Entity Object**
`FieldType Constants`
"Showing Changes to a FieldInfo (Field)" on page 836
"Notation Conventions for VBScript" on page 3
"Notation Conventions for Perl" on page 2

GetValueStatus

Description Identifies whether the field currently has a value.

Syntax

VBScript

fieldInfo.**GetValueStatus**

Perl

$*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.

Example

Perl

```
my($FieldNamesRef) = $entity->GetFieldNames();
   # Loop through the fields, showing name, type, old/new value...
   foreach $FN (@$FieldNamesRef) {
      # Get the field's original value...
      $FieldInfo = $entity->GetFieldOriginalValue($FN);
      $FieldValueStatus = $FieldInfo->GetValueStatus();
      if ($FieldValueStatus == $CQPerlExt::CQ_HAS_NO_VALUE) {
         $OldFV = "<no value>";
      } elsif ($FieldValueStatus == $CQPerlExt::CQ_VALUE_NOT_AVAILABLE) {
         $OldFV = "<value not available>";
      } elsif ($FieldValueStatus == $CQPerlExt::CQ_HAS_VALUE) {
         $OldFV = $FieldInfo->GetValue();
      } else {
         $OldFV = "<Invalid value status: " . $FieldValueStatus . ">";
}
```

See Also

GetValue
GetFieldValue of the **Entity Object**
SetFieldValue of the **Entity Object**
"Extracting Data About a Field in a Record" on page 831
"Showing Changes to an Entity (Record)" on page 838

ValidityChangedThisAction

Description

Returns True if the field's validity was changed by the current 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.

Syntax

VBScript

fieldInfo.**ValidityChangedThisAction**

Perl

$*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 current action was initiated, otherwise False.

See Also

ValidityChangedThisGroup
ValidityChangedThisSetValue
ValueChangedThisAction
GetFieldsUpdatedThisAction of the **Entity Object**
"Extracting Data About a Field in a Record" on page 831

ValidityChangedThisGroup

Description

Returns True if the field's validity was changed by the current group of **`SetFieldValue`** 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.

Syntax

VBScript

`fieldInfo``.`**ValidityChangedThisGroup**

Perl

`$`*`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`**, otherwise False.

See Also

`ValidityChangedThisAction`
`ValidityChangedThisSetValue`
`ValueChangedThisGroup`
`BeginNewFieldUpdateGroup` **of the Entity Object**
`GetFieldsUpdatedThisGroup` of the **Entity Object**
`FieldValidationStatus Constants`
“Extracting Data About a Field in a Record” on page 831

ValidityChangedThisSetValue

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.

Syntax

VBScript

fieldInfo.**ValidityChangedThisSetValue**

Perl

$*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
ValidityChangedThisGroup
GetFieldsUpdatedThisSetValue of the **Entity Object**
SetFieldValue of the **Entity Object**
"Extracting Data About a Field in a Record" on page 831

ValueChangedThisAction

Description

Returns True if this field's value was modified by the current 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.

Syntax

VBScript

fieldInfo.**ValueChangedThisAction**

Perl

$*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 current action was initiated, otherwise False.

See Also

`ValueChangedThisGroup`
`ValueChangedThisSetValue`
`BuildEntity` of the **Session Object**
`EditEntity` of the **Session Object**
`GetFieldsUpdatedThisAction` of the **Entity Object**
"Extracting Data About a Field in a Record" on page 831

ValueChangedThisGroup

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.

Syntax

VBScript

`fieldInfo``.` **ValueChangedThisGroup**

Perl

`$`*`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`
`ValueChangedThisSetValue`
`BeginNewFieldUpdateGroup` of the **Entity Object**
`GetFieldsUpdatedThisGroup` of the **Entity Object**
`SetFieldValue` of the **Entity Object**
“Extracting Data About a Field in a Record” on page 831

ValueChangedThisSetValue

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.

Syntax

VBScript

fieldInfo.**ValueChangedThisSetValue**

Perl

$*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` of the **Entity Object**
`SetFieldValue` of the **Entity Object**
`FieldType Constants` of the **`Enumerated Constants`**
"Extracting Data About a Field in a Record" on page 831

FieldInfos Object

18

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`** method. Use the **`Item`** method to retrieve items from the collection.

Note: FieldInfos object and its methods are for usage with Perl only.

See Also **`FieldInfo Object`**

FieldInfos Object Methods

The following list summarizes the FieldInfos object methods:

Method Name	Description
`Add`	Adds a FieldInfo object to the collection.
`Count`	Returns the number of items in the collection.
`Item`	Returns the specified item in the collection.
`ItemByName`	Returns the specified item in the collection.

Add

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.

Syntax

Perl

```
$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`
`Item`

Count

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

Syntax

Perl

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

Identifier	Description
collection	A FieldInfos collection object.
Return value	A Long indicating the number of items in the collection object. This collection always contains at least one item.

See Also **Item**
Add

Item

Description Returns the specified item in the FieldInfos collection.

Syntax

Perl

```
$fieldinfos->Item(itemNum);
```

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.
Return value	The FieldInfo object at the specified location in the collection.

See Also **Count**
Add

ItemByName

Description Returns the specified item in the FieldInfos collection.

Syntax

Perl

```
$fieldinfos->ItemByName(name);
```

Identifier	Description
fieldinfos	A FieldInfos collection object, representing the set of FieldInfos in one field of a record.
name	A String that serves as a key into the collection. This string corresponds to the **GetName** of the desired FieldInfos.
Return value	The FieldInfo object at the specified location in the collection.

See Also

Count
Add

Group Object

19

A Group object contains information about a single group of users. It is a single element in the returned collection of Group objects, such as a list of all groups included in a specific schema repository.

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 Rational ClearQuest Designer. See “Updating User Database Information” on page 21.

See Also

Database Object
User Object

Group Object Properties

The following list summarizes the Group object properties:

Property Name	Access	Description
`Active`	Read/Write	Indicates whether or not the group is active.
`Name`	Read/Write	Sets or returns the name of the group.
`SubscribedDatabases`	Read-only	Returns the collection of databases to which this group is subscribed.
`Users`	Read-only	Returns the collection of users belonging to this group.

Active

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.

Syntax

VBScript

group.**Active**

group.**Active** *boolean_value*

Perl

```
$group->GetActive();
$group->SetActive(boolean_value);
```

Identifier	Description
group	A Group object, representing one set of users associated with the current schema repository.
boolean_value	A Bool that specifies whether or not the group is active.
Return value	A Bool indicating whether or not the group is active.

See Also

`Active` of the **User Object**

Name

Description Sets or returns the name of the group.

Syntax

VBScript

group.**Name**

group.**Name** *group_name*

Perl

```
$group->GetName();
$group->SetName(group_name);
```

Identifier	Description
group	A Group object, representing one set of users associated with the current schema repository.
group_name	A String representing the new name for the group.
Return value	A String containing the name of the group.

Example

VBScript

```
set adminSession = CreateObject("ClearQuest.AdminSession")
adminSession.Logon "admin", "", ""
set groupList = adminSession.Groups
for each groupObj in groupList
   groupName = groupObj.Name
   msgbox groupName
Next
```

See Also **Active**

SubscribedDatabases

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.

Syntax

VBScript

group.**SubscribedDatabases**

Perl

*$group->***GetSubscribedDatabases();**

Identifier	Description
group	A Group object, representing one set of users associated with the current schema repository.
Return value	A Databases collection object containing the databases to which this group is subscribed.

See Also

`SubscribeDatabase`
`UnsubscribeAllDatabases`
`SubscribedDatabases` of the User object
Database Object
Databases Object
User Object

Users

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 a User object. To add users to a group, use the AddUser method.

Syntax

VBScript

group.**Users**

Perl

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

Identifier	Description
group	A Group object, representing one set of users associated with the current schema repository.
Return value	A Users collection object containing the users belonging to this group.

See Also

`AddUser`
User Object
Users Object
"Adding and Removing Users in a Group" on page 857.

Group Object Methods

The following list summarizes the Group object methods:

Note: For all Perl Get and Set methods that map to Visual Basic Properties, see the Properties section of this object.

Method Name	Description
AddUser	Adds a user to this group.
RemoveUser	Removes a user from this group.
SiteHasMastership	Tests whether this group is mastered in the session database.
SubscribeDatabase	Subscribes this group to the specified database.
UnsubscribeAllDatabases	Unsubscribes the group from all databases.
UnsubscribeDatabase	Unsubscribes the group from the specified database.

Additional Perl Get and Set Methods that map to Visual Basic properties:

Method Name	Description
GetActive	Indicates whether or not the group is active.
GetName	Returns the name of the group.
GetSubscribedDatabases	Returns the collection of databases to which this group is subscribed.
GetUsers	Returns the collection of users belonging to this group.
SetActive	Specifies whether or not the group is active.
SetName	Sets the name of the group.

AddUser

Description Adds a user to this group.

Syntax

VBScript

group.**AddUser** *user*

Perl

$group->**AddUser**(*user*);

Identifier	Description
group	A Group object, representing one set of users associated with the current schema repository.
user	The User object corresponding to the user account.
Return value	None.

See Also

User Object

"Adding and Removing Users in a Group" on page 857.

RemoveUser

Description Removes a user from this group.

Syntax

VBScript

group.**RemoveUser** *user*

Perl

$group->**RemoveUser**(*user*);

Identifier	Description
group	A Group object, representing one set of users associated with the current schema repository.
user	The User object corresponding to the user account.
Return value	None.

See Also

User Object

"Adding and Removing Users in a Group" on page 857.

SiteHasMastership

Description

Tests whether this Group object is mastered in the local, session database and returns True if it is mastered in the local site and otherwise returns False.

This method supports MultiSite operations.

An object can be modified or deleted only in its master database. An object's initial master database is the database in which it is first created, but the master database can be changed by using the MultiUtil tool.

Syntax

VBScript

group.**SiteHasMastership**

Perl

$*group*->**SiteHasMastership()**;

Identifier	Description
group	A Group object, representing one set of users associated with the current schema repository.
Return value	Returns True if this object is mastered in the session database, and otherwise returns False.

See Also

SiteHasMastership in Entity
SiteHasMastership in User
SiteHasMastership in Workspace

SubscribeDatabase

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.

Syntax

VBScript

group.**SubscribeDatabase** *database*

Perl

*$group->***SubscribeDatabase(***database***);**

Identifier	Description
group	A Group object, representing one set of users associated with the current schema repository.
database	The Database object to which the group will be subscribed.
Return value	None.

See Also

UnsubscribeAllDatabases
UnsubscribeDatabase
SubscribedDatabases

UnsubscribeAllDatabases

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.

Syntax

VBScript

group.**UnsubscribeAllDatabases**

Perl

*$group->***UnsubscribeAllDatabases();**

Identifier	Description
group	A Group object, representing one set of users associated with the current schema repository.
Return value	None.

See Also

SubscribeDatabase
UnsubscribeDatabase
Active
SubscribedDatabases

UnsubscribeDatabase

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.

Syntax

VBScript

group.**UnsubscribeDatabase** *database*

Perl

*$group->***UnsubscribeDatabase(***database***);**

Identifier	Description
group	A Group object, representing one set of users associated with the current schema repository.
database	The Database object from which the group will be unsubscribed.
Return value	None.

See Also

SubscribeDatabase
UnsubscribeAllDatabases
SubscribedDatabases

Groups Object

20

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 Rational ClearQuest Designer.

See Also **Group Object**

Groups Object Properties

The following list summarizes the Groups object properties:

PropertyName	Access	Description
`Count`	Read-only	Returns the number of items in the collection.

Count

Description

Returns the number of items in the collection.

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

Syntax

VBScript

collection.**Count**

Perl

*$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`

"Adding and Removing Users in a Group" on page 857.

Groups Object Methods

The following list summarizes the Groups object methods:

Method Name	Description
`Item`	Returns the item at the specified index in the collection.
`ItemByName`	(Perl only) Returns the specified item in the collection.

Note: For Perl methods that map to Visual Basic Properties, see the Properties section of this object.

The following list summarizes additional Perl Groups object methods:

Method Name	Access	Description
`Count`	Read-only	Returns the number of items in the collection.

Item

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

VBScript

```
collection.Item(itemNum)
collection.Item(name)
```

Perl

```
$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.

Example

VBScript

```
set adminSession = CreateObject("ClearQuest.AdminSession")
adminSession.Logon "admin", "", ""
set groupList = adminSession.Groups
numGroups = groupList.Count
For x = 0 to numGroups -1
   set groupObj= groupList.Item(x)
   groupName = groupObj.Name
   msgbox groupName
Next
```

Perl

```
$adminSession= CQAdminSession::Build();
$adminSession->Logon ("admin", "", "");
$groupList = $adminSession->Groups();
$numGroups = $groupList->Count();
for (x = 0;$x < $numGroups ; $x++){
   $groupObj = $groupList->Item($x);
   $groupName = $groupObj->Name();
   print $groupName,"\n";
   }
CQAdminSession::Unbuild($adminSession);
```

See Also `Count`
`Groups` property of the **`AdminSession Object`** (for VBScript)
`GetGroups` method of the **`AdminSession Object`** (for Perl)
"Adding and Removing Users in a Group" on page 857.

Histories Object

21

In Rational ClearQuest an entity may have 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

"Attachments and Histories" on page 7
History Object
HistoryField Object
HistoryFields Object

Histories Object Properties

The following list summarizes the Histories object properties:

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

Count

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

Syntax

VBScript

collection.**Count**

Perl

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

Identifier	Description
collection	A Histories collection object, representing the set of history entries in one history 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.

See Also `Item`

Histories Object Methods

The following list summarizes the Histories object methods:

Method Name	Description
`Item`	Returns the specified item in the collection.
`ItemByName`	(Perl only) Returns the specified item in the collection.

Note: For all Perl methods that map to Visual Basic Properties, see the Properties section of this object.

The following list summarizes Perl Histories object methods:

Method Name	Access	Description
`Count`	Read-only	Returns the number of items in the collection.

Item

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

VBScript

```
collection.Item(itemNum)
collection.Item(name)
```

Perl

```
$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`

History Object

22

In Rational ClearQuest an entity may have 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 Visual Basic property (**`Value`**) and one Perl method (**Get`Value`**).

See Also

"Attachments and Histories" on page 7
Histories Object
HistoryField Object
HistoryFields Object

History Object Properties

The following list summarizes the one History object property:

Property Name	Access	Description
`Value`	Read-only	Returns the String that contains information about one modification to a record, as displayed on one line of a history field.

Value

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.

Syntax

VBScript

history.**Value**

Perl

$history->**GetValue();**

Identifier	Description
History	A History object, representing one modification to a record.
Return value	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

History Object Methods

Note: For Perl methods that map to Visual Basic Properties, see the Properties section of this object.

The following list summarizes the one History object Perl method:

Method Name	Access	Description
Get`Value`	Read-only	Returns the String that contains information about one modification to a record, as displayed on one line of a history field.

HistoryField Object

23

In Rational ClearQuest an entity may have 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

"Attachments and Histories" on page 7
`HistoryFields` 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`	Read-only	Returns the unique keys of the history items in this field.
`FieldName`	Read-only	Returns the name of the history field.
`Histories`	Read-only	Returns this history field's collection of History objects.

DisplayNameHeader

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.

Syntax

VBScript

historyField.**DisplayNameHeader**

Perl

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

Identifier	Description
field	A HistoryField object, representing one field of a record.
Return value	For Visual Basic, 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 an array of strings.

See Also

FieldName

FieldName

Description

Returns the name of the history 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.

Syntax

VBScript

historyField.**FieldName**

Perl

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

Identifier	Description
field	A HistoryField object, representing one field of a record.
Return value	A String that contains the name of the field.

See Also

DisplayNameHeader

Histories

Description Returns this history field's collection of History objects. This a read-only property; the value can be viewed but not set.

Syntax

VBScript

historyField.**Histories**

Perl

$historyField->`Get`**Histories();**

Identifier	Description
historyField	A HistoryField object, representing one history field of a record.
Return value	A Histories collection object, which itself contains a set of **History Object** objects.

See Also **Histories Object**

HistoryField Object Methods

Note: For Perl methods that map to Visual Basic Properties, see the Properties section of this object.

The following list summarizes Perl HistoryField object methods:

Method Name	Access	Description
GetDisplayNameHeader	Read-only	Returns the unique keys of the history items in this field.
GetFieldName	Read-only	Returns the name of the history field.
GetHistories	Read-only	Returns this history field's collection of History objects.

HistoryFields Object

24

In Rational ClearQuest an entity may have 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`** method.

See Also

"Attachments and Histories" on page 7
History Object
Histories Object
HistoryField Object

HistoryFields Object Properties

The following list summarizes the HistoryFields object properties:

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

Count

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

Syntax

VBScript

collection.**Count**

Perl

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

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

See Also `Item`

HistoryFields Object Methods

The following list summarizes the HistoryFields object methods:

Method Name	Description
`Item`	Returns the specified item in the collection.
`ItemByName`	(Perl only) Returns the specified item in the collection.

Note: For all Perl methods that map to Visual Basic Properties, see the Properties section of this object.

The following list summarizes Perl HistoryFields object methods:

Method Name	Access	Description
`Count`	Read-only	Returns the number of items in the collection.

Item

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

VBScript

```
collection.Item(itemNum)
collection.Item(name)
```

Perl

```
$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

25

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 provides the **`AddItem`** method, which you can use to add a new item to the list. You can use the **`AddItems`** method to add a list of values.

The HookChoices object is stored in a variable called `choices` (for hooks, not global scripts) and you can only access it by that name.

Note: The HookChoices object is for Visual Basic only. For Perl, use a reference to an array of strings to return a choice list. See "Examples of Hooks and Scripts" on page 809.

See Also

`Entity Object`
FieldInfo Object
Examples of Hooks and Scripts

HookChoices Object Methods

The following list summarizes the HookChoices object methods:

Method Name	Description
`AddItem`	Adds a new item to the list of choices created by a CHOICE_LIST hook.
`AddItems`	Adds a list of new items to the list of choices created by a CHOICE_LIST hook.
`Sort`	Sorts the entries in the choice list.

AddItem

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 (for hooks, not global scripts); 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 (or, you can use the **`AddItems`** method). 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.

Syntax

VBScript

`choices.`**`AddItem`**`(`*`newChoice`*`)`

Identifier	Description
choices	A special HookChoices object.
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
***Creating a Dependent Choice List* on page 854**
`AddItems`

AddItems

Description

Adds a list of new items to the list of choices created by a CHOICE_LIST hook. You can use this method to add multiple items, instead of calling **`AddItem`** multiple times.

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(for hooks, not global scripts); 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 can call this method to build up a list of choices for the user. The object contains no items when you first access it. Order the items you list in the array in the order in which you want them to appear, because the list is not automatically sorted.

There is no RemoveItem method. Duplicate items are not automatically removed, but empty values are.

Syntax

VBScript

`choices.`**`AddItems`**`(`*`items`*`)`

Identifier	Description
choices	A special HookChoices object.
items	A VARIANT which contains an array of Strings, each of which is a list item in the choice list.
Return value	None.

Example

VBScript

```
Dim platform(3)
platform(0) = "Professional"
platform(1) = "Professional SP1"
platform(2) = "Server"
choices.AddItems platform
```

See Also

HookChoices Object
Examples of Hooks and Scripts

Sort

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.

Syntax

VBScript

`choices.`**Sort** *`sortAscending`*

Identifier	Description
choices	A special HookChoices object.
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`

Link Object

26

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 to be based on the same record type.

The methods of linking allow you to retrieve the:

- Parent and child record objects that are linked together.
- ID strings for the parent and child.
- EntityDef that is the template for the parent or child.
- 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` of the **Entity Object**
`GetDuplicates` of the **Entity Object**
`HasDuplicates` of the **Entity Object**
`IsDuplicate` of the **Entity Object**
`GetChildEntity` of the **Entity Object**

Link Object Methods

The following list summarizes the Link object methods:

Method Name	Description
`GetChildEntity`	Returns the Entity object that is the child (duplicate) in a pair of linked Entity objects.
`GetChildEntityDef`	Returns the **EntityDef Object** that is the template for the child (duplicate) in a pair of linked Entity objects.
`GetChildEntityDefName`	Returns the name of the EntityDef object that is the template for the child (duplicate) Entity object.
`GetChildEntityID`	Returns the ID String of the Entity object that is the child (duplicate) in a pair of linked Entity objects.
`GetParentEntity`	Returns the record that is the parent (original) in a pair of linked Entity objects.
`GetParentEntityDef`	Returns the **EntityDef Object** that is the template for the parent (original) in a pair of linked Entity objects.
`GetParentEntityDefName`	Returns the name of the EntityDef object that is the template for the parent (original) Entity object.
`GetParentEntityID`	Returns the ID String of the Entity object that is the parent (original) in a pair of linked Entity objects.

GetChildEntity

Description Returns the Entity object that is the child (duplicate) in a pair of linked Entity objects.

Syntax

VBScript

link.**GetChildEntity**

Perl

$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).

Examples

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
```

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` of the **Entity Object**
`GetDuplicates` of the **Entity Object**
`IsOriginal` of the **Entity Object**
"Updating Duplicate Records to Match the Parent Record" on page 820

GetChildEntityDef

Description Returns the **EntityDef Object** that is the template for the child (duplicate) in a pair of linked Entity objects.

Syntax

VBScript

link.**GetChildEntityDef**

Perl

$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**
GetEntityDef of the **Session Object**

GetChildEntityDefName

Description Returns the name of the **EntityDef Object** that is the template for the child (duplicate) Entity object.

Syntax

VBScript

link.**GetChildEntityDefName**

Perl

$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
GetEntityDefName of the Entity object
Entity Object

GetChildEntityID

Description Returns the ID String of the Entity object that is the child (duplicate) in a pair of linked Entity objects.

Syntax

VBScript

link.**GetChildEntityID**

Perl

$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** of Entity.

See Also

GetParentEntityID
GetDisplayName of the **Entity Object**
GetDuplicates of the **Entity Object**

GetParentEntity

Description Returns the record that is the parent (original) in a pair of linked Entity objects.

Syntax

VBScript

link.**GetParentEntity**

Perl

$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`** of the **Entity Object**
`IsDuplicate` of the **Entity Object**

GetParentEntityDef

Description Returns the **EntityDef Object** that is the template for the parent (original) in a pair of linked Entity objects.

Syntax

VBScript

link.**GetParentEntityDef**

Perl

$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**
GetEntityDef of the **Session Object**

GetParentEntityDefName

Description Returns the name of the **EntityDef Object** that is the template for the parent (original) Entity object.

Syntax

VBScript

link.**GetParentEntityDefName**

Perl

$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**
GetEntityDefName of the **Entity Object**

GetParentEntityID

Description Returns the ID String of the Entity object that is the parent (original) in a pair of linked Entity objects.

Syntax

VBScript

link.**GetParentEntityID**

Perl

$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** of Entity.

See Also **GetChildEntityID**
GetDisplayName of the **Entity Object**

Links Object

27

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`** method. Use the **`Item`** method to retrieve items from the collection.

Note: Links object and its methods are only applicable for usage with Perl script. It is not available in the COM API.

See Also **Link Object**

Links Object Methods

The following list summarizes the Links object methods:

Method Name	Description
`Add`	Adds an Attachment object to the collection.
`Count`	Returns the number of items in the collection.
`Item`	Returns the specified item in the collection.
`ItemByName`	Returns the specified item in the collection.

Add

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.

Syntax

Perl

```
$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
Item

Count

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

Syntax

Perl

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

Identifier	Description
links	A Links collection object.
Return value	A Long indicating the number of items in the collection object. This collection always contains at least one item.

See Also `Item`
`Add`

Item

Description Returns the specified item in the Links collection.

Syntax

Perl

```
$links->Item(itemNum);
```

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.
Return value	The Link object at the specified location in the collection.

See Also **Count**
Add

ItemByName

Description Returns the specified item in the Links collection.

Syntax

Perl

```
$links->ItemByName(name);
```

Identifier	Description
links	A Links collection object.
name	A String that serves as a key into the collection. This string corresponds to the **`GetChildEntityDefName`** of the desired Links.
Return value	The Link object at the specified location in the collection.

See Also **`Count`**
`Add`

MailMsg Object

28

A MailMsg object represents an e-mail message that you can send to your users.

Note: The MailMsg object and its methods is supported for the COM API only.

The main purpose for the MailMsg 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.

Note: For the email service to function correctly, each ClearQuest client must set up its email options.

For VBScript, you create a new OleMailMsg object using the CreateObject method as follows:

```
Dim mailmsg
Set mailmsg = CreateObject("PAINET.MAILMSG")
```

When you have a MailMsg 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	Add the e-mail address of a blind carbon-copy recipient to the mail message.
AddCc	Add the e-mail address of a carbon-copy recipient to the mail message.
AddTo	Add the e-mail address of a primary recipient to the mail message.
ClearAll	Resets the contents of the mail message object.
Deliver	Delivers the mail message.
MoreBody	Appends additional body text to the mail message.
SetBody	Sets the body text of the mail message.
SetFrom	Sets the return address of the mail message.
SetSubject	Sets the subject line of the e-mail message.

AddBcc

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.

Syntax

VBScript

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
AddTo
ClearAll
SetFrom
SetSubject

AddCc

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.

Syntax

VBScript

OleMailMsg.**AddCc** *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.

Examples

VBScript

```
Dim OleMailMsg
Set OleMailMsg = CreateObject("PAINET.MAILMSG")
CCx = ""
OleMailMsg.AddCc(CCx)
```

See Also

AddBcc
AddTo
ClearAll
SetFrom
SetSubject

AddTo

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.

Syntax

VBScript

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.

Examples

VBScript

```
Dim OleMailMsg
Set OleMailMsg = CreateObject("PAINET.MAILMSG")
Tox = "admin@rational.com"
OleMailMsg.AddTo(Tox)
```

See Also

AddBcc
AddCc
ClearAll
SetFrom
SetSubject

ClearAll

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.

Syntax

VBScript

OleMailMsg.**ClearAll**

Identifier	Description
OleMailMsg	An OleMailMsg object, representing the mail message to be sent.
Return value	None.

See Also

AddBcc
AddCc
AddTo
MoreBody
SetBody
SetFrom
SetSubject

Deliver

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.

Syntax

VBScript

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.

Examples

VBScript

```
Dim OleMailMsg
Set OleMailMsg = CreateObject("PAINET.MAILMSG")
' ...
OleMailMsg.Deliver
```

See Also

AddBcc
AddCc
AddTo
ClearAll
MoreBody
SetBody
SetFrom
SetSubject

MoreBody

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`**. 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.

Syntax

VBScript

`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`
`SetBody`

SetBody

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

Syntax

VBScript

OleMailMsg.**SetBody** *bodyText*

Identifier	Description
OleMailMsg	An OleMailMsg object, representing the mail message to be sent.
Return value	None.

Examples

VBScript

```
Dim OleMailMsg
Set OleMailMsg = CreateObject("PAINET.MAILMSG")
bodyx = "TEST MESSAGE"
OleMailMsg.SetBody(bodyx)
```

See Also

ClearAll
MoreBody

SetFrom

Description

Sets the return address of the mail message.

If you do not call this method, Rational 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.

Syntax

VBScript

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.

Examples

VBScript

```
Dim OleMailMsg
Set OleMailMsg = CreateObject("PAINET.MAILMSG")
Fromx = "admin@rational.com"
OleMailMsg.SetFrom(Fromx)
```

See Also

AddBcc
AddCc
AddTo
ClearAll
SetSubject

SetSubject

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.

Syntax

VBScript

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.

Examples

VBScript

```
Dim OleMailMsg
Set OleMailMsg = CreateObject("PAINET.MAILMSG")
Subjectx = "NSPW Project (ClearQuest) -- "
OleMailMsg.SetSubject(Subjectx)
```

See Also

AddBcc
AddCc
AddTo
ClearAll
SetFrom

PackageRev Object

29

A PackageRev object contains information about a particular version of a package. Each package revision identifies a specific version of a package. You use package revisions to modify a schema by applying a PackageRev to a SchemaRev.

See Also `PackageRevs Object`

PackageRev Object Properties

The following list summarizes the PackageRev object properties:

Property Name	Description
PackageName	Returns the name of the package this revision belongs to.
RevString	Returns the revision string of this package rev.

PackageName

Description

Returns the name of the package this revision belongs to.

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

Syntax

VBScript

packageRev.**PackageName**

Perl

$packageRev->**GetPackageName**();

Identifier	Description
packageRev	A PackageRev object.
Return value	Returns a String containing the name of the package.

See Also

GetEnabledEntityDefs of the **Session Object**

GetEnabledPackageRevs of the **Session Object**

Schema Object

RevString

Description

Returns the revision string of this package revision.

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

Syntax

VBScript

packageRev.**RevString**

Perl

*$packageRev->***GetRevString**();

Identifier	Description
packageRev	A PackageRev object.
Return value	A String indicating the revision string associated with this package revision.

See Also

GetEnabledEntityDefs of the **Session Object**

GetEnabledPackageRevs of the **Session Object**

Schema Object

PackageRev Object Methods

Note: For Perl methods that map to Visual Basic Properties, see the Properties section of this object.

Perl Get and Set Methods that map to Visual Basic properties:

Method Name	Access	Description
GetPackageName	Read-only	Returns the name of the package this revision belongs to.
GetRevString	Read-only	Returns the revision string of this package rev.

PackageRevs Object

30

A PackageRevs object is a collection object for PackageRev objects.

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

See Also

`PackageRev Object`

`GetEnabledEntityDefs` of the **`Session Object`**

`GetEnabledPackageRevs` of the **`Session Object`**

Schema Object

PackageRevs Object Properties

The following list summarizes the PackageRevs object properties:

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

Count

Description Returns the number of items in the collection. This is a read-only property; it can be viewed but not set.

Syntax

VBScript

collection.**Count**

Perl

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

Identifier	Description
collection	A PackageRevs collection object, representing the set of package 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`

PackageRevs Object Methods

The following list summarizes the PackageRevs object methods:

Method	Description
`Item`	Returns the item at the specified index in the collection.
`ItemByName`	(Perl only) Returns the specified item in the collection.

Note: For Perl methods that map to Visual Basic Properties, see the Properties section of this object.

The following list summarizes additional Perl PackageRevs object methods:

Method Name	Access	Description
`Count`	Read-only	Returns the number of items in the collection.

Item

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

VBScript

```
collection.Item(itemNum)
collection.Item(name)
```

Perl

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

Identifier	Description
collection	A PackageRevs collection object, representing the set of package 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 PackageRev object.
Return value	The PackageRev object at the specified location in the collection.

See Also

`Count`

ProductInfo Object

31

A CQProductInfo object provides ClearQuest product information, such as product version, license version and company information.

Note: The CQProductInfo object and its methods are for usage with Perl only.

The CQProductInfo methods return information to identify product information and its current build. The methods of CQProductInfo allow you to retrieve information such as:

- Product versions
- Company information
- Licensing information

To create a CQProductInfo object, you can use the CreateProductInfo method of a ClearQuest object. For example:

```
use CQPerlExt;
my $cqobject = CQClearQuest::Build();
my $prodinfo = $cqobject->CreateProductInfo();
print $prodinfo->GetProductVersion(),"\n";
CQClearQuest::Unbuild($cqobject);
```

You can also CQProductInfo methods directly, from Perl, without creating a ClearQuest session.

```
use CQPerlExt;
print CQProdInfo::GetProductVersion(),"\n";
```

See Also `ClearQuest Object`

ProductInfo Object Methods

The following list summarizes the CQProductInfo object methods:

Method name	Description
GetBuildNumber	Returns the product build number.
GetCompanyEmailAddress	Returns the company email address of the company for the current locale.
GetCompanyFullName	Returns the full name of the company in the current locale.
GetCompanyName	Returns the name of the company in the current locale.
GetCompanyWebAddress	Returns the web address of the company for the current locale.
GetDefaultDbSetName	Returns the default database set name.
GetFullProductVersion	Returns the full product version.
GetLicenseFeature	Returns the FlexLM feature name used to get a license.
GetLicenseVersion	Returns the version of the FlexLM feature that is used to get a license.
GetObjectModelVersionMajor	Returns a version number for the API itself.
GetObjectModelVersionMinor	Returns a version number for the API itself.
GetPatchVersion	Returns the current patch version of the product.
GetProductVersion	Returns the current version of the product.
GetStageLabel	Get the ClearCase label used to uniquely identify each build.
GetSuiteProductVersion	Returns the Suite product version.
GetWebLicenseVersion	Returns the version of the FlexLM feature that is used to get a web license.

GetBuildNumber

Description Returns the product build number. This number is used to uniquely identify each build.

Syntax

Perl

```
$prodinfo->GetBuildNumber();
```

Identifier	Description
prodinfo	A CQProductInfo object.
Return value	A String containing the product build number (for example, 00430).

Example

Perl

```
use CQPerlExt;

my $cqobject = CQClearQuest::Build();
my $prodinfo = $cqobject->CreateProductInfo();
print $prodinfo->GetBuildNumber(),"\n";
CQClearQuest::Unbuild($cqobject);
```

See Also **GetStageLabel**

GetCompanyEmailAddress

Description Returns the company email address of the company for the current locale.

Syntax

Perl

*$prodinfo->***GetCompanyEmailAddress**();

Identifier	Description
prodinfo	A CQProductInfo object.
Return value	A String containing the company email address (for example, Rational.com.)

Example

Perl

```
use CQPerlExt;

my $cqobject = CQClearQuest::Build();
my $prodinfo = $cqobject->CreateProductInfo();
print $prodinfo->GetCompanyEmailAddress(),"\n";
CQClearQuest::Unbuild($cqobject);
```

See Also

GetCompanyFullName
GetCompanyName
GetCompanyWebAddress

GetCompanyFullName

Description Returns the full name of the company in the current locale. "Rational Software Corporation" in English.

Syntax

Perl

*$prodinfo->***GetCompanyFullName**();

Identifier	Description
prodinfo	A CQProductInfo object.
Return value	A String containing the company fullname (for example, Ratioanl Software Corporation).

Example

Perl

```
use CQPerlExt;


my $cqobject = CQClearQuest::Build();
my $prodinfo = $cqobject->CreateProductInfo();
print $prodinfo->GetCompanyFullName(),"\n";
CQClearQuest::Unbuild($cqobject);
```

See Also

GetCompanyName
GetCompanyEmailAddress
GetCompanyWebAddress

GetCompanyName

Description Returns the name of the company in the current locale. "Rational Software" in English.

Syntax

Perl

*$prodinfo->***GetCompanyName**();

Identifier	Description
prodinfo	A CQProductInfo object.
Return value	A String containing the compnay name (for example, Rational Software).

Example

Perl

```
use CQPerlExt;

my $cqobject = CQClearQuest::Build();
my $prodinfo = $cqobject->CreateProductInfo();
print $prodinfo->GetCompanyName(),"\n";
CQClearQuest::Unbuild($cqobject);
```

See Also

GetCompanyFullName
GetCompanyEmailAddress
GetCompanyWebAddress

GetCompanyWebAddress

Description Returns the web address of the company for the current locale.

Syntax

Perl

*$prodinfo->***GetCompanyWebAddress**();

Identifier	Description
prodinfo	A CQProductInfo object.
Return value	A String containing the company web address (for example, www.Rational.com).

Example

Perl

```
use CQPerlExt;

my $cqobject = CQClearQuest::Build();
my $prodinfo = $cqobject->CreateProductInfo();
print $prodinfo->GetCompanyWebAddress(),"\n";
CQClearQuest::Unbuild($cqobject);
```

See Also

GetCompanyFullName
GetCompanyName
GetCompanyEmailAddress

GetDefaultDbSetName

Description Returns the default database set name.

Syntax

Perl

*$prodinfo->***GetDefaultDbSetName**();

Identifier	Description
prodinfo	A CQProductInfo object.
Return value	A String containing the default database set name (for example, Default).

Example

Perl

```
use CQPerlExt;


my $cqobject = CQClearQuest::Build();
my $prodinfo = $cqobject->CreateProductInfo();
print $prodinfo->GetDefaultDbSetName(),"\n";
CQClearQuest::Unbuild($cqobject);
```

See Also **GetSessionDatabase** of the **Session Object**

GetFullProductVersion

Description Returns the full product version. This method is equivalent to the **`GetProductVersion`** and **`GetPatchVersion`** return values separated by a space.

Syntax

Perl

`$prodinfo->``GetFullProductVersion()`;

Identifier	Description
prodinfo	A CQProductInfo object.
Return value	A String containing the full product version.

Example

Perl

```
use CQPerlExt;

my $cqobject = CQClearQuest::Build();
my $prodinfo = $cqobject->CreateProductInfo();
print $prodinfo->GetFullProductVersion(),"\n";
CQClearQuest::Unbuild($cqobject);
```

See Also **`GetProductVersion`**
`GetPatchVersion`

GetLicenseFeature

Description Returns the FlexLM feature name used to get a license.

Syntax

Perl

*$prodinfo->***GetLicenseFeature**();

Identifier	Description
prodinfo	A CQProductInfo object.
Return value	A String containing the license feature (for example, ClearQuest).

Example

Perl

```
use CQPerlExt;

my $cqobject = CQClearQuest::Build();
my $prodinfo = $cqobject->CreateProductInfo();
print $prodinfo->GetLicenseFeature(),"\n";
CQClearQuest::Unbuild($cqobject);
```

See Also **GetLicenseVersion**
GetWebLicenseVersion

GetLicenseVersion

Description Returns the version of the FlexLM feature that is used to get a license.

Syntax

Perl

*$prodinfo->***GetLicenseVersion**();

Identifier	Description
prodinfo	A CQProductInfo object.
Return value	A String containing the license version (for example, 1.1).

Example

Perl

```
use CQPerlExt;

my $cqobject = CQClearQuest::Build();
my $prodinfo = $cqobject->CreateProductInfo();
print $prodinfo->GetLicenseVersion(),"\n";
CQClearQuest::Unbuild($cqobject);
```

See Also **GetLicenseFeature**
GetWebLicenseVersion

GetObjectModelVersionMajor

Description Returns a version number for the API itself. The major version number increases whenever an incompatible change is made in the API. The major number should be checked as part of an initial interaction with ClearQuest to be certain that the API is compatible with what the caller expects.

Syntax

Perl

*$prodinfo->***GetObjectModelVersionMajor**();

Identifier	Description
prodinfo	A CQProductInfo object.
Return value	A Long containing the object model version (for example, 1).

Example

Perl

```
use CQPerlExt;

my $cqobject = CQClearQuest::Build();
my $prodinfo = $cqobject->CreateProductInfo();
print $prodinfo->GetObjectModelVersionMajor(),"\n";
CQClearQuest::Unbuild($cqobject);
```

See Also **GetObjectModelVersionMinor**

GetObjectModelVersionMinor

Description Returns a version number for the API itself. The minor number may increase when other upward-compatible changes are made.

Syntax

Perl

*$prodinfo->***GetObjectModelVersionMinor**();

Identifier	Description
prodinfo	A CQProductInfo object.
Return value	A Long containing the object model version (for example, 1).

Example

Perl

```
use CQPerlExt;

my $cqobject = CQClearQuest::Build();
my $prodinfo = $cqobject->CreateProductInfo();
print $prodinfo->GetObjectModelVersionMinor(),"\n";
CQClearQuest::Unbuild($cqobject);
```

See Also **GetObjectModelVersionMajor**

GetPatchVersion

Description Returns the current patch version of the product.

Syntax

Perl

$prodinfo->**GetPatchVersion**();

Identifier	Description
prodinfo	A CQProductInfo object.
Return value	A String containing the patch version, if any.

Example

Perl

```
use CQPerlExt;

my $cqobject = CQClearQuest::Build();
my $prodinfo = $cqobject->CreateProductInfo();
print $prodinfo->GetPatchVersion(),"\n";
CQClearQuest::Unbuild($cqobject);
```

See Also **GetBuildNumber**

GetProductVersion

Description Returns the current version of the product.

Syntax

Perl

```
$prodinfo->GetProductVersion();
```

Identifier	Description
prodinfo	A CQProductInfo object.
Return value	A String containing the product version (for example, 2002.05.00).

Example

Perl

```
use CQPerlExt;

my $cqobject = CQClearQuest::Build();
my $prodinfo = $cqobject->CreateProductInfo();
print $prodinfo->GetProductVersion(),"\n";
CQClearQuest::Unbuild($cqobject);
```

See Also **GetFullProductVersion**
GetSuiteProductVersion

GetStageLabel

Description Returns the ClearCase label used to uniquely identify each build.

Syntax

Perl

*$prodinfo->***GetStageLabel**();

Identifier	Description
prodinfo	A CQProductInfo object.
Return value	A String containing the stage label of the build (for example, CQ.OM.200110291206).

Example

Perl

```
use CQPerlExt;

my $cqobject = CQClearQuest::Build();
my $prodinfo = $cqobject->CreateProductInfo();
print $prodinfo->GetStageLabel(),"\n";
CQClearQuest::Unbuild($cqobject);
```

See Also **GetSuiteProductVersion**

GetSuiteProductVersion

Description Returns the Suite product version.

Syntax

Perl

*$prodinfo->***GetSuiteProductVersion**();

Identifier	Description
prodinfo	A CQProductInfo object.
Return value	A String containing the Suite product version (for example, 2002.05.00).

Example

Perl

```
use CQPerlExt;

my $cqobject = CQClearQuest::Build();
my $prodinfo = $cqobject->CreateProductInfo();
print $prodinfo->GetSuiteProductVersion(),"\n";
CQClearQuest::Unbuild($cqobject);
```

See Also **GetProductVersion**
GetLicenseVersion

GetWebLicenseVersion

Description Returns the version of the FlexLM feature that is used to get a web license.

Syntax

Perl

*$prodinfo->***GetWebLicenseVersion**();

Identifier	Description
prodinfo	A CQProductInfo object.
Return value	A String containing the web license version (for example, 1.1).

Example

Perl

```
use CQPerlExt;

my $cqobject = CQClearQuest::Build();
my $prodinfo = $cqobject->CreateProductInfo();
print $prodinfo->GetWebLicenseVersion(),"\n";
CQClearQuest::Unbuild($cqobject);
```

See Also **GetLicenseVersion**

QueryDef Object

32

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 **`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.

For example, in the following statement:

Select * from <some defect> where (….) AND1 ((…) OR2 ((…) AND3((…) OR4 (…))))

the root operator is AND (AND1). Its next level sub-node operator is an OR (OR2). The complete hierarchy is:

```
AND1 (AND1 is the root operator)
\
   OR2 (OR2 is suboperator of AND1)
   \
      AND3 (AND3 is suboperator of OR2)
      \
         OR4 (OR4 is suboperator of AND3)
```

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 "Working with Queries" on page 15
`BuildFilterOperator`
`BuildQuery` of the Session object
ResultSet Object
Session Object
"Building Queries for Defects and Users" on page 817

QueryDef Object Properties

The following list summarizes the QueryDef object properties:

Property Name	Access	Description
IsAggregated	Read-only	Returns a Boolean indicating whether any fields of the query are aggregated.
IsDirty	Read-only	Returns a Boolean indicating whether the query has changed.
IsMultiType	Read-only	Returns a Boolean indicating whether the QueryDef object is multitype.
IsSQLGenerated	Read-only	Returns a Bool indicating whether any fields of the query are SQL generated.
Name	Read/Write	Sets or returns the name associated with the query.
QueryFieldDefs	Read-only	Returns the collection of QueryFieldDef objects included in the query.
QueryType	Read-only	Returns an Integer indicating list, report, or chart.
SQL	Read/Write	Sets or returns the SQL string associated with the query.

IsAggregated

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.

Syntax

VBScript

querydef.**IsAggregated**

Perl

$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
AggregateFunction of the **QueryFieldDef Object**

IsDirty

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.

Syntax

VBScript

querydef.**IsDirty**

Perl

*$querydef->***GetIsDirty();**

Identifier	Description
querydef	A QueryDef object.
Return value	True if the query has changed, otherwise False.

See Also

`Save`

IsMultiType

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 **`GetEntityDefFamilyName`** and **`GetEntityDefFamilyNames`**.

Syntax

VBScript

`querydef``.`**IsMultiType**

Perl

$*`querydef`***->GetIsMultiType()**;

Identifier	Description
querydef	A QueryDef object.
Return value	True if the object is multitype.

See Also

`GetEntityDefFamilyName`
`GetEntityDefFamilyNames`

IsSQLGenerated

Description Returns a Bool indicating whether any fields of the query are SQL generated.

Syntax

VBScript

`querydef`.**IsSQLGenerated**

Perl

$*querydef***->GetIsSQLGenerated()**;

Identifier	Description
querydef	A QueryDef object.
Return value	A Bool containing the value True if the any fields in the query are SQL generated, otherwise False.

See Also `SQL`

Name

Description Sets or returns the name associated with the query.

Syntax

VBScript

querydef.**Name**

querydef.**Name** *newName*

Perl

```
$querydef->GetName();
$querydef->SetName(newName);
```

Identifier	Description
querydef	A QueryDef object.
newName	A String containing the new name of the query.
Return value	A String containing the name of the query.

See Also `Save`

QueryFieldDefs

Description Returns the collection of QueryFieldDef objects included in the QueryDef. You include fields in a query using the **`BuildField`** method.

The new QueryFieldDefs object that this method returns provides alternate methods to do what you can do with the **`BuildField`** or **`GetFieldByPosition`** methods.

Syntax

VBScript

`querydef.`**QueryFieldDefs**

Perl

$*querydef***->GetQueryFieldDefs()**;

Identifier	Description
querydef	A QueryDef object.
Return value	A QueryFieldDefs object.

See Also **`BuildField`**

QueryType

Description Returns an integer indicating whether the saved query has the property of being a list, a report, or a chart.

Syntax

VBScript

querydef.**QueryType**

Perl

$querydef->**Get**QueryType();

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

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.

Syntax

VBScript

querydef.**SQL**

```
set workspace = session.GetWorkSpace
set querydef = workspace.GetQueryDef queryName
querydef.SQL string_of_SQL_statements
```

Perl

$querydef->**GetSQL**();

```
$workspace = $session->GetWorkSpace();
$querydef = $workspace->GetQueryDef(queryName);
$querydef->SetSQL(string_of_SQL_statements);
```

Identifier	Description
querydef	A QueryDef object.
string_of_SQL_statements	A String containing the individual SQL statements.
Return value	For the Get, returns a String containing the SQL that will be executed when the query is run. For the Set, no return value. Returns an exception if user doesn't have SQL writer privilege.

Examples

VBScript

```
set session = GetSession
set workspace = session.GetWorkSpace
'Get the QueryDef by supplying a query name
set querydef = workspace.GetQueryDef "Public Queries\Defects"
'Provide a string of SQL statements
querydef.SQL "select distinct T1.dbid,T1.id,T1.headline from Defect
T1,statedef T2 where T1.state = T2.id and (T1.dbid <> 0 and (T2.name =
'Submitted'))"
```

Perl

```
$workspace = $session->GetWorkSpace();

$querydef = $workspace->GetQueryDef(queryName);

$querydef->SetSQL("select distinct T1.dbid,T1.id,T1.headline from Defect
T1,statedef T2 where T1.state = T2.id and (T1.dbid <> 0 and (T2.name =
'Submitted'))");
```

See Also

***BuildSQLQuery* of the Session object**
Session Object

QueryDef Object Methods

Method Name	Description
BuildField	Selects a field to include in the query's search results.
BuildFilterOperator	Creates the top-level **QueryFilterNode Object** for the query.
BuildUniqueKeyField	Builds and returns a unique key query field.
CreateTopNode	Creates a new root filter node, automatically attaches the old top node as its child.
GetFieldByPosition	Returns a QueryFieldDef object with the specified position.
GetPrimaryEntityDefName	Returns the primary EntityDef name for the QueryDef.
IsFieldLegalForQuery	Returns a Bool indicating whether a field is legal to be included in a query.
Save	Saves the query to the specified file.

Note: For all Perl Get and Set methods that map to Visual Basic Properties, see the Properties section of this object.

Additional Perl Methods that map to Visual Basic properties:

Method Name	Description
CreateTopNode	Creates a new root filter node, automatically attaches the old top node as its child.
GetIsSQLGenerated	Returns a Bool indicating whether any fields of the query are SQL generated.
GetName	Returns the name associated with the query.
GetQueryFieldDefs	Returns the collection of QueryFieldDef objects included in the query.
GetSQL	Returns the SQL string associated with the query.
IsAggregated	Returns a Boolean indicating whether any fields of the query are aggregated.
IsDirty	Returns a Boolean indicating whether the query has changed.
IsMultiType	Returns a Boolean indicating whether the QueryDef object is multitype.
GetQueryType	Returns an Integer indicating list, report, or chart.
SetName	Sets the name associated with the query.
SetSQL	Sets the SQL string associated with the query.

BuildField

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).

Syntax

VBScript

querydef.**BuildField** *field_name*

Perl

$*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.

Example

VBScript

```
' create a query for defect where id = SAMPL00000001
Dim session

Set session = CreateObject("CLEARQUEST.SESSION")
session.UserLogon "admin", "", "SAMPL", AD_PRIVATE_SESSION, ""

 Set QueryDef = session.BuildQuery("defect")
 QueryDef.BuildField ("headline")
 QueryDef.BuildField ("id")

 Set filternode1 = QueryDef.BuildFilterOperator (AD_BOOL_OP_AND)
 filterNode1.BuildFilter "id", AD_COMP_OP_EQ, "SAMPL00000001"
 Set rsltset = session.BuildResultSet(QueryDef)
 rsltset.Execute
 Status = rsltset.MoveNext
```

Perl

```
$queryDef = $CQSession->BuildQuery("Defect");
@dbfields = ("ID","State","Headline");
foreach $field (@dbfields) {
      $queryDef->BuildField($field);
      }
```

See Also

`QueryFieldDefs`
`BuildFilterOperator`
`BuildQuery` of the Session object
`GetFieldDefNames` of the EntityDef object
EntityDef Object
ResultSet Object
Session Object
"Building Queries for Defects and Users" on page 817
"Getting a List of Defects and Modifying a Record" on page 847
`GetLocalFieldPathNames` of the **`EntityDef Object`**
`FieldPathName` of the **`QueryFieldDef Object`**

BuildFilterOperator

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.

Syntax

VBScript

querydef.**BuildFilterOperator** *bool_operator*

Perl

$*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

VBScript

```
submit_date < 01/03/2001 AND
(submitter = jjones OR submitter = clopez OR submitter = kwong)
```

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:

```
set myQueryDef = sessionObj.BuildQuery("Defect")
myQueryDef.BuildField("id")
myQueryDef.BuildField("headline")
set 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, "2001-01-03"

set 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.
```

```
set 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.

Perl

```
@owner = ("jsmith");
@state = ("closed");
$queryDef = $CQsession->BuildQuery("defect");
@dbfields = ("ID","State","Headline");
foreach $field (@dbfields) {
      $queryDef->BuildField($field);
      }
$operator = $queryDef->BuildFilterOperator($CQPerlExt::CQ_BOOL_OP_AND);
$operator->BuildFilter("Owner", $CQPerlExt::CQ_COMP_OP_EQ,\@owner);
$operator->BuildFilter("State", $CQPerlExt::CQ_COMP_OP_NOT_IN, \@state);
```

See Also

`BuildField`

`BuildFilter` of the QueryFilterNode object
`BuildFilterOperator` of the QueryFilterNode object
`BoolOp Constants`
QueryFilterNode Object
"Building Queries for Defects and Users" on page 817

BuildUniqueKeyField

Description Builds and returns a unique key query field.

Syntax

VBScript

`querydef.`**BuildUniqueKeyField**

Perl

$*querydef***->BuildUniqueKeyField()**;

Identifier	Description
querydef	A QueryDef object.
Return value	Returns a QueryFieldDef object.

See Also **`AddUniqueKey`** of the **`QueryFieldDefs Object`**

CreateTopNode

Description Creates a new root filter node, automatically attaches the old top node as its child.

Syntax

VBScript

querydef.**CreateTopNode** *bool_op*

Perl

*$querydef->***CreateTopNode***(bool_op);*

Identifier	Description
querydef	A QueryDef object.
bool_op	A Long whose value is one of the **BoolOp Constants**.
Return value	The new QueryFilterNode object.

See Also **BoolOp Constants**

QueryFilterNode Object

GetFieldByPosition

Description Returns a QueryFieldDef object with the specified position. A QueryDef contains a list of QueryFieldDefs.

Syntax

VBScript

`querydef.`**GetFieldByPosition** *position*

Perl

$*querydef***->GetFieldByPosition(***position***);**

Identifier	Description
querydef	A QueryDef object.
position	A Long containing the position of the field (QueryFieldDef) in the list of fields (QueryFieldDefs).
Return value	Returns a QueryFieldDef object.

See Also

QueryFieldDefs
Name
Item of the **QueryFieldDefs Object**

GetPrimaryEntityDefName

Description

Returns the primary EntityDef name for the QueryDef. This is the name of the record type for the query.

When a QueryDef is created with the **`BuildQuery`** method of the **`Session Object`**, you provide a primary EntityDef (record type) name.

Syntax

VBScript

`querydef`.**GetPrimaryEntityDefName**

Perl

$*`querydef`***->GetPrimaryEntityDefName()**;

Identifier	Description
querydef	A QueryDef object.
Return value	A String containing the name of the record type (EntityDef) for the query.

See Also

`Name`
`EntityDef Object`
`BuildQuery` of the **`Session Object`**

IsFieldLegalForQuery

Description Returns a Bool indicating whether a field is legal to be included in a query. By default , this is set to True.

Syntax

VBScript

`querydef.`**IsFieldLegalForQuery** *`fieldName`*

Perl

$*`querydef`***->IsFieldLegalForQuery(***`fieldName`***);**

Identifier	Description
querydef	A QueryDef object.
fieldName	A String containing the name of the field.
Return value	A Bool containing the value True if the field is legal to be included in a query, otherwise False. By default, this is set to True.

See Also **`Name`**
`IsLegalForFilter` of the **`QueryFieldDef Object`**

Save

Description Saves the query to the specified file.

Syntax

VBScript

```
querydef.Save fileName
```

Perl

```
$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**

QueryFieldDef Object

33

QueryFieldDef objects represent the fields you include in a QueryDef. Each QueryFieldDef object is a field in a QueryDef, each of which was created with the QueryDef.BuildField method.

The QueryFieldDef object includes methods for setting the sort order of a query.

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`**).

See Also

"Working with Queries" on page 15
QueryDef Object
QueryFilterNode Object
`BuildQuery` of the Session object
ResultSet Object
Session Object
"Building Queries for Defects and Users" on page 817
"Sorting a Result Set" on page 848

QueryFieldDef Object Properties

The following list summarizes the QueryFieldDef object properties:

Property Name	Access	Description
AggregateFunction	Read/Write	Sets or returns a SQL aggregate function for the field.
ChoiceList	Read-only	Returns the items in a choice list for a QueryFieldDef.
DataType	Read-only	Returns the underlying datatype of the QueryFieldDef object.
Description	Read-only	Returns the description of the field from the schema.
FieldPathName	Read/Write	Sets or returns the name of the field from the schema.
FieldType	Read-only	Returns the field type of the QueryFieldDef.
Function	Read/Write	Set or returns a function for the QueryFieldDef .
IsGroupBy	Read/Write	Sets or returns whether the field is in a group-by clause.
IsLegalForFilter	Read-only	Returns whether you can use the field in a query filter.
IsShown	Read/Write	Enables you to sort by a field but not display the field. Sets or returns whether you want to show the field or not.
Label	Read/Write	Sets or returns the column label for a field in a ResultSet.
SortOrder	Read/Write	Sets or returns the numeric order of precedence for the field, when there is more than one sort key.
SortType	Read/Write	Sets or returns the sort type for the field.

AggregateFunction

Description

Sets or returns a SQL aggregate function for the field. When setting an aggregate function, the function you select must make sense for the field type.

For example, use DB_AGGR_COUNT to return the count of RECORDS that have some value in that field. You would typically choose dbid as the field to count, since you typically mean to count all records. The other functions return either the minimum value, maximum value, the average value, or the sum of a field. Sum and Average only work for numeric fields.

For example, "select count(dbid) from defect" returns the total number of defects in the database.

You can also specify a group-by field, and get the statistic broken down by some other field. For example, "select count(dbid), owner from defect group by owner" returns the number of defects per owner, showing you the count and the owner for each owner.

Its usually illegal to mix in these aggregate functions with other fields in the select clause unless they are also mentioned in a group-by clause. You can have more than one aggregate however, such as "select count(dbid), max(severity), min(due_date), owner group by owner."

Syntax

VBScript

queryfielddef.**AggregateFunction**

queryfielddef.**AggregateFunction** *NewValue*

Perl

*$queryfielddef->***GetAggregateFunction**

*$queryfielddef->***SetAggregateFunction** *(NewValue);*

Identifier	Description
queryfielddef	A QueryFieldDef object.
NewValue	A Long that specifies the function type for the field. The value corresponds to one of the DbAggregate constants.
Return value	Returns a Long that identifies the function type for the field. The value corresponds to one of the DbAggregate constants.

Example

Perl

```
use CQPerlExt;
#Start a ClearQuest session
#$AdminSession= CQAdminSession::Build();
$SessionObj = CQSession::Build();

$dbsetname = "CQMS.SAMPL.HOME";
```

```
#Refresh list of accessible databases
$databases = $SessionObj->GetAccessibleDatabases("MASTR", "", $dbsetname);

#Log into a database
$SessionObj->UserLogon("admin","","SAMPL",$dbsetname);

$querydef = $SessionObj->BuildQuery("defect") ;
$querydef->BuildField("id") ;
$querydef->BuildField("owner.login_name") ;

$queryfielddefs = $querydef->GetQueryFieldDefs();
$idfield = $queryfielddefs->ItemByName("id");
$idfield->SetAggregateFunction($CQPerlExt::CQ_DB_AGGR_COUNT);

$ownerfield = $queryfielddefs->ItemByName("owner.login_name");
$ownerfield->SetIsGroupBy(TRUE);

$resultset = $SessionObj->BuildResultSet($querydef);
$ct = $resultset->ExecuteAndCountRecords();
for ($i = 0; $i < $ct; $i++) {
   $resultset->MoveNext();
   print $resultset->GetColumnValue(1);
   print " ";
   print $resultset->GetColumnValue(2);
   print "\n";
}
CQAdminSession::Unbuild($AdminSession);
CQSession::Unbuild($SessionObj);
```

See Also

"Sorting a Result Set" on page 848

DbAggregate Constants

IsGroupBy

ChoiceList

Description Returns the items in a choice list for a QueryFieldDef.

Syntax

VBScript

*queryfielddef.***ChoiceList**

Perl

*$queryfielddef->***GetChoiceList***();*

Identifier	Description
queryfielddef	A QueryFieldDef object.
Return value	For VBScript, a Variant which is an array of strings, if there is a static choice-list available for this field from the schema. For Perl, a reference to an array of strings, if there is a static choice-list available for this field from the schema.

See Also

"Sorting a Result Set" on page 848
GetFieldChoiceList of the **Entity Object**
GetParamChoiceList of the **ResultSet Object**
HookChoices Object

DataType

Description Returns the underlying datatype of the QueryFieldDef object. The CType enum that is returned identifies the basic column datatype of the field.

Syntax

VBScript

queryfielddef.**DataType**

Perl

*$queryfielddef->***GetDataType***();*

Identifier	Description
queryfielddef	A QueryFieldDef object.
Return value	Returns a Long that identifies the underlying data type for the field. The value corresponds to one of the CType constants.

See Also **GetColumnType** of the **ResultSet Object**
CType Constants
FieldType

Description

Description Returns the description of the field from the schema.

Syntax

VBScript

*queryfielddef.***Description**

Perl

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

Identifier	Description
queryfielddef	A QueryFieldDef object.
Return value	Returns a String, containing the description of the field.

See Also "Sorting a Result Set" on page 848

FieldPathName

Description

Sets or returns the name of the field from the schema. This might be a "dotted name" if it is from within a reference (for example, owner.login_name).

Field path names are normally set on the **`BuildField`** function of the QueryDef object.

Syntax

VBScript

*`queryfielddef.`***FieldPathName**

*`queryfielddef.`***FieldPathName** *`NewValue`*

Perl

*`$queryfielddef->`***GetFieldPathName** *`();`*

*`$queryfielddef->`***SetFieldPathName** *`(NewValue);`*

Identifier	Description
`queryfielddef`	A QueryFieldDef object.
NewValue	A String that specifies the path for the field.
Return value	Returns a String that identifies the path for the field.

See Also

"Sorting a Result Set" on page 848
`BuildField` of the **`QueryDef Object`**
`GetLocalFieldPathNames` of the **`EntityDef Object`**
"Using Field Path Names to Retrieve Field Values" on page 834

FieldType

Description Returns the field type of the QueryFieldDef. The FieldType enum that is returned identifies the schema datatype of the field.

Syntax

VBScript

*queryfielddef.***FieldType**

Perl

*$queryfielddef->***GetFieldType** *();*

Identifier	Description
queryfielddef	A QueryFieldDef object.
Return value	Returns a Long that identifies the field type for the field. The value corresponds to one of the FieldType constants.

See Also **FieldType Constants**
DataType
"Sorting a Result Set" on page 848

Function

Description

Set or returns a function for the QueryFieldDef . The DbFunction enum identifies the function for the field.

The following date functions are currently supported. These DbFunctions apply to fields of type _DATE_TIME (that is, they can only be applied to datetime fields).

For example, given the submit date of 8/29/2002, you get the following values for the DbFunctions:

- year: 1/1/2002 //ie the date that is the first of the year
- week: 35,2002 //ie the 35th week of year 2002 separated with a comma
- month: 8/1/2002 //the first of the month
- day: 8/29/2002 //the day itself

Syntax

VBScript

*queryfielddef.***Function**

*queryfielddef.***Function** *NewValue*

Perl

*$queryfielddef->***GetFunction** *();*

*$queryfielddef->***SetFunction** *(NewValue);*

Identifier	Description
queryfielddef	A QueryFieldDef object.
NewValue	A Long that specifies the function type for the field. The value corresponds to one of the **DbFunction Constants**.
Return value	Returns a Long that identifies the function type for the field. The value corresponds to one of the **DbFunction Constants**.

See Also

"Sorting a Result Set" on page 848

DbFunction Constants

IsGroupBy

Description Sets or returns whether the field is in a group-by clause. Enables you to include a field in the group-by clause when aggregation functions are used.

Syntax

VBScript

*queryfielddef.***IsGroupBy**

*queryfielddef.***IsGroupBy** *NewValue*

Perl

*$queryfielddef->***GetIsGroupBy***();*

*$queryfielddef->***SetIsGroupBy***(NewValue);*

Identifier	Description
queryfielddef	A QueryFieldDef object.
NewValue	A Boolean that specifies whether the field is included in a group-by clause when aggregation functions are used.
Return value	Returns a Boolean whose value is True if the field is included in a group-by clause when aggregation functions are used, False otherwise.

See Also "Sorting a Result Set" on page 848

`AggregateFunction`

IsLegalForFilter

Description Returns whether you can use the field in a query filter. The default is True.

Syntax

VBScript

*queryfielddef.***IsLegalForFilter**

Perl

*$queryfielddef->***GetIsLegalForFilter***();*

Identifier	Description
queryfielddef	A QueryFieldDef object.
Return value	Returns a Boolean whose value is True if the field can be used in a filter, False otherwise. The default is True.

See Also "Sorting a Result Set" on page 848

IsShown

Description Enables you to sort by a field but not display the field. Sets or returns whether you want to show the field or not. The default is True.

Syntax

VBScript

*queryfielddef.***IsShown**

*queryfielddef.***IsShown** *NewValue*

Perl

*$queryfielddef->***GetIsShown***();*

*$queryfielddef->***SetIsShown***(NewValue);*

Identifier	Description
queryfielddef	A QueryFieldDef object.
NewValue	A Boolean that specifies whether the field is shown in a ResultSet.
Return value	Returns a Boolean whose value is True if the field is to be included in a ResultSet, False otherwise.

See Also "Sorting a Result Set" on page 848
BoolOp Constants
QueryFilterNode Object

Label

Description

Sets or returns the column label for a field in a ResultSet. The column label is the heading text for a specified column. The return value defaults to fieldpathname if not set.

The value you specify, when you use this method to set the label value is the return value you get for the GetColumnLabel method of the ResultSet object.

Syntax

VBScript

*queryfielddef.***Label**

*queryfielddef.***Label** *NewValue*

Perl

*$queryfielddef->***GetLabel***();*

*$queryfielddef->***SetLabel***(NewValue);*

Identifier	Description
queryfielddef	A QueryFieldDef object.
NewValue	A String whose value specifies the column label value for a ResultSet.
Return value	Returns a String containing the column label value for a ResultSet. If no column label value is set, the return value is the **`FieldPathName`**.

See Also

"Sorting a Result Set" on page 848

`GetColumnLabel` of the **`ResultSet Object`**

`FieldPathName`

SortOrder

Description Sets or returns the numeric order of precedence for the field, when there is more than one sort key.

Syntax

VBScript

queryfielddef.**SortOrder**

queryfielddef.**SortOrder** *NewValue*

Perl

*$queryfielddef->***GetSortOrder***();*

*$queryfielddef->***SetSortOrder***(NewValue);*

Identifier	Description
queryfielddef	A QueryFieldDef object.
NewValue	A Long whose value represents the numeric order of precedence when there is more than one sort key.
Return value	Returns a Long containing the value that represents the numeric order of precedence of the sort.

Examples

VBScript

```
idfield.SetSortOrder 1
```

Perl

```
$idfield->SetSortOrder(1);
```

See Also "Sorting a Result Set" on page 848

SortType

Description Sets or returns the sort type for the field. You can specify whether to sort ascending or descending with the **`Sort Constants`**.

Syntax

VBScript

*queryfielddef.***SortType**

*queryfielddef.***SortType** *NewValue*

Perl

*$queryfielddef->***GetSortType***();*

*$queryfielddef->***SetSortType***(NewValue);*

Identifier	Description
queryfielddef	A QueryFieldDef object.
NewValue	A Long that specifies the sort type for the field. The value corresponds to one of the **`Sort Constants`**.
Return value	Returns a Long that identifies the sort type for the field. The value corresponds to one of the **`Sort Constants`**.

Examples

VBScript

```
idfield.SetSortType AD_SORT_ASC
```

Perl

```
$idfield->SetSortType($CQPerlExt::CQ_SORT_ASC);
```

See Also "Sorting a Result Set" on page 848

`Sort Constants`

QueryFieldDef Object Methods

Note: For all Perl Get and Set methods that map to Visual Basic Properties, see the Properties section of this object.

The following Perl Methods map to Visual Basic properties:

Method Name	Description
GetAggregateFunction	Returns a SQL aggregate function for the field.
GetChoiceList	Returns the items in a choice list for a QueryFieldDef.
GetDataType	Returns the underlying datatype of the QueryFieldDef object.
GetDescription	Returns the description of the field from the schema.
GetFieldPathName	Returns the name of the field from the schema.
GetFieldType	Returns the field type of the QueryFieldDef.
GetFunction	Returns a function for the QueryFieldDef .
GetIsGroupBy	Returns whether the field is in a group-by clause.
GetIsLegalForFilter	Returns whether you can use the field in a query filter.
GetIsShown	Returns whether the field is shown or not.
GetLabel	Returns the column label for a field in a ResultSet.
GetSortOrder	Returns the numeric order of precedence for the field, when there is more than one sort key.
GetSortType	Returns the sort type for the field.
SetAggregateFunction	Sets a SQL aggregate function for the field.
SetFieldPathName	Sets the name of the field from the schema.
SetFunction	Set a function for the QueryFieldDef .
SetIsGroupBy	Sets whether the field is in a group-by clause.
SetIsShown	Enables you to sort by a field but not display the field. Sets whether you want to show the field or not.
SetLabel	Sets the column label for a field in a ResultSet.
SetSortOrder	Sets the numeric order of precedence for the field, when there is more than one sort key.
SetSortType	Sets the sort type for the field.

QueryFieldDefs Object

34

The QueryFieldInfos object is a collection of QueryFieldDef objects.

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

See Also **`QueryFieldDef Object`**

QueryFieldDefs Object Properties

The following list summarizes the QueryFieldDefs object properties:

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

Count

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

Syntax

VBScript

queryfielddefs.**Count**

Perl

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

Identifier	Description
queryfielddefss	An QueryFieldDef collection object, representing the set of QueryFieldDefs available for fetching as a collection.
Return value	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**
QueryFieldDef Object

QueryFieldDefs Object Methods

The following list summarizes the QueryFieldDefs object methods:

Method Name	Description
`Add`	Adds a QueryFieldDef object to the collection.
`AddUniqueKey`	Adds a unique key field QueryFieldDef to the QueryFieldDefs object.
`Item`	Returns the specified item in the collection.
`ItemByName`	(Perl only) Returns the specified item in the collection.
`Remove`	Removes a QueryFieldDef object from the collection.

Note: For Perl methods that map to Visual Basic Properties, see the Properties section of this object.

The following list summarizes additional Perl QueryFieldDefs object methods:

Method Name	Access	Description
`Count`	Read-only	Returns the number of items in the collection.

Add

Description

Adds a new QueryFieldDef object to this QueryFieldDefs collection. While Perl has separate methods for adding a QueryFieldDef to the collection, the COM interface provides one method.

For VB, you use the Add method to add a new QueryFieldDef to the collection. If the Variant argument:

- Contains a BSTR, then the content of the BSTR is treated as the fieldPath.
- Does not contain a BSTR, it should have type VT_EMTPY.

For Perl, there are two separate methods that accomplish the same functionality:

- AddByFieldPath
- AddNew

Passing in a fieldpath argument to this method, fills in the fieldpath for the new object. If you do not specifiy a fieldpath, then you must use the **`FieldPathName`** method of the **`QueryFieldDef Object`**.

Note: This method provides an alternate method to the **`BuildField`** method of the **`QueryDef Object`**.

These methods add a new QueryFieldDef object to the end of the collection. You can retrieve items from the collection using the **`Item`** method.

Syntax

VBScript

`queryfielddefs``.`**Add** *`path`*

Perl

`$queryfielddefs->``AddByFieldPath`*`(fieldPath);`*

`$queryfielddefs->``AddNew`*`();`*

Identifier	Description
queryfielddefss	An QueryFieldDef collection object, representing the set of QueryFieldDefs available for fetching as a collection.
path	(VB only) A Variant containing either a field path as a BSTR, or VT_EMPTY which adds a new QueryFieldDef object to this collection.
fieldPath	(Perl only) A String containing the field path to the QueryFieldDef object to add to this collection.
Return value	Returns the QueryFieldDef object that was added.

See Also

`BuildField` of the **`QueryDef Object`**
`FieldPathName` of the **`QueryFieldDef Object`**
`Count`
`Item`

AddUniqueKey

Description

Adds the unique key field of the primary record type of this query to the QueryFieldDefs object.

Creates and adds to the collection a QueryFieldDef object that makes up a unique key field. This field is a concatenation, into a single string, of all the fields that make up a unique key (if there are more than one). For example, if you have a unique key of first-name + last-name, this would generate SQL like (syntax varies by vendor):

```
select concat(first_name, last_name) from defect.
```

You would get back a single result column with values like "Joe Jones".

Syntax

VBScript

queryfielddefs.**AddUniqueKey**

Perl

$queryfielddefs->**AddUniqueKey**();

Identifier	Description
queryfielddefss	An QueryFieldDef collection object, representing the set of QueryFieldDefs available for fetching as a collection.
Return value	Returns the QueryFieldDef object.

See Also

Count
Item
BuildUniqueKeyField of the **QueryDef Object**

Item

Description

Returns the specified item in the QueryFieldDefs collection.

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

- For VB, the argument to the Item method can be either a numeric index or a name.
- For Perl, there are two separate methods:
  - **Item**, which takes a numerical index argument
  - **ItemByName**, which takes a string name argument

Syntax

VBScript

queryfielddefs.**Item** *item*

Perl

$queryfielddefs->**Item**(*itemNum*);

$queryfielddefs->**ItemByName**(*itemName*);

Identifier	Description
queryfielddefss	An QueryFieldDef collection object, representing the set of QueryFieldDefs available for fetching as a collection.
item	(VB only) A Variant containing either an numeric index (item number) or a BSTR name (item name).
itemNum	(Perl only) 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.
itemName	(Perl only) A String that serves as a key into the collection.
Return value	The QueryFieldDef object at the specified location in the collection.

See Also

Count
Add

Remove

Description Removes a QueryFieldDef object from the QueryFieldDefs collection.

Syntax

VBScript

```
queryfielddefs.Remove item
```

Perl

```
$queryfielddefs->Remove(item);
```

Identifier	Description
queryfielddefss	An QueryFieldDef collection object, representing the set of QueryFieldDefs available for fetching as a collection.
item	The QueryFieldDef object to add to this collection.
Return value	A Boolean that is True if the QueryFieldDef object was removed successfully, otherwise False.

See Also `Count`
`Item`

QueryFilterNode Object

35

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` of the **Session Object**
`BuildFilterOperator` of the **QueryDef Object**
ResultSet Object
QueryDef Object
"Building Queries for Defects and Users" on page 817

QueryFilterNode Object Methods

The following list summarizes the QueryFilterNode object methods.

Method Name	Description
`BuildFilter`	Adds a comparison operand to the node.
`BuildFilterOperator`	Creates a nested **QueryFilterNode Object** that contains the specified Boolean operator.

BuildFilter

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 Visual Basic Variant or Perl reference to an array of strings 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:

```
@dateRange = (“2000-12-01”, “2000-12-15”);
$node->BuildFilter("submit_date", $CQPerlExt::CQ_COMP_OP_BETWEEN,
 \@dateRange);
```

See also the example given for QueryDef's **`BuildFilterOperator`** method.

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.

To obtain valid values for the field_name argument, call the **`GetFieldDefNames`** method of the EntityDef object upon which the query was based.

Syntax

VBScript

node.**BuildFilter** *field_name, comparison_operator, value*

Perl

$*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	The value you want to find in the specified field. In Visual Basic, specify the value as a Variant or Variant array. In Perl, specify the value as a reference to a string or an array of strings.
Return value	None.

Examples

VBScript

```
Dim dateRange(2)
dateRange(0) = "2000-12-01"
```

```
dateRange(1) = "2000-12-15"
node.BuildFilter "submit_date", AD_COMP_OP_BETWEEN, dateRange
```

Perl

```
#Example1

@dateRange = ("2000-12-01", "2000-12-15");
$node->BuildFilter("submit_date", $CQPerlExt::CQ_COMP_OP_BETWEEN,
      \@dateRange);


#Example2

@owner = ("jsmith");

@state = ("closed");

$queryDef = $CQsession->BuildQuery("defect");

@dbfields = ("ID","State","Headline");

foreach $field (@dbfields) {

   $queryDef->BuildField($field);

   }

$operator=$queryDef->BuildFilterOperator($CQPerlExt::CQ_BOOL_OP_AND);

$operator->BuildFilter("Owner", $CQPerlExt::CQ_COMP_OP_EQ,\@owner);

$operator->BuildFilter("State", $CQPerlExt::CQ_COMP_OP_NOT_IN, \@state);
```

See Also

`BuildFilterOperator`
`BuildFilterOperator` of the **QueryDef Object**
"Building Queries for Defects and Users" on page 817
"Notation Conventions for VBScript" on page 3
"Notation Conventions for Perl" on page 2

BuildFilterOperator

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.

Syntax

VBScript

node.**BuildFilterOperator** *bool_operator*

Perl

$*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.

Examples

VBScript

```
' query for (id in idRange) AND (submitter = jjones OR clopez OR kwong)
Set qdef = sessionObj.BuildQuery("Defect")
qdef.BuildField ("id")
qdef.BuildField ("headline")
'Here is the root operator
Set filterNode1 = qdef.BuildFilterOperator(AD_BOOL_OP_AND)
Dim idRange(2)
idRange(0) = "SAMPL00000055"
idRange(1) = "SAMPL00000057"

filterNode1.BuildFilter "id", AD_COMP_OP_IN, idRange
'Here is the subnode operator
Set 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"
```

Perl

```
# query for (Owner = jsmith) AND (state = closed)
@owner = ("jsmith");
@state = ("closed");
```

```
$queryDef = $CQSession->BuildQuery("Defect");
@dbfields = ("ID","State","Headline");
foreach $field (@dbfields) {
      $queryDef->BuildField($field);
      }
$FilterNode1 = $queryDef->BuildFilterOperator($CQPerlExt::CQ_BOOL_OP_AND);
$FilterNode1->BuildFilter("Owner", $CQPerlExt::CQ_COMP_OP_EQ, \@owner);
$FilterNode1->BuildFilter("State", $CQPerlExt::CQ_COMP_OP_NOT_IN, \@state);

$resultSet = $CQsession->BuildResultSet($queryDef);
$resultSet->Execute;
$num_columns = $resultSet->GetNumberOfColumns;
```

See Also

`BuildFilter`
`BuildFilterOperator` of the **QueryDef Object**
"Building Queries for Defects and Users" on page 817

ReportMgr Object

36

The ReportMgr object provides an interface for generating reports.

Note: The ReportMgr object is for Windows only.

You can use this object to write external applications to execute reports defined in the Rational 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`	Executes the report and generates the resulting HTML file.
`GetQueryDef`	Returns the QueryDef object associated with the report.
`GetReportPrintJobStatus`	Returns the current status of the print job.
`SetHTMLFileName`	Sets the output file name for the report.

Note: These methods are for Windows only.

ExecuteReport

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.

Syntax

VBScript

reportMgr.**ExecuteReport**

Perl

$reportMgr->**ExecuteReport**();

Identifier	Description
reportMgr	The ReportMgr object associated with the current session.
Return value	None.

See Also

GetReportMgr of the **Workspace Object**
SetHTMLFileName

GetQueryDef

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.

Syntax

VBScript

reportMgr.**GetQueryDef**

Perl

$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

Description

Returns the current status of the print job.

This method indicates whether or not the report writing tool has finished generating the report. After this method returns crPrintingCompleted, you can open the generated report file and begin examining the data.

Syntax

VBScript

reportMgr.**GetReportPrintJobStatus**

Perl

$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

SetHTMLFileName

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.

Syntax

VBScript

reportMgr.**SetHTMLFileName** *htmlPath*

Perl

$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

GetReportMgr of the **Workspace Object**

ResultSet Object

37

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.

Note that:

- Columns are numbered from (1 through N), not (0 through N—1).
- After the result set is generated, you may need to fill in parameter values, if it is a parameterized query.
- After the result set is generated and parameters (if any) are set, you may execute it to see the output of the query. It is permitted to execute the result set multiple times, if you wish to rerun the query. (Perhaps you have cleared and reset the parameter values.) It is also legal to get the SQL for the query.
- Conceptually a query may generate so much output that it would be impossible or greatly inefficient to just copy it from the database over into the memory of the program. So, you have to use a "cursor" to navigate through the output, using the **`MoveNext`** method.
- Immediately after executing the result set, the cursor is positioned "just before" the first item, so you have to call MoveNext before you can extract the first value.
- To get the value after you are positioned, use the **`GetColumnValue`** method.

See Also

`BuildResultSet` of the Session object
QueryDef Object
Session Object
“Running a Query and Reporting on its Result Set” on page 845

ResultSet Object Properties

The following list summarizes the ResultSet object properties:

Property Name	Description
`MaxMultiLineTextLength`	Sets or returns the current limit on length of data fetched from a multiline text field.
`RecordCount`	Returns the record count (the number of rows) of the result set.

MaxMultiLineTextLength

Description

Gets or sets the current limit on data to be fetched for a multiline, text field.

This is useful if your results include one or more fields containing a long, multiline text entry, and there is a chance that fetching the data could overrun your buffer space. It is also useful if you just want to browse results and want better performance.

By default, there is no limit on the length of data fetched from a multiline, text field.

You can reset the default by setting the length parameter to zero (0).

Syntax

VBScript

resultset.**MaxMultiLineTextLength**

MaxMultiLineTextLength *max_length*

Perl

$*resultset*->**GetMaxMultiLineTextLength()**;

$*resultset*->**SetMaxMultiLineTextLength**(*$max_length*);

Identifier	Description
resultset	A ResultSet object, representing the rows and columns of data resulting from a query.
max _length	A Long specifying the current maximum length in bytes of data fetched from a multiline text field.
Return value	Returns a Long containing the current maximum length in bytes of data fetched from a multiline text field.

Example

Perl

```
$queryDefObj = $SessionObj->BuildQuery("Defect");
$queryDefObj->BuildField("description");
$queryDefObj->BuildField("id");
$resultSetObj = $SessionObj->BuildResultSet($queryDefObj);

$resultSetObj->SetMaxMultiLineTextLength(5);
# not setting the above max multiline text length
# or setting it to 0 will fetch the entire data of
# the long varchar column

$resultSetObj->Execute();
$status = $resultSetObj->MoveNext();
$i=0;
while ($status == 1) {
    $xnote = $resultSetObj->GetColumnValue(1);
```

```
    print $i++,". desc=",$xnote,"\n";
    $entyObj = $SessionObj->GetEntity( "defect",
        $resultSetObj->GetColumnValue(2));
    $SessionObj->EditEntity($entyObj,"modify");
    $entyObj->SetFieldValue("headline","testXXX".($i));
    $retval = $entyObj->Validate();
    $entyObj->Commit();

    $status = $resultSetObj->MoveNext();
}
```

See Also **`BuildResultSet`** of the **`Session Object`**

RecordCount

Description

Returns the record count (the number of rows) of the result set.

To get a record count, you first use **`EnableRecordCount`** to enable row count before calling RecordCount (GetRecordCount, for Perl) to get the number of records.

Syntax

VBScript

`resultset`.**RecordCount**

Perl

$*`resultset`***->GetRecordCount()**;

Identifier	Description
resultset	A ResultSet object, representing the rows and columns of data resulting from a query.
Return value	A Long containing the number of records in the result set.

Example

VBScript

```
Set ResultSet = cqSession.BuildResultSet(qrydef)
ResultSet.EnableRecordCount
ResultSet.Execute
count = ResultSet.RecordCount
```

See Also

`EnableRecordCount`

ResultSet Object Methods

The following list summarizes the ResultSet object methods:

Note: For all Perl Get and Set methods that map to Visual Basic Properties, see the Properties section of this object.

Method Name	Description
AddParamValue	Assigns one or more values to a parameter.
ClearParamValues	Clears all values associated with a parameter.
EnableRecordCount	Enables a record count for the result set.
Execute	Runs the query and fills the result set with data.
GetColumnLabel	Returns the heading text for the specified column.
GetColumnType	Returns the type of data stored in the specified column.
GetColumnValue	Returns the value stored in the specified column of the current row.
GetNumberOfColumns	Returns the number of columns in each row of the result set.
GetNumberOfParams	Returns the number of parameters in this query.
GetParamChoiceList	Returns a list of permitted values for the parameter.
GetParamComparisonOperator	Returns the comparison operator associated with the parameter.
GetParamFieldType	Returns the field type of the parameter.
GetParamLabel	Returns the name of the parameter.
GetParamPrompt	Returns the prompt string displayed to the user for the given parameter.
GetRowEntityDefName	Based on the result set, this method returns a String with the record type (EntityDef) name of the current row.
GetSQL	Returns the SQL string that expresses the query.
LookupPrimaryEntityDefName	Returns the name of the EntityDef object on which the query is based.
MoveNext	Moves the to the next record in the data set.
SetParamComparisonOperator	Sets the comparison operator associated with the parameter.

Additional Perl Get and Set Methods that map to Visual Basic properties:

Method Name	Description
Get`MaxMultiLineTextLength`	Gets current limit on length of data fetched from a multiline text field.
Get`RecordCount`	Returns the record count (the number of rows) of the result set.
Set`MaxMultiLineTextLength`	Sets a limit on length of data fetched from a multiline text field.

AddParamValue

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.

Syntax

VBScript

resultset.**AddParamValue** *param_number, value*

Perl

$*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.
value	For Visual Basic, a Variant containing the value for the parameter. For Perl, a String containing the value for the parameter.
Return value	None.

See Also **ClearParamValues**

ClearParamValues

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.

Syntax

VBScript

resultset.**ClearParamValues** *param_number*

Perl

$*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.
Return value	None.

See Also **AddParamValue**

EnableRecordCount

Description

Enables a record count for the result set.

To get a record count, you first use EnableRecordCount to enable row count and then call **`RecordCount`** to get the number of records. Use this method after defining the ResultSet object, but before executing it.

Syntax

VBScript

`resultset``.`**EnableRecordCount**

Perl

$*`resultset`***->EnableRecordCount()**;

Identifier	Description
resultset	A ResultSet object, representing the rows and columns of data resulting from a query.
Return value	None.

Example

VBScript

```
Set ResultSet = cqSession.BuildResultSet(qrydef)
ResultSet.EnableRecordCount
ResultSet.Execute
count = ResultSet.RecordCount
```

See Also

`RecordCount`

Execute

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.

Syntax

VBScript

resultset.**Execute**

Perl

$*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
GetSQL
MoveNext
BuildResultSet of the Session object
Session Object
"Running a Query and Reporting on its Result Set" on page 845
"Getting a List of Defects and Modifying a Record" on page 847

GetColumnLabel

Description

Returns the heading text for the specified column.

Columns are numbered from 1 to N, not 0 to N—1.

Syntax

VBScript

resultset.**GetColumnLabel** *columnNum*

Perl

$*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
GetColumnValue
GetNumberOfColumns
"Running a Query and Reporting on its Result Set" on page 845
Label method of the **QueryFieldDef Object**

GetColumnType

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.

Syntax

VBScript

resultset.**GetColumnType** *columnNum*

Perl

$*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
GetColumnValue
GetNumberOfColumns
CType Constants
DataType method of the **QueryFieldDef Object**

GetColumnValue

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 Rational 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.

Syntax

VBScript

resultset.**GetColumnValue** *columnNum*

Perl

$*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 Visual Basic, 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.

See Also

Execute
GetColumnLabel
GetColumnType
GetNumberOfColumns
MoveNext
"Running a Query and Reporting on its Result Set" on page 845
"Getting a List of Defects and Modifying a Record" on page 847

GetNumberOfColumns

Description Returns the number of columns in each row of the result set.

Syntax

VBScript

resultset.**GetNumberOfColumns**

Perl

$*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
GetColumnType
GetColumnValue
"Running a Query and Reporting on its Result Set" on page 845

GetNumberOfParams

Description Returns the number of parameters in this query.

Syntax

VBScript

resultset.**GetNumberOfParams**

Perl

$*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
GetParamComparisonOperator
GetParamFieldType
GetParamLabel
GetParamPrompt

GetParamChoiceList

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.

Syntax

VBScript

resultset.**GetParamChoiceList** *param_number*

Perl

$*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.
Return value	For Visual Basic, 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 an array of strings.

See Also

GetNumberOfParams
GetParamComparisonOperator
GetParamFieldType
GetParamLabel
GetParamPrompt

GetParamComparisonOperator

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.

Syntax

VBScript

resultset.**GetParamComparisonOperator** *param_number*

Perl

$*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.
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
GetParamChoiceList
GetParamFieldType
GetParamLabel
GetParamPrompt
CompOp Constants

GetParamFieldType

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.

Syntax

VBScript

resultset.**GetParamFieldType** *param_number*

Perl

$*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.
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
GetParamChoiceList
GetParamComparisonOperator
GetParamLabel
GetParamPrompt
FieldType Constants

GetParamLabel

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.

Syntax

VBScript

resultset.**GetParamLabel** *param_number*

Perl

$*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.
Return value	A String containing the name of the parameter.

See Also

GetNumberOfParams
GetParamChoiceList
GetParamComparisonOperator
GetParamFieldType
GetParamPrompt

GetParamPrompt

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.

Syntax

VBScript

resultset.**GetParamPrompt** *param_number*

Perl

$*resultset***->GetParamPrompt(***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.
Return value	A String containing the prompt string displayed to the user.

See Also

GetNumberOfParams
GetParamChoiceList
GetParamComparisonOperator
GetParamFieldType
GetParamLabel

GetRowEntityDefName

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.

Syntax

VBScript

resultset.**GetRowEntityDefName**

Perl

$*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 EntityDef (record type) of the specified row.

See Also

"Running a Query Against More than One Record Type" on page 852

IsMultiType

GetSQL

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.

Syntax

VBScript

`resultset``.`**GetSQL**

Perl

$*`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`

LookupPrimaryEntityDefName

Description

Returns the name of the **EntityDef Object** on which the query is based.

A ResultSet can be 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.

Syntax

VBScript

resultset.**LookupPrimaryEntityDefName**

Perl

$*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` of the Session object
EntityDef Object
QueryDef Object
Session Object
"Running a Query and Reporting on its Result Set" on page 845

MoveNext

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.

Syntax

VBScript

```
fetchStatus = resultset.MoveNext
```

Perl

```
$fetchStatus = $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
GetColumnValue
"Running a Query and Reporting on its Result Set" on page 845
"Getting a List of Defects and Modifying a Record" on page 847

SetParamComparisonOperator

Description

Sets the comparison operator associated with the parameter.

The parameter number is a Long whose value is between 1 and the total number of parameters.

Syntax

VBScript

resultset.**SetParamComparisonOperator** *param_number, compOp*

Perl

$*resultset***->SetParamComparisonOperator(***param_number, compOp***);**

Identifier	Description
resultset	A ResultSet object, representing the rows and columns of data resulting from a query.
param_number	A Long identifying the parameter.
compOp	A Long indicating the comparison operator for the parameter. The value corresponds to a value in the CompOp enumerated type.
Return value	None.

See Also

GetParamComparisonOperator
GetNumberOfParams
GetParamChoiceList
GetParamFieldType
GetParamLabel
GetParamPrompt
AddParamValue
CompOp Constants

Schema Object

38

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 Rational ClearQuest Designer. You can get a list of schemas defined in the schema repository (master database) by accessing the **`Schemas`** method of the **AdminSession Object**.

See Also

`Schemas` 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`	Read-only	Returns a string containing the name of the schema.
`SchemaRevs`	Read-only	Returns the collection containing schema revisions.

Name

Description Returns the name of this schema.

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

Syntax

VBScript

schema.**Name**

Perl

```
$schema->GetName();
```

Identifier	Description
schema	A Schema object.
Return value	A String containing the name of this schema.

See Also **SchemaRevs**

SchemaRevs

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.

Syntax

VBScript

schema.**SchemaRevs**

Perl

```
$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

Schema Object Methods

Note: For Perl methods that map to Visual Basic Properties, see the Properties section of this object.

The following list summarizes the Perl Schema object methods:

Method Name	Access	Description
GetName	Read-only	Returns a string containing the name of the schema.
GetSchemaRevs	Read-only	Returns the collection containing schema revisions.

SchemaRev Object

39

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 **`Upgrade`** method of the Database object.

To discover which packages and package revisions apply to the current user database, use the **`GetEnabledPackageRevs`** and the **`GetEnabledEntityDefs`** methods.

See Also

`SetInitialSchemaRev` of the **`Database Object`**
`SchemaRev` of the **`Database Object`**
`Upgrade` of the **`Database Object`**
Schema Object
`PackageRev Object`

SchemaRev Object Properties

The following list summarizes the SchemaRev object properties:

Property Name	Access	Description
`Description`	Read-only	Returns a description of this schema revision.
`RevID`	Read-only	Returns the version ID of this schema revision.
`Schema`	Read-only	Returns the schema to which this revision belongs.

Description

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.

Syntax

VBScript

schemaRev.**Description**

Perl

$schemaRev->**Get**Description();

Identifier	Description
schemaRev	A SchemaRev object.
Return value	A String containing the description of the schema revision.

See Also **Schema Object**

RevID

Description

Returns the version number of this schema revision.

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

Syntax

VBScript

schemaRev.**RevID**

Perl

$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

Description

Returns the schema to which this revision belongs.

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

Syntax

VBScript

```
schemaRev.Schema
```

Perl

```
$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:

Method Name	Description
`GetEnabledEntityDefs`	Returns the EntityDefs collection object enabled in the current schema for a given package revision.
`GetEnabledPackageRevs`	Returns a collection object representing the PackageRev set that is enabled in the current revision of the schema.

Note: For Perl methods that map to VB Properties, see the Properties section of this object.

The following list summarizes additional Perl SchemaRev object methods:

Method Name	Access	Description
`GetDescription`	Read-only	Returns a description of this schema revision.
`GetRevID`	Read-only	Returns the version ID of this schema revision.
`GetSchema`	Read-only	Returns the schema to which this revision belongs.

GetEnabledEntityDefs

Description

Returns the collection (an EntityDefs object) of EntityDef objects that are enabled in the current schema for a given package revision.

Use with **`GetEnabledPackageRevs`** to discover which packages and package revisions apply to the current user database.

Syntax

VBScript

`schemaRev``.`**GetEnabledEntityDefs** *`packName, rev`*

Perl

`$schemaRev->``GetEnabledEntityDefs(`*`packName, rev);`*

Identifier	Description
schemaRev	A SchemaRev object.
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`

GetEnabledPackageRevs

Description

Returns a collection object representing the PackageRev set that is enabled in the current schema revision.

Use with **`GetEnabledEntityDefs`** 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.

Syntax

VBScript

`schemaRev``.`**GetEnabledPackageRevs**

Perl

`$schemaRev->``GetEnabledPackageRevs`*`;`*

Identifier	Description
schemaRev	A SchemaRev object.
Return values	The PackageRevs collection object of the PackageRev set.

See Also

`GetEnabledEntityDefs`
`GetEnabledPackageRevs` in **Session Object**
`SchemaRev` of the **`Database Object`**

SchemaRevs Object

40

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`** method. Use the **`Item`** method to retrieve items from the collection.

See Also

`Schemas` of the **AdminSession Object**
SchemaRev Object
Schemas Object

SchemaRevs Object Properties

The following list summarizes the SchemaRevs object properties:

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

Count

Description Returns the number of items in the collection. This is a read-only property; it can be viewed but not set.

Syntax

VBScript

collection.**Count**

Perl

*$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`

SchemaRevs Object Methods

The following list summarizes the SchemaRevs object methods:

Method	Access	Description
`Item`	Read-only	Returns the item at the specified index in the collection.
`ItemByName`	Read-only	(Perl only) Returns the specified item in the collection.

Note: For Perl methods that map to VB Properties, see the Properties section of this object.

The following list summarizes additional Perl SchemaRevs object methods:

Method Name	Access	Description
`Count`	Read-only	Returns the number of items in the collection.

Item

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

VBScript

```
collection.Item(itemNum)
collection.Item(name)
```

Perl

```
$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`

Schemas Object

41

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`** method. Use the **`Item`** method to retrieve items from the collection.

See Also **SchemaRev Object**

Schemas Object Properties

The following list summarizes the Schemas object properties:

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

Count

Description

Returns the number of items in the collection.

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

Syntax

VBScript

collection.**Count**

Perl

*$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`

Schemas Object Methods

The following list summarizes the Schemas object methods:

Method Name	Description
`Item`	Returns the item at the specified index in the collection.
`ItemByName`	(Perl only) Returns the specified item in the collection.

Note: For Perl methods that map to Visual Basic Properties, see the Properties section of this object.

The following list summarizes additional Perl Schemas object methods:

Method Name	Access	Description
`Count`	Read-only	Returns the number of items in the collection.

Item

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

VBScript

```
collection.Item(itemNum)
collection.Item(name)
```

Perl

```
$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`

Session Object

42

Users access a Rational 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. After 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 stand-alone 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.

Language Example	Syntax for Manually Creating the Session Object (or the AdminSession Object) in an External Application
Visual Basic	set currentSession = CreateObject("CLEARQUEST.SESSION")

Language Example	Syntax for Manually Creating the Session Object (or the AdminSession Object) in an External Application
Perl	$currentSession = CQSession::Build(); When you are done with the object, destroy it: CQSession::Unbuild($currentSession);

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 default connection) 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 Session methods that you can call before calling UserLogon such as **`GetAccessibleDatabases`** and **`OutputDebugString`**.

Task-Oriented Session Methods

- To submit or lookup entities, you need to know the appropriate EntityDef. Session operations relating to entity defs:
  `GetEntityDef`
  `GetDefaultEntityDef`
  `GetEntityDefNames`
  `GetReqEntityDefNames`
  `GetAuxEntityDefNames`
  `GetSubmitEntityDefNames`
  `GetQueryEntityDefNames`
- To lookup, submit, delete or modify an entity, use:
  `BuildEntity`
  `EditEntity`
  `GetEntity`
  `GetEntityByDbId`

DeleteEntity
StringIdToDbId
DbIdToStringId

- To add or delete duplicate relationships between entities, use:
  MarkEntityAsDuplicate
  UnmarkEntityAsDuplicate
- To search the database rather than just doing a lookup of a single entity, use the following Query-related operations:
  OpenQueryDef
  BuildResultSet
  BuildSQLQuery
  BuildQuery
- To get family information, use:
  GetEntityDefFamilyNames
  GetEntityDefFamilyName
  GetQueryEntityDefNames

See Also "Getting Session and Database Information" on page 850

Session Object Properties

The following list summarizes the Session object properties:

Property Name	Access	Description
`NameValue`	Read/Write	Gets or sets the value associated with a given variable name.

NameValue

Description

Sets or returns the value associated with a given variable name.

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 (for Visual Basic).

ClearQuest supports the use of session-wide variables for storing information. Once created, you can access session-wide variables through the current Session object at any time and from functions or subroutines, including hook routines, that have access to the Session object. When the current session ends, either because the user logged out or you deleted the Session object, all of the variables associated with that Session object are deleted. A Session-wide variable is accessed through the NameValue property (GetNameValue and SetNameValue methods for Perl). In addition, the HasValue method can be used to check whether a variable exists.

You can also store objects as session variables. For example:

```
set sessionObj.NameValue ("Obj") = object
```

or

```
set sessionObj.NameValue ("CalendarHandle") = param.ObjectItem
```

In the above example, param is the parameter to a record script hook and contains an object handle.

Elsewhere, you can then manipulate the properties of the object. For example:

```
Dim Calender
'Get the object handle
Set Calender = MySession.NameValue("CalendarHandle")
'Do something with the object ...
```

Syntax

VBScript

session.**NameValue** *variable_name*
session.**NameValue** *variable_name*, newValue

Perl

*$session->***GetNameValue(***variable_name***);**
*$session->***SetNameValue(***variable_name, newValue***);**

Identifier	Description
session	The Session object that represents the current database-access session.
variable_name	A String containing the name of the variable to get or set.
newValue	For Visual Basic, a reference to a Variant specifying the new value for the variable. For Perl, a String specifying the new value for the variable.

Identifier	Description
Return value	For Visual Basic, a reference to a Variant containing the value for the variable. For Perl, a String containing the value for the variable.

Example

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"
```

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`

Using Sessionwide Variables **on page 13**

Session Object Methods

The following list summarizes the Session object methods:

Method Name	Description
AddListMember	Adds a list member to the named list.
Build	(Perl only) Creates a Session object.
BuildEntity	Creates a new record of the specified type and begins a Submit action.
BuildQuery	Creates and returns a new QueryDef object for the specified record type.
BuildResultSet	Creates and returns a result set that can be used to run a query.
BuildSQLQuery	Creates and returns a ResultSet object using a raw SQL string.
CQDataCodePageIsSet	Returns whether the ClearQuest data code page has been set, or not.
DbIdToStringId	Returns the IDid string translated from dbid.
DeleteEntity	Deletes the specified record from the current database.
DeleteListMember	Deletes a member from a named list.
EditEntity	Performs the specified action on a record and makes the record available for editing.
FireRecordScriptAlias	Calls the action that calls the hook script; use to simulate a user choosing an action that launches a hook.
GetAccessibleDatabases	Returns a list of databases that are available for the specified user to log in to.
GetAuxEntityDefNames	Returns an array of strings, each of which corresponds to the name of one of the *schema* stateless record types.
GetClientCodePage	Returns a String describing the client's code page.
GetCQDataCodePage	Returns a String describing the ClearQuest data code page.
GetDefaultEntityDef	Returns the schema's default EntityDef object.
GetDisplayNamesNeedingSiteExtension	Gets the names of objects needing a site extension.
GetEnabledEntityDefs	Returns the EntityDefs collection object enabled in the current schema for a given package revision.

Method Name	Description
GetEnabledPackageRevs	Returns a collection object representing the PackageRev set that is enabled for the current revision of the schema.
GetEntity	Returns the specified record.
GetEntityByDbId	Returns the record with the specified database ID.
GetEntityDef	Returns the specified EntityDef object.
GetEntityDefFamilyName	Returns the requested EntityDef object if it is a family.
GetEntityDefFamilyNames	Returns an array containing the requested EntityDef family names.
GetEntityDefNames	Returns an array containing the names of the record types in the current database's *schema*.
GetEntityDefOrFamily	Returns the named EntityDef object.
GetInstalledDbSets	Returns the list of registered database sets.
GetInstalledMasterDbs	Returns the list of registered master databases.
GetInstalledMasters	Returns the list of registered database sets and master databases.
GetListDefNames	Returns a list of the dynamic lists in the current database.
GetListMembers	Returns the choice values of a dynamic list.
GetLocalReplica	Returns information about database replication.
GetMaxCompatibleFeatureLevel	Gets the maximum database version number supported by the ClearQuest client running on this machine.
GetMinCompatibleFeatureLevel	Gets the minimum database version number supported by the ClearQuest client running on this machine.
GetProductInfo	(Perl only) Returns a CQProductInfo object.
GetProductVersion	Returns the internal product version string that is hard-coded in header file.
GetQueryEntityDefFamilyNames	Returns the names of all family query EntityDefs.
GetQueryEntityDefNames	Returns the names of the record types that are suitable for use in queries.
GetReqEntityDefNames	Returns the names of the state-based record types in the current database's *schema*.
GetServerInfo	Returns a String identifying the session's OLE server.

Method Name	Description
GetSessionDatabase	Returns general information about the database that is being accessed in the current session.
GetSessionFeatureLevel	Gets the version number of the ClearQuest client currently running on this machine.
GetSiteExtendedNames	Gets extended names of database objects.
GetSiteExtension	Gets site extension for a database.
GetStageLabel	Returns stage label used for the build; the stage label is dynamically generated for each build.
GetSubmitEntityDefNames	Returns an array containing the names of the record types that are suitable for use in creating a new record.
GetSuiteProductVersion	Returns the suite version string.
GetUnextendedName	Gets the unextended name of a database.
GetUserEmail	Returns the electronic mail address of the user who is logged in for this session.
GetUserFullName	Returns the full name of the user who is logged in for this session.
GetUserGroups	Returns a list of the groups to which the current user belongs.
GetUserLoginName	Returns the name that was used to log in for this session.
GetUserMiscInfo	Returns miscellaneous information about the user who is logged in for this session.
GetUserPhone	Returns the telephone number of the user who is logged in for this session.
GetWorkSpace	Returns the session's Workspace object.
HasUserPrivilege	Tests a user privilege level in a security context.
HasValue	Returns a Bool indicating whether the specified session variable exists.
IsClientCodePageCompatibleWithCQDataCodePage	Returns whether the client code page is compatible with the ClearQuest data code page, or not.
IsMetadataReadonly	Returns a boolean indicating whether the session's metadata is read-only.
IsMultisiteActivated	Returns a boolean indicating whether the current database has been activated for multisite operations.

Method Name	Description
IsPackageUpgradeNeeded	Returns a boolean indicating whether the current revision of package that is applied to the schema is the highest package revision that is available for the package.
IsReplicated	Returns a boolean indicating whether the current database has at lease two replicated sites.
IsRestrictedUser	Tests user privileges in a security context.
IsSiteExtendedName	Tests whether a database name is an extended name.
IsStringInCQDataCodePage	Returns whether, or not, the ClearQuest data code page contains a given String.
IsUnix	Returns a Boolean indicating whether ClearQuest is running on a UNIX machine.
IsUnsupportedClientCodePage	Returns whether the client code page is unsupported, or not.
IsUserAppBuilder	Tests schema designer privileges in a security context.
IsUserSuperUser	Tests super user privileges in a security context.
IsWindows	Returns a Boolean indicating whether ClearQuest is running on a Windows machine.
LoadEntity	Gets latest values of a record.
LoadEntityByDbId	Gets latest values of a record.
MarkEntityAsDuplicate	Modifies the specified record to indicate that it is a *duplicate* of another record.
OpenQueryDef	Loads a query from a file.
OutputDebugString	Specifies a message that can be displayed by a debugger or a similar tool.
ParseSiteExtendedName	Splits a database name into an unextended name and a site extension.
SetListMembers	Sets the members in a named list.
SetRestrictedUser	Sets user restrictions in a security context.
StringIdToDbId	Returns the dbid number translated from string id.
Unbuild	(Perl only) Deletes a Session object, when you are done with it.
UnmarkEntityAsDuplicate	Removes the indication that the specified record is a *duplicate* of another record.
UserLogon	Log in as the specified user for a database session.

Method Name	Description
ValidateStringInCQDataCodePage	Checks to see if a given String is in the ClearQuest data code page for the session's schema-repository.

Note: For Perl methods that map to Visual Basic Properties, see the Properties section of this object.

The following list summarizes additional Perl Session object methods:

Method Name	Description
GetNameValue	Gets the value associated with a given variable name.
SetNameValue	Sets the value associated with a given variable name.

AddListMember

Description Adds a list member to the named list.

Syntax

VBScript

session.**AddListMember** *listName, listMember*

Perl

$*session***->AddListMember(***listName, listMember***);**

Identifier	Description
session	The Session object that represents the current database-access session.
listName	A String containing the name of the list.
listMember	A String containing the member in the list.
Return value	None

Examples

VBScript

```
' This example assumes there is at least 1 dynamic list
' in the current database-access session.
set sessionObj = GetSession
sessionObj.UserLogon "admin", "", "SAMPL", AD_PRIVATE_SESSION, ""
' Get a list of the names of Dynamic Lists that exist in this database...
DynamicListNamesRef = sessionObj.GetListDefNames
' For one of the lists, print out its members...
set ListName = DynamicListNamesRef(0)
   members = sessionObj.GetListMembers(ListName)
   ' print out the list members...
   For Each member In members
      print member
   Next
' Add a member, then print the list again...
set newMember = "XYZ"
MsgBox "Adding member: " + newMember + " to list" + ListName
sessionObj.AddListMember ListName, newMember
   members = sessionObj.GetListMembers(ListName)
   ' print out the list members...
   For Each member In members
      print member
   Next
```

Perl

```
# This example assumes there is at least 1 dynamic list
# in the current database-access session.
$sessionObj = $entity->GetSession();
$sessionObj->UserLogon("admin","","SAMPL","");

# Get a list of the names of Dynamic Lists that exist in this database...
$ListDefNamesREF = $sessionObj->GetListDefNames();
# For one of the lists, print out its members...
$ListName = @$ListDefNamesREF[0];
Print $ListName, "\n";
$members = $sessionObj->GetListMembers($ListName);
foreach $member (@$members){
   print $member, "\n";
   }
# Add a member, then print the list again...
$NewValue = "XYZ";
print "\nAdding member '$NewValue' to list '$ListName'...\n";
$sessionObj->AddListMember($ListName, $NewValue);
$members = $sessionObj->GetListMembers($ListName);
foreach $member (@$members){
   print $member, "\n";
   }
```

See Also

GetListMembers
DeleteListMember
SetListMembers
GetListDefNames

Build

Description Creates a Session object.

Note: This method is for Perl only.

Syntax

Perl

```
CQSession::Build();
```

Identifier	Description
CQSession	A static function specifier for a Session object.
Return value	A newly created Session object.

Example

Perl

```
use CQPerlExt;

my $sessionObj = CQSession::Build();

CQSession::Unbuild($sessionObj);
```

See Also

Unbuild
AdminSession Object

BuildEntity

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`** 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`** and **`Commit`** methods 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.

Syntax

VBScript

session.**BuildEntity** (*entitydef_name*)

Perl

$*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.

Examples

VBScript

```
set sessionObj = GetSession

' Create a new "defect" record
set entityObj = sessionObj.BuildEntity("defect")
```

Perl

```
$sessionobj = $entity->GetSession();
# Create a new "defect" record


$entityobj = $sessionobj->BuildEntity("defect");
```

See Also

`EditEntity`
`GetEntity`
`GetSubmitEntityDefNames`
`Commit` of the **`Entity Object`**
`SetFieldValue` of the **`Entity Object`**

`Validate` of the **`Entity Object`**
Entity Object
"Managing Records (Entities) that are Stateless and Stateful" on page 822

BuildQuery

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 the 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.

Syntax

VBScript

session.**BuildQuery** (*entitydef_name*)

Perl

$*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.

Examples

VBScript

```
set sessionObj = GetSession

' Create a query for "defect" records
set queryDefObj = sessionObj.BuildQuery("defect")
```

Perl

```
$sessionObj = $entity->GetSession();


# Create a query for "defect" records
$queryDefObj = $sessionObj->BuildQuery("defect");
```

See Also

`BuildResultSet`
`GetEntityDefNames`
`GetQueryEntityDefNames`

`BuildField` of the **QueryDef Object**
ResultSet Object
"Building Queries for Defects and Users" on page 817

BuildResultSet

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`**.

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.

Syntax

VBScript

`session``.`**BuildResultSet** `(`*`querydef`*`)`

Perl

$*`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.

Examples

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)
resultSetObj.Execute
```

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);

$resultsetObj->Execute();
```

See Also

`BuildQuery`
`BuildSQLQuery`
`Execute` of the **ResultSet Object**
QueryDef Object
"Building Queries for Defects and Users" on page 817
"Running a Query and Reporting on its Result Set" on page 845

BuildSQLQuery

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`**, 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`**, 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.

Syntax

VBScript

session.**BuildSQLQuery** (*SQL_string*)

Perl

$*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.

Examples

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)
```

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`
ResultSet Object
“Building Queries for Defects and Users” on page 817

CQDataCodePageIsSet

Description Returns whether the ClearQuest data code page has been set, or not.

Syntax

VBScript

*session.*__CQDataCodePageIsSet__

Perl

*$session->*__CQDataCodePageIsSet__*();*

Identifier	Description
session	The Session object that represents the current database-access session.
Return value	Returns True if the ClearQuest data code page has been set, False otherwise.

Examples

VBScript

```
isSet = session.CQDataCodePageIsSet
```

Perl

```
$isSet = $session->CQDataCodePageIsSet();
```

See Also

GetClientCodePage
GetCQDataCodePage
IsClientCodePageCompatibleWithCQDataCodePage
IsStringInCQDataCodePage
IsUnsupportedClientCodePage
ValidateStringInCQDataCodePage
CQDataCodePageIsSet of the **AdminSession Object**

DbIdToStringId

Description

Returns the ID string translated from dbid.

Dbid is a unique number assigned to every record by ClearQuest.

For stateful records, the string ID is the display name (for example, RAMBU00001234). For stateless records, the display name is composed of a concatenation of all the unique key field values with a space character between.

For example, if a project record type has two key fields, name and department and they are both designated as unique key fields, the string id would be "<name> <department>".

For a project with name "ACME" and department "Finance" the string id is "ACME Finance".

Syntax

VBScript

*session.*__DbIdToStringId__ *db_id*

Perl

*$session->*__DbIdToStringId__*(db_id);*

Identifier	Description
session	The Session object that represents the current database-access session.
db_id	A Long containing a record dbid.
Return value	Returns a String containing the record's string ID (display name).

Examples

VBScript

```
id = session.DbIdToStringId "33554434"
```

Perl

```
$id = $session->DbIdToStringId ("33554434");
```

See Also

StringIdToDbId
GetDbId of the **Entity Object**
GetDisplayName of the **Entity Object**

DeleteEntity

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.

Syntax

VBScript

session.**DeleteEntity** *entity, deleteActionName*

Perl

$*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.
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 ("").

Examples

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"
```

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
EditEntity
GetEntity
Entity Object

DeleteListMember

Description Deletes a member from a named list.

Syntax

VBScript

session.**DeleteListMember** *listName, listMember*

Perl

$*session***->DeleteListMember(***listName, listMember***);**

Identifier	Description
session	The Session object that represents the current database-access session.
listName	A String containing the name of the list.
listMember	A String containing the member in the list.
Return value	None.

Examples

VBScript

```
' This example assumes there is at least 1 dynamic list
' in the current database-access session.
set sessionObj = GetSession
sessionObj.UserLogon "admin", "", "SAMPL", AD_PRIVATE_SESSION, ""

' Get a list of the names of Dynamic Lists that exist in this database...
DynamicListNamesRef = sessionObj.GetListDefNames
' Get the name of one of the lists
set ListName = DynamicListNamesRef(0)
   print ListName
' Get the names of the list members
   members = sessionObj.GetListMembers(ListName)
   ' print out the list members...
   For Each member In members
      print member
   Next
' Add a member, then print the list again...
set newMember = "XYZ"
MsgBox "Adding member: " + newMember + " to list" + ListName
sessionObj.AddListMember ListName, newMember
   members = sessionObj.GetListMembers(ListName)
   ' print out the list members...
```

```
   For Each member In members
      print member
   Next
' Now delete a member, and print the list again...
MsgBox "Deleting member: " + newMember + " from list"
sessionObj.DeleteListMember ListName, newMember
   members = sessionObj.GetListMembers(ListName)
   ' print out the list members...
   For Each member In members
      print member
   Next
```

Perl

```
# This example assumes there is at least 1 dynamic list
# in the current database-access session.
$sessionObj = $entity->GetSession();
$sessionObj->UserLogon("admin","","SAMPL","");

# Get a list of the names of Dynamic Lists that exist in this database...
$ListDefNamesREF = $sessionObj->GetListDefNames();

# For one of the lists, print out its members...
$ListName = @$ListDefNamesREF[0];
Print $ListName, "\n";
$members = $sessionObj->GetListMembers($ListName);
foreach $member (@$members){
   print $member, "\n";
   }
# Add a member, then print the list again...
$NewValue = "XYZ";
print "\nAdding member '$NewValue' to list '$ListName'...\n";
$sessionObj->AddListMember($ListName, $NewValue);
$members = $sessionObj->GetListMembers($ListName);
foreach $member (@$members){
   print $member, "\n";
   }
# Remove the item we just added...
print "\nDeleting member '$NewValue' from list '$ListName'...\n";
$sessionObj->DeleteListMember($ListName, $NewValue);
# Print the list again
$members = $sessionObj->GetListMembers($ListName);
```

```
foreach $member (@$members){
   print $member, "\n";
   }
```

See Also

GetListMembers
AddListMember
SetListMembers
GetListDefNames

EditEntity

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 **`GetEntityByDbId`** or **`GetEntity`**, or by running a query. If you created the Entity object using **`BuildEntity`** 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.

Syntax

VBScript

session.**EditEntity** *entity, edit_action_name*

Perl

$*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.

Examples

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"
```

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

`SetFieldValue` of the **`Entity Object`**
`BuildEntity`
`GetEntity`
`GetEntityByDbId`
`Commit` of the **Entity Object**
`Validate` of the **Entity Object**
`GetActionDefNames` of the **Entity Object**
"ActionType Constants" on page 783
"Updating Duplicate Records to Match the Parent Record" on page 820
"Managing Records (Entities) that are Stateless and Stateful" on page 822
"Getting a List of Defects and Modifying a Record" on page 847

FireRecordScriptAlias

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.

ClearQuest has an action type of _RECORD_SCRIPT_ALIAS and if an action is of this type, you can call this method, instead of calling EditEntity, Validate and Commit.

Syntax

VBScript

session.**FireRecordScriptAlias** *entity*, *editActionName*

Perl

$*session***->FireRecordScriptAlias**(*entity*, *editActionName*);

Identifier	Description
session	The Session object that represents the current database-access session.
entity	The entity must be an entity object previously returned by BuildEntity, GetEntityByDbId, or GetEntity.
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
EditEntity
Validate
FireNamedHook
"Notation Conventions for VBScript" on page 3
"Notation Conventions for Perl" on page 2

GetAccessibleDatabases

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, database set name and other information needed to log in to it.

Syntax

VBScript

session.**GetAccessibleDatabases** (*master_db_name, user_login_name, database_set)*

Perl

$*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 ClearQuest logical database name of the *schema repository*. In most cases, this value should be "MASTR".
user_login_name	A String that specifies the user's login. Using an empty string for this argument tells this function to return a list of all databases associated with this schema repository, not just those accessible to a specific user.
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. This causes the function to use the product-default database set name (that is, the product version number).
Return value	For VB, an Array of Variants each containing a **DatabaseDesc Object**. For Perl, a **DatabaseDescs Object** collection.

Example

VBScript

```
set sessionObj = GetSession

' Get the list of databases in the
' master database set.
databases = sessionObj.GetAccessibleDatabases("MASTR","admin","")
for each db in databases
   ' Get the name of the database
   dbName = db.GetDatabaseName
   sessionObj.UserLogon "admin", "", dbName, AD_PRIVATE_SESSION, ""
   dbConnectString = db.GetDatabaseConnectString
Next
```

The following code provides a MsgBox indicating the connect vendor information and logical name for all the user databases in a specific dbset.

```
Sub Test()
Dim session
Dim databases
Dim dbConnectString
Dim dbConnectName
Dim db
Set session = CreateObject("CLEARQUEST.SESSION")
databases = session.GetAccessibleDatabases("MASTR", "admin", "")
session.UserLogon "admin", "", "SAMPL", AD_PRIVATE_SESSION, ""
For Each db In databases
   dbConnectString = db.GetDatabaseConnectString
   dbConnectName = db.GetDatabaseName
   MsgBox dbConnectString
   MsgBox dbConnectName
Next
End Sub
```

Perl

```
use CQPerlExt;
#Start a ClearQuest session
$sessionObj = CQSession::Build();
#Get a list of accessible databases
$databases = $sessionObj->GetAccessibleDatabases("MASTR", "admin", "");
$count = $databases->Count();
#Foreach accessible database, login as joe with password gh36ak3
for($x=0;$x<$count;$x++){
   $db = $databases->Item($x);
   $dbName = $db->GetDatabaseName();
   # Logon to the database
   $sessionObj->UserLogon( "joe", "gh36ak3", $dbName, "" );
   #...
   }
CQSession::Unbuild($sessionObj);
```

See Also

UserLogon
GetDatabaseName of the **DatabaseDesc Object**
GetSessionDatabase

GetAuxEntityDefNames

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.

Syntax

VBScript

`session`.**GetAuxEntityDefNames**

Perl

$session->GetAuxEntityDefNames`();`

Identifier	Description
session	The Session object that represents the current database-access session.
Return value	For Visual Basic, a Variant containing an Array of Strings is returned. Each String contains the name of a stateless record type. For Perl, a reference to an array of strings.

Examples

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
```

Perl

```
$sessionObj = $entity->GetSession();

#Get an array containing the names of the stateless record
#types in the current database's schema.
$AuxEntityDefNames = $sessionObj->GetAuxEntityDefNames();


#Iterate over the state-based record types
foreach $name ( @$AuxEntityDefNames){
   print $name, "\n";
```

```
   $EntityDefObj = $sessionObj->GetEntityDef( $name);
# Do something with the EntityDef object
   # ...
   }
```

See Also

GetEntityDef
GetEntityDefNames
GetQueryEntityDefNames
GetReqEntityDefNames
GetSubmitEntityDefNames

GetClientCodePage

Description

Returns a String describing the client's code page (for example, "1252 (ANSI - Latin I)" or "20127 (US-ASCII)"). This method does not require the Session to be logged in.

Syntax

VBScript

*session.*__GetClientCodePage__

Perl

*$session->*__GetClientCodePage__*();*

Identifier	Description
session	The Session object that represents the current database-access session.
Return value	Returns a String describing the client's code page.

Examples

VBScript

```
myClientCP = session.GetClientCodePage
```

Perl

```
$myClientCP = $session->GetClientCodePage();
```

See Also

CQDataCodePageIsSet
GetCQDataCodePage
IsClientCodePageCompatibleWithCQDataCodePage
IsStringInCQDataCodePage
IsUnsupportedClientCodePage
ValidateStringInCQDataCodePage
CQDataCodePageIsSet of the **AdminSession Object**

GetCQDataCodePage

Description

Returns a String describing the ClearQuest data code page (for example, "1252 (ANSI - Latin I)" or "20127 (US-ASCII)").

Syntax

VBScript

session.**GetCQDataCodePage**

Perl

*$session->***GetCQDataCodePage***();*

Identifier	Description
session	The Session object that represents the current database-access session.
Return value	Returns a String describing the ClearQuest data code page.

Examples

VBScript

```
myCQDataCP = session.GetCQDataCodePage
```

Perl

```
$myCQDataCP = $session->GetCQDataCodePage();
```

See Also

CQDataCodePageIsSet
GetClientCodePage
IsClientCodePageCompatibleWithCQDataCodePage
IsStringInCQDataCodePage
IsUnsupportedClientCodePage
ValidateStringInCQDataCodePage
CQDataCodePageIsSet of the **AdminSession Object**

GetDefaultEntityDef

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 ("").

Syntax

VBScript

session.**GetDefaultEntityDef**

Perl

$session->GetDefaultEntityDef();

Identifier	Description
session	The Session object that represents the current database-access session.
Return value	The default EntityDef object.

Examples

VBScript

```
set sessionObj = GetSession

set defEntityDef = sessionObj.GetDefaultEntityDef
```

Perl

```
#Create a ClearQuest session
$sessionObj = $entity->GetSession();


#Get the default record type of the schema
$defEntityDef = $sessionObj->GetDefaultEntityDef();
```

See Also

GetEntityDef
EntityDef Object

GetDisplayNamesNeedingSiteExtension

Description

Gets the site-extended names of all stateless entities of the specified record type that need site extension.

This method supports MultiSite operations. Its purpose is to avoid name collisions.

For example, at different sites, there may be two users named "Tom", which could cause a name collision in a MultiSite environment. Each user "Tom" can login with their local name, but to be unique in a MultiSite environment, the name of each "Tom" needs to be "site-extended", meaning a site-specific suffix needs to be added. This method returns stateless names that are site-extended.

This method applies only to "stateless" record types, such as the names of users, groups, choice lists, workspaces, queries, reports, charts, and folders.

You can use site-extended names wherever unextended names are used.

Syntax

VBScript

session.**GetDisplayNamesNeedingSiteExtension** *entity_def_name*

Perl

$*session*->**GetDisplayNamesNeedingSiteExtension**(*$entity_def_name*);

Identifier	Description
session	The Session object for the current database session.
entity_def_name	A String containing an EntityDef name (the name of the record type). This is the name returned by the **GetName** in EntityDef.
Return value	In Visual Basic, the return value is a Variant consisting of an array of strings. In Perl, the return value is a reference to an array of strings.

See Also

GetSiteExtendedNames
GetSiteExtendedNames (in Workspace)
GetSiteExtension
GetUnextendedName
IsSiteExtendedName
ParseSiteExtendedName

GetEnabledEntityDefs

Description

Returns the EntityDefs collection object enabled in the current schema for a given package revision.

Use with **`GetEnabledPackageRevs`** to discover which packages and package revisions apply to the current user database. If you pass a package name and a null revision, this method returns the EntityDefs that have the named package installed regardless of revision (as if the revision were a wildcard.)

Syntax

VBScript

session.**GetEnabledEntityDefs** (*packageName, revString)*

Perl

*$session->***GetEnabledEntityDefs(***packageName, revString***);**

Identifier	Description
session	The Session object that represents the current database-access session.
packageName	A String containing a Package name.
revString	A String containing the revision string of the PackageRev.
Return value	The EntityDefs object for the current package revision.

Examples

VBScript

```
Set sessionObj = CreateObject("CLEARQUEST.SESSION")
sessionObj.UserLogon "admin", "", "SAMPL", AD_PRIVATE_SESSION, ""

Set packages = sessionObj.GetEnabledPackageRevs
For each pack in packages
   a = pack.PackageName()
   b = pack.RevString()
   MsgBox (a)
   MsgBox (b)
   Set edefs = sessionObj.GetEnabledEntityDefs(a, b)
   For each edef in edefs
      edefName = edef.GetName()
      MsgBox (edefName)
   Next
Next
```

Perl

```
use CQPerlExt;
#Start a ClearQuest session
```

```
$Session = CQSession::Build();
$Session->UserLogon("admin","","SAMPL","");

$packages = $Session->GetEnabledPackageRevs();
for($x=0;$x<$packages->Count();$x++){
  $pack = $packages->Item($x);
  $a = $pack->GetPackageName();
  $b = $pack->GetRevString();
  print "$a $b\n";
  $edefs = $Session->GetEnabledEntityDefs($a,$b);

  for($y=0;$y<$edefs->Count();$y++){
    $edef = $edefs->Item($y);
    $name = $edef->GetName();
    print "entitydefname:$name\n";
  }
}
CQSession::Unbuild($Session);
```

See Also

GetEnabledPackageRevs
GetEnabledEntityDefs of the SchemaRev Object
PackageRev Object

GetEnabledPackageRevs

Description

Returns the PackageRev set that is enabled in the current revision of the schema.

When you call this method from the Session object, the schema revision is already known when you log onto the user database.

See this method, **`GetEnabledPackageRevs`**, in its other object, **SchemaRev Object**, for an alternative use.

Syntax

VBScript

session.**GetEnabledPackageRevs**

Perl

*$session->*__GetEnabledPackageRevs__*();*

Identifier	Description
session	The Session object that represents the current database-access session.
Return value	The PackageRevs collection object containing the PackageRev set.

Examples

VBScript

```
Set sessionObj = CreateObject("CLEARQUEST.SESSION")
sessionObj.UserLogon "admin", "", "SAMPL", AD_PRIVATE_SESSION, ""
Set packages = sessionObj.GetEnabledPackageRevs
For each pack in packages
   a = pack.PackageName()
   b = pack.RevString()
   MsgBox (a)
   MsgBox (b)
Next
```

Perl

```
use CQPerlExt;
#Start a ClearQuest session
$Session = CQSession::Build();
$Session->UserLogon("admin","","SAMPL","");
$packages = $Session->GetEnabledPackageRevs();
for($x=0;$x<$packages->Count();$x++){
  $pack = $packages->Item($x);
  $a = $pack->GetPackageName();
  $b = $pack->GetRevString();
  print "$a $b\n";
```

```
  $edefs = $Session->GetEnabledEntityDefs($a,$b);
}
CQSession::Unbuild($Session);
```

See Also

GetEntity
GetEnabledPackageRevs in **SchemaRev Object**
PackageRev Object

GetEntity

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 types, 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 **`GetEntityByDbId`**.

Note: To optimize performance, if you want to retrieve some but not all field values for a record, it is more efficient to use queries than calling EditEntity. You can build a query on the State field to avoid getting the whole record.

Syntax

VBScript

session.**GetEntity** (*entity def_name, display_name*)

Perl

*$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 display name of the record. Display name should be either the visible ID (123@SITE) for request entities or the unique key fields for an aux entity.
Return value	An **Entity Object** corresponding to the requested record. Returns an Entity object or one of the following error messages: 1 = AD_ENTITY_NOT_FOUND - Does not exist 3 = AD_ENTITY_HIDDEN - Exists but hidden from current user

Examples

VBScript

```
set sessionObj = GetSession
sessionObj.UserLogon "admin", "", "SAMPL", AD_PRIVATE_SESSION, ""
set record1 = sessionObj.GetEntity("defect", "DEF00013323")
```

Perl

```
#Create a ClearQuest session
$sessionObj = $entity->GetSession();
$sessionObj->UserLogon("admin","","SAMPL","");
#Get record DEF00013323
$record1 = $sessionObj->GetEntity( "defect", "DEF00013323" );
```

See Also

`BuildEntity`
`EditEntity`
`GetEntityByDbId`
`GetFieldValue` of the `Entity Object`
`GetValue` of the **FieldInfo Object**
"Managing Records (Entities) that are Stateless and Stateful" on page 822

GetEntityByDbId

Description

Returns the requested record (Entity) using the record's unique 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.

Syntax

VBScript

session.**GetEntityByDbId** (*entitydef_name, db_id)*

Perl

$*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. The unique id of the record (Entity).
Return value	An **Entity Object** corresponding to the requested record. Returns an Entity object or one of the following error messages: 1 = AD_ENTITY_NOT_FOUND - Does not exist 3 = AD_ENTITY_HIDDEN - Exists but hidden from current user

Examples

VBScript

```
' Save this record's ID for later use.
set sessionObj = GetSession
set record1 = sessionObj.GetEntity("defect", "DEF00013323")

id = record1.GetDbId

' ...
' Get the record again
set record1 = sessionObj.GetEntityByDbId("defect", id)
```

Perl

```
#Assume you have $entityObj, an Entity Object

#Save the session and record id for later use:

$sessionObj = $entityObj->GetSession();

$dbid = $entityObj->GetDbId();

# ...

#Later, to get the record again:

$entityObj = $sessionObj->GetEntityByDbId("defect",$dbid);
```

See Also

`BuildEntity`
`EditEntity`
`GetEntity`
`GetDbId` of the Entity object
"Managing Records (Entities) that are Stateless and Stateful" on page 822

GetEntityDef

Description

Returns the requested 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).

Syntax

VBScript

session.**GetEntityDef** (*entitydef_name*)

Perl

*$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.

Examples

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
```

Perl

```
my($session, $nameList, $field, $entityDefObj, $actionName);
  $session = $entity->GetSession();
  $entityDefObj = $session->GetEntityDef(
                    $entity->GetEntityDefName());

  $session->OutputDebugString("##> Action names for " .
                    $entityDefObj->GetName() . "\n");

$nameList = $entityDefObj->GetActionDefNames();

foreach $actionName(@$nameList)
```

```
{
  $session->OutputDebugString("\t##> $actionName\n");
}
```

See Also

GetAuxEntityDefNames
GetEntityDefNames
GetQueryEntityDefNames
GetReqEntityDefNames
GetSubmitEntityDefNames
EntityDef Object
GetEntityDefFamilyName
GetEntityDefFamilyNames

GetEntityDefFamilyName

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 a family. Otherwise, see the **`IsFamily`** method.

Note: The correct name of this method is different for Perl. See the syntax below.

Syntax

VBScript

`session``.`**GetEntityDefFamilyName** `(`*`entitydefName`*`)`

Perl

`$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.

See Also

`IsFamily`
`GetEntityDefFamilyNames`
`GetIsMaster` to the **DatabaseDesc Object**

GetEntityDefFamilyNames

Description Returns an array that contains the names of all family EntityDefs in the schema repository. Provides support for multitype queries

Syntax

VBScript

session.**GetEntityDefFamilyNames**

Perl

$session->**GetEntityDefFamilyNames**();

Identifier	Description
session	The Session object that represents the current database-access session.
Return value	For Visual Basic, a Variant containing an Array of Strings is returned. Each String contains the name of an EntityDef (record type). For Perl, returns a reference to an array of strings containing the requested EntityDef names.

See Also

IsFamily
GetEntityDefFamilyName
GetEntityDefNames

GetEntityDefNames

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.

Syntax

VBScript

`session``.`**GetEntityDefNames**

Perl

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

Identifier	Description
session	The Session object that represents the current database-access session.
Return value	For Visual Basic, 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 an array of strings is returned.

Examples

VBScript

```
set sessionObj = GetSession

' Get the list of names of all record types.
set 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
```

Perl

```
#Create a ClearQuest session
$sessionObj = $entity->GetSession();


#Get 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`
`GetEntityDef`
`GetQueryEntityDefNames`
`GetReqEntityDefNames`
`GetSubmitEntityDefNames`
EntityDef Object

GetEntityDefOrFamily

Description Returns the named EntityDef object. An EntityDef Family is a special kind of EntityDef.

Syntax

VBScript

session.**GetEntityDefOrFamily** (*name*)

Perl

*$session->***GetEntityDefOrFamily**(*name*);

Identifier	Description
session	The Session object that represents the current database-access session.
name	A String containing the name of an EntityDef object.
Return value	The requested EntityDef object.

See Also **GetAuxEntityDefNames**
GetEntityDef

GetInstalledDbSets

Description

(Perl only) For Visual Basic, see **GetInstalledMasters**.

Returns the list of registered database sets.

The value returned is an array reference. The returned values of **GetInstalledDbSets** and **GetInstalledMasterDbs** always contain the same number of strings. The contents of both are ordered so that each schema repository (master database) listed in masterDBs belongs to the database set at the same index in dbSets.

Syntax

Perl

*$session->***GetInstalledDbSets***();*

Identifier	Description
session	The Session object that represents the current database-access session.
Return value	Returns a reference to an array of strings for the database sets.

Examples

Perl

```
# This program runs in the context of a
# standalone program (not from within a hook)...

use CQPerlExt;
# Create the session object...
$Session = CQSession::Build();
or die "Couldn't create the ClearQuest 'session' object.\n";

# Get the list of master databases and dbsets installed on this
# machine; note that both functions return references to
# arrays...
my($MasterDBsREF) = $Session->GetInstalledMasterDbs();
my(@MasterDBs) = @$MasterDBsREF;
my($DbSetsREF) = $Session->GetInstalledDbSets();
my(@DbSets) = @$DbSetsREF;

my($N) = $#MasterDBs;

printf ("There are %d DbSet(s) installed on this machine.\n", ($N+1));
for (my($i)=0; $i <= $N; $i++) {
print "DbSet #" . $i . ": " .
" DbSet=" . $DbSets[$i] .
" MasterDB=" . $MasterDBs[$i] .
```

```
"\n";
}

CQSession::Unbuild($Session);
```

See Also

`GetInstalledMasterDbs`

`GetIsMaster` of the **DatabaseDesc Object**

GetInstalledMasterDbs

Description

(Perl only) For Visual Basic, see **GetInstalledMasters**.

Returns the list of registered master databases.

The value returned is an array reference. The returned values of **GetInstalledDbSets** and **GetInstalledMasterDbs** always contain the same number of strings. The contents of both are ordered so that each schema repository (master database) listed in masterDBs belongs to the database set at the same index in dbSets.

Syntax

Perl

*$session->***GetInstalledMasterDbs***();*

Identifier	Description
session	The Session object that represents the current database-access session.
Return value	Returns a reference to an array of strings for the master database sets.

Examples

Perl

```
# This program runs in the context of a
# standalone program (not from within a hook)...

use CQPerlExt;

# Create the session object...
$Session = CQSession::Build();
  or die "Couldn't create the ClearQuest 'session' object.\n";

# Get the list of master databases and dbsets installed on this
# machine; note that both functions return references to
# arrays...
my($MasterDBsREF) = $Session->GetInstalledMasterDbs();
my(@MasterDBs) = @$MasterDBsREF;
my($DbSetsREF) = $Session->GetInstalledDbSets();
my(@DbSets) = @$DbSetsREF;

my($N) = $#MasterDBs;

printf ("There are %d DbSet(s) installed on this machine.\n", ($N+1));
for (my($i)=0; $i <= $N; $i++) {
print "DbSet #" . $i . ": " .
```

```
" DbSet=" . $DbSets[$i] .
" MasterDB=" . $MasterDBs[$i] .
"\n";
}

CQSession::Unbuild($Session);
```

See Also **`GetIsMaster`** of the **DatabaseDesc Object**

GetInstalledMasters

Description

(Visual Basic only) For Perl, see **`GetInstalledMasterDbs`** and **`GetInstalledDbSets`**.

Returns the list of registered database sets and master databases.

The values returned are Variants.

Syntax

VBScript

`session.`**GetInstalledMasters** `(dbSets, masterDBs)`

Identifier	Description
session	The Session object that represents the current database-access session.
dbSets	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.
masterDBs	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	Returns the values that are inserted in the empty Variants for the arguments.

Examples

VBScript

```
set sessionObj = GetSession

Dim dbSets, masterDBs

set dbSets = sessionObj.GetInstalledMasters (dbSets, masterDBs)
For Each db in dbSets
   ' ...
   Next
```

See Also

`GetIsMaster` of the **DatabaseDesc Object**

GetListDefNames

Description Returns a list of the dynamic lists in the current database.

Syntax

VBScript

```
sessionObj.GetListDefNames()
```

Perl

```
$sessionObj->GetListDefNames();
```

Identifier	Description
session	The Session object that represents the current database-access session.
Return value	For Visual Basic, a Variant containing an Array whose elements are strings is returned. Each String contains the name of one field. For Perl, a reference to an array of strings is returned

Example

VBScript

```
' This example assumes there is at least 1 dynamic list
' in the current database-access session.
set sessionObj = GetSession
sessionObj.UserLogon "admin", "", "SAMPL", AD_PRIVATE_SESSION, ""

' Get a list of the names of Dynamic Lists that exist in this database...
DynamicListNamesRef = sessionObj.GetListDefNames
' For each of the lists, print out its members...
For Each ListName in DynamicListNamesRef
   print ListName
   ' Then, for each list, get the list members in each list,
   members = sessionObj.GetListMembers(ListName)
   ' print out the list members...
   For Each member In members
      print member
   Next
Next
```

Perl

```
# This example assumes there is at least 1 dynamic list
# in the current database-access session.
$sessionObj = $entity->GetSession();
```

```
$sessionObj->UserLogon("admin","","SAMPL","");


# Get a list of the names of Dynamic Lists that exist in this database...
$ListDefNamesREF = $sessionObj->GetListDefNames();
$NListDefNames = scalar @$ListDefNamesREF;
if ( $NListDefNames == 0) {
    print "\n"
        ."There are no dynamic lists in this database.\n"
        ."Unable to continue.\n"
       ."Re-invoke this program specifying a user database with some dynamic
lists defined.\n";
    exit 1;
} else {
    print "\nThere are $NListDefNames dynamic lists in this database:\n";
    foreach $ListName (@$ListDefNamesREF) {
        print "  '$ListName'\n";
    }
}
# For one of the lists, print out its members...
$ListName = @$ListDefNamesREF[0];
$members = $sessionObj->GetListMembers($ListName);
foreach $member (@$members){
   print $member, "\n";
   }
```

See Also

GetListMembers
AddListMember
DeleteListMember
SetListMembers

GetListMembers

Description Returns the choice values of a dynamic list.

Syntax

VBScript

```
sessionObj.GetListMembers(list_name)
```

Perl

```
$sessionObj->GetListMembers(list_name);
```

Identifier	Description
session	The Session object that represents the current database-access session.
list_name	A String containing the name of the dynamic list.
Return value	For Visual Basic, a Variant containing an Array whose elements are strings is returned. Each String contains a choice list value. For Perl, a reference to an array of strings is returned

Example

VBScript

```
set sessionObj = GetSession
sessionObj.UserLogon "admin", "", "SAMPL", AD_PRIVATE_SESSION, ""
List = sessionObj.GetListMembers("test")
' Get the Count before continuing.
' If the count is 0, specify a user database
' with some dynamic lists defined.
For Each listName In List
MsgBox listName
Next
```

Perl

```
# Perl Example 1
$sessionObj = $entity->GetSession();
$sessionObj->UserLogon("admin","","SAMPL","");
$list = $sessionObj->GetListMembers("test");
# If the count is 0, specify a user database
# with some dynamic lists defined.
foreach $x (@$list){
  print "List:$x\n";
}
# Perl Example 2
# check if a field value is included in a dynamic list
$result = "Invalid HW_Version entered";
```

```
# selected value must be from dynamic list
my $field_value = $entity->GetFieldValue($fieldname)->GetValue();
my $valid_values = $session->GetListMembers("HW_Versions");
foreach (@$valid_values) {
    if ($field_value eq $_) {
        $result = "";
        return $result;
    }
}
return $result;
```

See Also

GetFieldChoiceList
AddListMember
DeleteListMember
SetListMembers
GetListDefNames

GetLocalReplica

Description

Gets replication information and returns an information object.

If your current ClearQuest release supports ClearQuest MultiSite, then this method returns an Entity object of type ratl_replicas.

You can use the returned object to determine whether your local ClearQuest database has been replicated with ClearQuest MultiSite. You can also use this method to find information about current replication, such as the names and locations of replica databases.

The Replica object that this method returns is like any Entity object returned by the GetEntity method on the Session object. This means that you can use any of the methods associated with an Entity object to query the Replica object.

Syntax

VBScript

```
set replicaObj = session.GetLocalReplica
```

Perl

```
$replicaObj = session->GetLocalReplica();
```

Identifier	Description
session	The Session object that represents the current database-access session.
Return value	The "ratl_replicas" Entity object associated with the current session or NULL if the current database has not been updated for ClearQuest MultiSite.

You can create a query against the "ratl_replicas" Entity (which contains the list of replicas known to this database) and compare the "Name" field against replicaName (see example following) to determine if the information applies to the local database or one of the other replicas. Or you can compare the "Host" field to localReplicaHost to determine how you might have to communicate with other programs dealing with the particular replica. For example, if the replica is not local, you might have to use e-mail.

Example

VBScript

```
set session = GetSession
set replicaObj = session.GetLocalReplica
fieldNameList = replicaObj.GetFieldNames

For Each fieldName in fieldNameList
    set fieldInfoObj = GetFieldValue(fieldName)
    fieldType = fieldInfoObj.GetType
    fieldValue = fieldInfoObj.GetValue
    If fieldName = "Name" Then  'replica db name
        If fieldValue = "<local>" Then
            'Database has not been replicated
```

```
        else
            localReplicaName = fieldValue
        End If
    ElseIf fieldName = "Host" Then 'db host name
        'host name of replica database:
        replicaHost = fieldValue
    End If
Next
```

Perl

```
use CQPerlExt;

my $sess;
my $entity;

$sess = CQSession::Build();
$sess->UserLogon("admin", "", "MULTI", "CQMS.MS_ACCESS.SITEA");

if ($sess->IsReplicated()) {
   # print out the local replica name
    $entity = $sess->GetLocalReplica();
    printf "Local replica is %s.\n", $entity->GetDisplayName();
}
CQSession::Unbuild($sess);
```

See Also

GetDisplayNamesNeedingSiteExtension
GetSiteExtendedNames
GetSiteExtendedNames (in Workspace)
GetSiteExtension
GetUnextendedName
IsSiteExtendedName
ParseSiteExtendedName
SiteHasMastership (in Workspace)
SiteHasMastership (in User)

GetMaxCompatibleFeatureLevel

Description Gets the maximum database version number supported by the ClearQuest client running on this machine.

Syntax

VBScript

```
cqdb_max = session.GetMaxCompatibleFeatureLevel
```

Perl

```
$cqdb_max = session->GetMaxCompatibleFeatureLevel();
```

Identifier	Description
session	The Session object that represents the current database-access session.
Return value	The maximum feature level, a Long.

The feature level is read-only.

See Also

DatabaseFeatureLevel
GetMinCompatibleFeatureLevel
GetSessionFeatureLevel

GetMinCompatibleFeatureLevel

Description Gets the minimum database version number supported by the ClearQuest client running on this machine.

Syntax

VBScript

cqdb_min = session.**GetMinCompatibleFeatureLevel**

Perl

$*cqdb_min = session*->**GetMinCompatibleFeatureLevel()**;

Identifier	Description
session	The Session object that represents the current database-access session.
Return value	The minimum feature level, a Long.

The feature level is read-only.

See Also

DatabaseFeatureLevel
GetMaxCompatibleFeatureLevel
GetSessionFeatureLevel

GetProductInfo

Description Returns a CQProductInfo object. Using Perl, you can use this object to retrieve product information.

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

For VBScript, use the following Session object methods:

- GetProductVersion
- GetSuiteVersion
- GetStageLabel

Syntax

Perl

```
session->GetProductInfo();
```

Identifier	Description
session	The Session object that represents the current database-access session.
Return value	Returns a CQProductInfo object.

See Also `ProductInfo Object`

GetProductVersion

Description Returns the internal product version string that is hard-coded in header file.

You don't need to be logged in to a database to use this method.

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

Syntax

VBScript

session.**GetProductVersion**

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

See Also

GetSuiteProductVersion
GetStageLabel

GetQueryEntityDefFamilyNames

Description Returns the names of all family query EntityDefs.

Syntax

VBScript

session.**GetQueryEntityDefFamilyNames**

Perl

$session->**GetQueryEntityDefFamilyNames**();

Identifier	Description
session	The Session object that represents the current database-access session.
Return value	For Visual Basic, 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 an array of strings is returned.

See Also **GetSuiteProductVersion**
GetStageLabel

GetQueryEntityDefNames

Description

Returns 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.

Syntax

VBScript

session.**GetQueryEntityDefNames**

Perl

$session->**GetQueryEntityDefNames**();

Identifier	Description
session	The Session object that represents the current database-access session.
Return value	For Visual Basic, 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 an array of strings is returned.

Examples

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
```

Perl

```
$sessionObj = $entity->GetSession();

# Get the list of names of the record types that support queries.
$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`
`GetAuxEntityDefNames`
`GetEntityDef`
`GetEntityDefNames`
`GetReqEntityDefNames`
`GetSubmitEntityDefNames`
EntityDef Object

GetReqEntityDefNames

Description

Returns 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.

Syntax

VBScript

`session``.`**GetReqEntityDefNames**

Perl

`$sessionObj->``GetReqEntityDefNames();`

Identifier	Description
session	The Session object that represents the current database-access session.
Return value	For Visual Basic, 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 an array of strings is returned.

Examples

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
```

Perl

```
$sessionObj = $entity->GetSession();

#Get the names of the state-based record types.

$entityDefNames = $sessionObj->GetReqEntityDefNames();


#Iterate over the state-based record types

foreach $name ( @$entityDefNames){
```

```
    print $name, "\n";
    $entityDefObj = $session->GetEntityDef( $name);
    # Do something with the EntityDef object
    # ...
}
```

See Also

BuildQuery
GetAuxEntityDefNames
GetEntityDef
GetEntityDefNames
GetQueryEntityDefNames
GetSubmitEntityDefNames
EntityDef Object

GetServerInfo

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.

Note: This method is for COM only. It is not available in the Perl API.

Syntax

VBScript

session.**GetServerInfo**

Identifier	Description
session	The Session object that represents the current database-access session.
Return value	A String identifying the OLE server.

Examples

VBScript

```
set sessionObj = GetSession

serverName = sessionObj.GetServerInfo
```

See Also **GetSessionDatabase**

GetSessionDatabase

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.

Syntax

VBScript

```
databaseDescObj = session.GetSessionDatabase
```

Perl

```
databaseDescObj = $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.

Examples

VBScript

```
set sessionObj = GetSession
set dbDescObj = sessionObj.GetSessionDatabase
currentdbName = dbDescObj.GetDatabaseName
SetFieldValue "headline", currentdbName
```

Perl

```
$sessionObj = $entity->GetSession();
$dbDescObj = $sessionObj->GetSessionDatabase();
$currentdbName = dbDescObj->GetDatabaseName();
$entity->SetFieldValue ("headline", $currentdbName);
```

See Also

GetAccessibleDatabases
DatabaseDesc Object
"Getting Session and Database Information" on page 850
UserLogon

GetSessionFeatureLevel

Description Gets the version number of the ClearQuest client currently running on this machine.

Syntax

VBScript

session.**GetSessionFeatureLevel**

Perl

session->**GetSessionFeatureLevel()**;

Identifier	Description
session	The Session object that represents the current database-access session.
Return value	A Long containing the feature level.

The feature level is read-only.

Examples

VBScript

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

sessionObj.UserLogon "admin", "", "SAMPL", AD_PRIVATE_SESSION, ""


Level = sessionObj.GetSessionFeatureLevel()
MsgBox (Level)
```

Perl

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


$Session->UserLogon("admin","","SAMPL","");


$level = $Session->GetSessionFeatureLevel();
print "Level:$level\n";
CQSession::Unbuild($Session);
```

See Also

DatabaseFeatureLevel
GetMaxCompatibleFeatureLevel
GetMinCompatibleFeatureLevel

GetSiteExtendedNames

Description

Gets site-extended names, given a display name and record type, and returns them in an array.

This method supports MultiSite operations and may be useful in detected or resolving naming conflicts. A *site-extended* name contains a site-specific extension that makes it unique among all names in the MultiSite environment. An extended name can be used in APIs wherever unextended names are used. If the specified name is already an extended name, no error will occur and extended names will still be returned. If the specified name is invalid, the returned array will be empty.

To see whether a name is extended or unextended, use the **`IsSiteExtendedName`** method.

Syntax

VBScript

`session``.`**GetSiteExtendedNames** *`entitydef_name, display_name`*

Perl

`$session->`**GetSiteExtendedNames**`(`*`$entitydef_name, $display_name`*`);`

Identifier	Description
session	The Session object that represents the current database-access session.
entity_def_name	A String containing an EntityDef name (the name of the record type). This is the name returned by the **`GetName`** method in EntityDef.
display_name	A String containing the display name of an Entity.
Return value	For Visual Basic, a Variant containing the site-extended names for the given display name and entity type. For Perl, a reference to an array of strings containing the site-extended names for the given display name and entity type.

See Also

`GetDisplayNamesNeedingSiteExtension`
`GetLocalReplica`
`GetSiteExtension`
`GetSiteExtendedNames` in Workspace
`GetUnextendedName`
`IsSiteExtendedName`
`ParseSiteExtendedName`

GetSiteExtension

Description

Given the display name of a database object, gets the site extension number (the database identifier or dbid) of a replica database and returns it as a Long.

This method supports MultiSite operations. A locally assigned name may not be unique among all names in a MultiSite environment. This method returns a site extension number that can be appended to the display name of a database object to create an extended name. To find ambiguous display names, use the **GetDisplayNamesNeedingSiteExtension** method.

Syntax

VBScript

site_ext_num = *session*.**GetSiteExtension** *display_name*

Perl

$*site_ext_num* = $*session*->**GetSiteExtension**($*display_name*);

Identifier	Description
session	The Session object that represents the current database-access session.
display_name	A String containing the display name of a database object as returned by **GetDisplayNamesNeedingSiteExtension**.
Return value	A Long containing the site extension number or zero (0) if not found.

See Also

GetDisplayNamesNeedingSiteExtension
GetLocalReplica
GetSiteExtendedNames
GetUnextendedName
IsSiteExtendedName
ParseSiteExtendedName

GetStageLabel

Description Returns stage label used for the build; the stage label is dynamically generated for each build.

You don't need to be logged in to a database to use this method.

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

Syntax

VBScript

session.**GetStageLabel**

Identifier	Description
session	The Session object that represents the current database-access session.
Return value	A String containing the stage label used for the build.

See Also

`GetProductVersion`
`GetSuiteProductVersion`

GetSubmitEntityDefNames

Description

Returns 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.

Syntax

VBScript

session.**GetSubmitEntityDefNames**

Perl

$session->**GetSubmitEntityDefNames**();

Identifier	Description
session	The Session object that represents the current database-access session.
Return value	For Visual Basic, 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 an array of strings is returned.

Examples

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
```

Perl

```
#Create a ClearQuest session
$sessionObj = $entity->GetSession();


$entityDefNames = $sessionObj->GetSubmitEntityDefNames();
```

```
#Iterate over the suitable record types

foreach $name (@$entityDefNames){
   $entityDefObj = $sessionObj->GetEntityDef( $name );

   #Do something with the EntityDef object

   # ...
   }
```

See Also

GetAuxEntityDefNames
GetEntityDef
GetEntityDefNames
GetQueryEntityDefNames
GetReqEntityDefNames
EntityDef Object

GetSuiteProductVersion

Description

Returns the Suite version string. This is the same version string as the one returned by the Suite version dll and displayed in the About box of ClearQuest.

You don't need to be logged in to a database to use this method.

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

Syntax

VBScript

session.**GetSuiteProductVersion**

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

See Also

`GetProductVersion`

GetUnextendedName

Description

Gets the unextended display name of a specified database object.

This method is useful in a MultiSite environment. It parses an extended name and returns the unextended name. An *extended* name contains a site-specific extension that makes it unique among all names in a MultiSite environment. An *unextended* name is known to be unique only to its site. If the specified name is already an unextended name, then no error will occur, and you will just receive the same name. If the specified name is not a valid name, you will receive an empty string.

Syntax

VBScript

```
unextended_name = session.GetUnextendedName extended_name
```

Perl

```
$unextended_name = $session->GetUnextendedName($extended_name);
```

Identifier	Description
session	The Session object that represents the current database-access session.
extended_name	A String containing the extended name of a database object.
Return value	A String containing the unextended name or empty if the input name is not valid.

See Also

GetDisplayNamesNeedingSiteExtension
GetLocalReplica
GetSiteExtendedNames
GetSiteExtendedNames (in Workspace)
GetSiteExtension
IsSiteExtendedName
ParseSiteExtendedName
SiteHasMastership (in Workspace)
SiteHasMastership (in User)

GetUserEmail

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 e-mail address using the schema repository object User. Simply assign a new value to the e-mail property of User.

Syntax

VBScript

session.**GetUserEmail**

Perl

$session->**GetUserEmail**();

Identifier	Description
session	The Session object that represents the current database-access session.
Return value	A String containing the e-mail address of the user who is logged in for this session.

Examples

VBScript

```
set sessionObj = GetSession

' Get the user's personal information
userName = sessionObj.GetUserFullName
userLogin = sessionObj.GetUserLoginName
userEmail = sessionObj.GetUserEmail
```

Perl

```
#Create a ClearQuest session
$sessionObj = $entity->GetSession();


#Get the user's personal information
$userName = $sessionObj->GetUserFullName();
$userLogin = $sessionObj->GetUserLoginName();
$userEmail = $sessionObj->GetUserEmail();
```

See Also

GetUserFullName
GetUserGroups
GetUserLoginName
GetUserMiscInfo
GetUserPhone
Email of the **User Object**
"Getting Session and Database Information" on page 850

GetUserFullName

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.

Syntax

VBScript

session.**GetUserFullName**

Perl

$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.

Examples

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
```

Perl

```
#Create 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

GetUserGroups
GetUserLoginName
GetUserMiscInfo
GetUserPhone
Fullname of the **User Object**
"Getting Session and Database Information" on page 850

GetUserGroups

Description

Returns a list of active user groups to which the current user belongs.

The returned list can be empty.

Syntax

VBScript

session.**GetUserGroups**

Perl

$session->**GetUserGroups**();

Identifier	Description
session	The Session object that represents the current database-access session.
Return value	For Visual Basic, a Variant containing an array String of Variants is returned. Each String names an active 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 an array of strings is returned.

Examples

VBScript

```
set sessionObj = GetSession

' Iterate over the user's groups
userGroups = sessionObj.GetUserGroups
If IsEmpty(userGroups) Then
   ' Code to handle if no user groups exist
Else
   For Each group in userGroups
      ' ...
Next
```

Perl

```
use strict;
use CQPerlExt;


# Create session object
my $sessionObj = CQSession::Build();
$sessionObj->UserLogon("user", "password", "SAMPL", "");


# get the user groups
my $userGroups = $sessionObj->GetUserGroups();


if (!@$userGroups) {
```

```
    #Code to handle if no user groups exist
    print "no user groups\n";
}
else {
    # print out all groups
    foreach my $group (@$userGroups) {
        print "Group $group\n";
    }
}
exit(0);
CQSession::Unbuild($sessionObj);
```

See Also

GetUserFullName
GetUserLoginName
GetUserMiscInfo
GetUserPhone
AddUser method of the **Group Object**
"Getting Session and Database Information" on page 850

GetUserLoginName

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.

Syntax

VBScript

session.**GetUserLoginName**

Perl

$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.

Examples

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
```

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

GetUserFullName
GetUserGroups
GetUserMiscInfo

`GetUserPhone`
User Object
“Getting Session and Database Information” on page 850

GetUserMiscInfo

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, e-mail 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.

Syntax

VBScript

session.**GetUserMiscInfo**

Perl

$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.

Examples

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
```

Perl

```
#Create 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

`GetUserFullName`
`GetUserGroups`
`GetUserLoginName`
`GetUserPhone`
`MiscInfo` of the **User Object**
"Getting Session and Database Information" on page 850

GetUserPhone

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.

Syntax

VBScript

session.**GetUserPhone**

Perl

$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.

Examples

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
```

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

GetUserFullName
GetUserGroups
GetUserLoginName

`GetUserMiscInfo`
`Phone` of the **User Object**
"Getting Session and Database Information" on page 850

GetWorkSpace

Description

Returns the session's Workspace object.

You can use the Workspace object to manipulate saved queries, charts, and reports in the ClearQuest workspace.

Syntax

VBScript

session.**GetWorkSpace**

Perl

$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.

Examples

VBScript

```
set sessionObj = GetSession

' Get the workspace for manipulating query, chart, and report info.
set wkSpc = sessionObj.GetWorkSpace
```

Perl

```
#Get a ClearQuest session
$sessionObj = $entity->GetSession();


#Get the workspace for manipulating query, chart, and report
$MyWorkSpace = $sessionObj->GetWorkSpace();


#Get a list of queries in the workspace...
$MyQueriesListREF = $MyWorkSpace->GetAllQueriesList();
foreach (@$MyQueriesListREF) {
  print ("$_\n");
}
#The QueryDef object contains information about a workspace
#query, including the query name and the SQL string used
#to execute the query.
foreach $QueryName (@$MyQueriesListREF) {
  # Get the QueryDef associated with that query...
  $QueryDef = $MyWorkSpace->GetQueryDef($QueryName);
  # Build the ResultSet object to hold the results of
```

```
  # the query...
  $ResultSet = $Session->BuildResultSet($QueryDef);
  # Execute the query...
  $ResultSet->Execute();
  # Get the query's short name (without the pathname)...
  @QueryPath = split('/', $QueryName);
  $QueryShortName = @QueryPath[$#QueryPath];
  # Process/display the results of the query...
  print "\n" if ($PrintDetails);
  print "$QueryShortName:  ";
  for ($N = 0; (($ResultSet->MoveNext()) ==
  $CQPerlExt::CQ_SUCCESS); $N++) {
    if ($PrintDetails) {
      printresultrow();
    }
  }
  print "$N\n";
}
```

See Also **Workspace Object**

HasUserPrivilege

Description

Tests a user privilege level and, for the specified privilege, returns Boolean True if the current user has the privilege and otherwise returns False. Use this method to determine whether the user has the privilege to perform the specific task.

Data access, reporting, and management can be controlled at the database, record type (EntityDef), and field (column) levels. This method tests privileges related to record types and fields. To manage security at the record type and field levels, both your ClearQuest client and the session database must support security privileges.

You can test the user privileges by specifying one of the **`UserPrivilegeMaskType Constants`**.

Syntax

VBScript

`session``.`**HasUserPrivilege**`(`*`priv_mask`*`)`

Perl

`session``->`**HasUserPrivilege**`($`*`priv_mask`*`);`

Identifier	Description
session	The Session object that represents the current database-access session.
priv_mask	A UserPrivilegeMaskType enumerated constant (which is a Long) specifying the privilege to test.
Return value	A Boolean True if the current user has the specified privilege; otherwise, the return value is Boolean False.

Examples

VBScript

```
has_privilege = session.HasUserPrivilege AD_SUPER_USER
```

Perl

```
$has_privilege = $session->HasUserPrivilege (CQPerlExt::CQ_SUPER_USER);
```

See Also

`UserPrivilegeMaskType Constants`
`IsRestrictedUser`
`IsUserAppBuilder`
`IsUserSuperUser`
`SetRestrictedUser`

HasValue

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.

Syntax

VBScript

session.**HasValue** (*name*)

Perl

$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	Returns Boolean True if the variable exists in this session, otherwise False.

Examples

VBScript

```
set sessionObj = GetSession
' Test for existence of the web session variable
if sessionObj.HasValue ("_CQ_WEB_SESSION") then
   Value = sessionObj.NameValue("_CQ_WEB_SESSION")
   ' Either exit or do something else
end if
```

Perl

```
# Get a ClearQuest session
$sessionObj = $entity->GetSession();
if ( $sessionObj->HasValue( "foo" ) ) {
   #Get the old value of the session variable "foo"
   $fooValue = $sessionObj->NameValue( "foo" );
}
```

See Also

NameValue

Using Sessionwide Variables **on page 13**

IsClientCodePageCompatibleWithCQDataCodePage

Description

Returns whether the client code page is compatible with the ClearQuest data code page, or not. This returns True if all characters in the ClearQuest data code page are present in the client code page. If this method returns False, ClearQuest will not let you edit anything in the database (that is, you will have read-only access).

Syntax

VBScript

*session.***IsClientCodePageCompatibleWithCQDataCodePage**

Perl

*$session->***IsClientCodePageCompatibleWithCQDataCodePage** *();*

Identifier	Description
session	The Session object that represents the current database-access session.
Return value	Returns True if the client code page is compatible with the ClearQuest data code page, False otherwise.

Examples

VBScript

```
isComp = session.IsClientCodePageCompatibleWithCQDataCodePage
```

Perl

```
$ isComp = $session->IsClientCodePageCompatibleWithCQDataCodePage();
```

See Also

CQDataCodePageIsSet
GetClientCodePage
GetCQDataCodePage
IsStringInCQDataCodePage
IsUnsupportedClientCodePage
ValidateStringInCQDataCodePage
CQDataCodePageIsSet of the **AdminSession Object**

IsMetadataReadonly

Description Returns a Bool indicating whether the session's metadata is read-only.

Syntax

VBScript

session.**IsMetadataReadonly**

Perl

*$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.

Examples

VBScript

```
set sessionObj = GetSession

If sessionObj.IsMetadataReadonly Then
   ' ...
End If
```

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**

IsMultisiteActivated

Description Returns TRUE if the current database has been activated for multisite operations. This method returns TRUE even the current database is the only exisiting replica.

Syntax

VBScript

*session.***IsMultisiteActivated**

Perl

*$session->***IsMultisiteActivated** *();*

Identifier	Description
session	The Session object that represents the current database-access session.
Return value	Returns True if the current database has been activated for multisite operations.

Examples

Perl

```
use CQPerlExt;

my $sess;
my $entity;

$sess = CQSession::Build();
$sess->UserLogon("admin", "", "MULTI", "CQMS.MS_ACCESS.SITEA");

if ($sess->IsMultisiteActivated()) {
   # print out list of available replicas
   my $qryDef;
    my $resultSet;
    my $status;

    $qryDef = $sess->BuildQuery("ratl_replicas");
    $qryDef->BuildField("name");
    $qryDef->BuildField("clan");
    $qryDef->BuildField("family");

    $resultSet = $sess->BuildResultSet($qryDef);
    $resultSet->Execute();
    $status = $resultSet->MoveNext();
```

```
   printf ("\nname\tclan\tfamily\n");
    while($status == $CQPerlExt::CQ_SUCCESS) {
     printf("%s\t%s\t%s\n",
        $resultSet->GetColumnValue(1),
        $resultSet->GetColumnValue(2),
        $resultSet->GetColumnValue(3));
    $status = $resultSet->MoveNext();
   }
}
CQSession::Unbuild($sess);
```

See Also

IsReplicated

IsPackageUpgradeNeeded

Description

Returns TRUE if the current revision of package that is applied to the schema isn't the highest package revision that is available for the package. Typically, package upgrade is needed when the current revision is no more the latest one. The API also returns more information to the caller in the form of current_rev and highest_rev which specify the current package revision and the highest available package revision that should be applied.

Syntax

VBScript

```
session.IsPackageUpgradeNeeded package_name, current_rev, highest_rev
```

Perl

```
$session->IsPackageUpgradeNeeded(package_name, current_rev, highest_rev);
```

Identifier	Description
session	The Session object that represents the current database-access session.
package_name	A String containing the package name.
current_rev	A reference to the current package revision.
highest_rev	A reference to the latest available package revision.
Return value	Returns Boolean True if the current revision of package that is applied to the schema isn't the highest package revision that is available for the package.

See Also

GetEnabledPackageRevs
PackageRev Object
Schema Object

IsReplicated

Description Returns True if the current database has at lease two replicated sites.

Syntax

VBScript

*session.*__IsReplicated__

Perl

*$session->*__IsReplicated__*();*

Identifier	Description
session	The Session object that represents the current database-access session.
userName	A String containing the name you want to give to the new user.
Return value	Returns True if the current database has at lease two replicated sites.

Examples

Perl

```
use CQPerlExt;

my $sess;
my $entity;

$sess = CQSession::Build();
$sess->UserLogon("admin", "", "MULTI", "CQMS.MS_ACCESS.SITEA");

if ($sess->IsReplicated()) {
   # print out the local replica name
    $entity = $sess->GetLocalReplica();
    printf "Local replica is %s.\n", $entity->GetDisplayName();
}
CQSession::Unbuild($sess);
```

See Also **IsMultisiteActivated**

IsRestrictedUser

Description

Tests whether the current user has restricted access and action privileges and returns Boolean True if privileges are restricted and otherwise returns False. Is user is a Superuser, this method returns False.

Data access, reporting, and management can be controlled at the database, record type, and field levels. This method tests privileges related to record types and fields. To manage security at the record type and field levels, both your ClearQuest client and the session database must support security privileges. In general, ClearQuest supports the following user roles: Active User, Schema Designer, User Administrator, and Super User. (See "ClearQuest user privileges" in the Administering Rational ClearQuest manual for more information on roles.) This method refers to record type and field privileges, not roles.

If you have restrictions on your record type and field privileges, you can test your privilege levels with **`HasUserPrivilege`**. You can also use action and field hooks to programmatically control who can change record and field values.

For example, by default an *Active User* can see all records in a database, and a *Schema Designer* can create public queries and reports. However, a *Super User* can selectively revoke these privileges on specific record types or fields within a record type.

Note: This method is for COM only. It is not available in the Perl API.

Syntax

VBScript

`session``.`**IsRestrictedUser**

Identifier	Description
session	The Session object that represents the current database-access session.
Return value	A Boolean True if the user has restrictions and otherwise is False.

See Also

“UserPrivilegeMaskType Constants” on page 801

`HasUserPrivilege`
`IsUserAppBuilder`
`IsUserSuperUser`
`SetRestrictedUser`
`IsMetadataReadonly`

IsSiteExtendedName

Description

Tests whether the specified name is an extended name and returns Boolean True if it is an extended name and otherwise returns False.

This method supports MultiSite operations. Its purpose is to avoid name collisions. This method tests whether a name is an extended name, meaning that it has a *site-extension* identifier that makes it unique among all names in the MultiSite environment. If the name is not extended, you can use the **GetSiteExtendedNames** method to get the extended name. You can use the **GetDisplayNamesNeedingSiteExtension** method to get all names needing a site extension.

Syntax

VBScript

```
valid_path = session.IsSiteExtendedName display_name
```

Perl

```
$valid_path = $session->IsSiteExtendedName($display_name);
```

Identifier	Description
session	The Session object that represents the current database-access session.
db_display_name	A String containing an entity display name.
Return value	A Boolean, True or False.

See Also

GetDisplayNamesNeedingSiteExtension
GetSiteExtendedNames
GetSiteExtendedNames (in Workspace)
GetSiteExtension
GetUnextendedName
ParseSiteExtendedName

IsStringInCQDataCodePage

Description

Returns whether, or not, the ClearQuest data code page contains a given String.

This method takes a String argument and checks to see if the String is in the ClearQuest data code page for the Session's schema-repository.

Syntax

VBScript

session.**IsStringInCQDataCodePage** *stringToCheck*

Perl

*$session->***IsStringInCQDataCodePage** *($stringToCheck);*

Identifier	Description
session	The Session object that represents the current database-access session.
stringToCheck	A String that specifies what you are checking to see is in the ClearQuest data code page.
Return value	Returns True if the ClearQuest data code page contains a given String, False otherwise.

Examples

VBScript

```
IsInCodePage = session.IsStringInCQDataCodePage stringToCheck
```

Perl

```
$isInCodePage = $session->IsStringInCQDataCodePage($stringToCheck);
```

See Also

CQDataCodePageIsSet
GetClientCodePage
GetCQDataCodePage
IsClientCodePageCompatibleWithCQDataCodePage
IsUnsupportedClientCodePage
ValidateStringInCQDataCodePage
CQDataCodePageIsSet of the **AdminSession Object**

IsUnix

Description Returns True if ClearQuest is running on a UNIX machine and False if it is running on Windows.

Note: This method is only available for Perl. It is not available in the COM API.

Syntax

Perl

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

Identifier	Description
session	The Session object that represents the current database-access session.
Return value	A Boolean, True or False.

Examples

Perl

```
$is_unix_flag = $my_session->IsUnix();
```

See Also `IsWindows`

IsUnsupportedClientCodePage

Description

Returns whether the client code page is unsupported, or not. You do not need to login to use this method.

Note: The client code page is separate from the ClearQuest data code page.

Syntax

VBScript

session.`IsUnsupportedClientCodePage`

Perl

$session->`IsUnsupportedClientCodePage` *();*

Identifier	Description
session	The Session object that represents the current database-access session.
Return value	Returns True if the client code page is unsupported, False otherwise.

Examples

VBScript

```
isUnsupported = session.IsUnsupportedClientCodePage
```

Perl

```
$isUnsupported = $session->IsUnsupportedClientCodePage();
```

See Also

CQDataCodePageIsSet
GetClientCodePage
GetCQDataCodePage
IsClientCodePageCompatibleWithCQDataCodePage
IsStringInCQDataCodePage
ValidateStringInCQDataCodePage
CQDataCodePageIsSet of the **AdminSession Object**

IsUserAppBuilder

Description Tests whether the current user has *Schema Designer* privileges to create hooks and applications that run against all databases associated with the session database. Returns Boolean True if the user has application privileges and otherwise returns False.

Syntax

VBScript

```
app_builder = session.IsUserAppBuilder
```

Perl

```
$app_builder = $session->IsUserAppBuilder();
```

Identifier	Description
session	The Session object that represents the current database-access session.
Return value	A Boolean, True or False.

See Also

"UserPrivilegeMaskType Constants" on page 801

HasUserPrivilege
IsRestrictedUser
IsUserSuperUser
SetRestrictedUser

IsUserSuperUser

Description

Tests whether the current user has *Super User* privileges. Returns Boolean True if the user is a Super User and otherwise returns False.

You can use this method for action access hooks, rather than querying a database to find out if the named user is a SuperUser.

Syntax

VBScript

```
super_user = session.IsUserSuperUser
```

Perl

```
$super_user = $session->IsUserSuperUser();
```

Identifier	Description
session	The Session object that represents the current database-access session.
Return value	A Boolean, True or False.

See Also

"UserPrivilegeMaskType Constants" on page 801

HasUserPrivilege
IsRestrictedUser
IsUserAppBuilder
SetRestrictedUser

IsWindows

Description Returns True if ClearQuest is running on a Windows machine and False otherwise.

Note: This method is only available for Perl. It is not available in the COM API.

Syntax

Perl

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

Identifier	Description
session	The Session object that represents the current database-access session.
Return value	A Boolean, True or False.

Examples

Perl

```
$is_windows_flag = $my_session->IsWindows();
```

See Also **IsUnix**

LoadEntity

Description

Returns the specified record with latest database values.

This method is the same as **`GetEntity`**, except that it ensures that you are using the latest values in the database.

Returns a different error message if record is hidden from the user vs record doesn't exist

Syntax

VBScript

`session``.`**LoadEntity** `(`*`entity def_name, display_name`*`)`

Perl

`$`*`session`*`->`**LoadEntity**`(`*`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 containing the display name of the record
Return value	An **Entity Object** corresponding to the requested record. Returns an Entity object or one of the following error messages: 1 = AD_ENTITY_NOT_FOUND - Does not exist 3 = AD_ENTITY_HIDDEN - Exists but hidden from current user

Examples

VBScript

```
set entityObj = session.LoadEntity("defect", "Sampl00000001");
```

Perl

```
$entityObj = $session->LoadEntity("defect", "Sampl00000001");
```

See Also

`GetEntity`

LoadEntityByDbId

Description

Returns the record with the specified database ID and the latest database values.

This method is the same as **`GetEntityByDbId`**, except that it ensures that you are using the latest values in the database.

Returns a different error message if record is hidden from the user vs record doesn't exist

Syntax

VBScript

session.**LoadEntityByDbId** (*entitydef_name*, *db_id*)

Perl

$*entity*->**LoadEntityByDbId**(*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 (EntityDef) to which the desired record belongs.
db_id	A Long that is the number used by the database to identify the record. The unique ID of the record
Return value	An **Entity Object** corresponding to the requested record. Returns an Entity object or one of the following error messages: 1 = AD_ENTITY_NOT_FOUND - Does not exist 3 = AD_ENTITY_HIDDEN - Exists but hidden from current user

Examples

VBScript

```
set entityObj = session.LoadEntityByDbId("defect", 33554433)
```

Perl

```
$entityObj = $session->LoadEntityByDbId("defect", 33554433);
```

See Also

`GetEntityByDbId`

MarkEntityAsDuplicate

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.

Syntax

VBScript

session.**MarkEntityAsDuplicate** *duplicate, original, duplicate_action_name*

Perl

*$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.

Examples

VBScript

```
set sessionObj = GetSession
idName = GetFieldValue("id").GetValue
set currentObj = sessionObj.GetEntity("defect", idName)

' Mark the entity with ID="SAMPL00000031" as a duplicate of this entity.
' Use the action named "duplicate".
set dupEntityObj = sessionObj.GetEntity("defect", "SAMPL00000031")
sessionObj.MarkEntityAsDuplicate dupEntityObj, currentObj, "duplicate"

' Validate and commit the duplicate entity since it
' is currently modifiable.
error = dupEntityObj.Validate
if error = "" then
  dupEntityObj.Commit
End If
```

Perl

```
#Get a ClearQuest session
$sessionObj = $entity->GetSession();

#Mark the entity with ID="SAMPL00000031" as a duplicate of this
#entity. Use the action named "duplicate".
$dupEntityObj = $sessionObj->GetEntity("defect", "SAMPL00000031");
$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
"Notation Conventions for VBScript" on page 3
"Notation Conventions for Perl" on page 2

OpenQueryDef

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.

Syntax

VBScript

session.**OpenQueryDef** (*filename*)

Perl

$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.

Examples

VBScript

```
set sessionobj = GetSession

' Get the query from file "C:\queries\myQuery.txt"
set queryDefObj = sessionObj.OpenQueryDef("C:\queries\myQuery.txt")
```

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

Description

Specifies a message that can be displayed by a debugger or a similar tool.

The argument value of this method is passed to the Win32 API call OutputDebugString. 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.

The Windows debugging utility dbwin32.exe is included with ClearQuest. It is located in the ClearQuest installation directory. When dbwin32.exe is active, it displays all the messages generated by OutputDebugString.

After you start dbwin32.exe, insert the OutputDebugString method wherever you want to have a text string output in your hook code. Then, execute the hook or script you created or modified. After executing the hook, use the information in the DBWIN32 console to assess whether you have a bug and how to correct it.

Syntax

VBScript

session.**OutputDebugString** *debugString*

Perl

$*session*->**OutputDebugString**(*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.

Examples

VBScript

```
set sessionObj = GetSession
sessionObj.OutputDebugString "This is a test message."
```

Perl

```
#Get a ClearQuest session
$sessionObj = $entity->GetSession();


#Display a debug string via a debugger
$sessionObj->OutputDebugString("This is a test message.");
```

See Also

"Debugging Your Code" on page 4

ParseSiteExtendedName

Description

Splits the name of a database object into an unextended name and a site extension. Returns true if successful and otherwise returns False.

Note: This method is for COM only. It is not available in the Perl API.

If the specified name is an unextended name, the returned name will be the same as the specified name. This method supports MultiSite operations and may be useful in detected or resolving naming conflicts. A *site-extended* name contains a site-specific extension that makes it unique among all names in the MultiSite environment. An *unextended* name is the name used when the object was created and is only guaranteed to be unique to its site.

Syntax

VBScript

session.**ParseSiteExtendedName** *name*, *return_unextended_name*, *return_dbid*

Identifier	Description
session	The Session object that represents the current database-access session.
name	A String containing the name of a database object.
return_unextended_name	In Visual Basic, a Variant containing the returned, unextended name of the database object.
return_dbid	In Visual Basic, a Variant containing the returned database identifier.
Return value	Boolean True if successful and otherwise False.

See Also

GetDisplayNamesNeedingSiteExtension
GetLocalReplica
GetSiteExtendedNames
GetSiteExtendedNames (in Workspace)
GetSiteExtension
GetUnextendedName
IsSiteExtendedName

SetListMembers

Description

Sets the members in a named list.

For Perl, note that this parameter is an array, so no delimiter is needed to separate member items.

Syntax

VBScript

```
session.SetListMembers listName, Members
```

Perl

```
$session->SetListMembers(listName, Members);
```

Identifier	Description
session	The Session object that represents the current database-access session.
listName	A String containing the name of the list.
Members	For VBScript, a Variant array of strings containing the members of the list. For Perl, a reference to an array of Strings containing the members of the list.
Return value	None.

Examples

VBScript

```
' This example assumes there is at least 1 dynamic list
' in the current database-access session.
set sessionObj = GetSession
sessionObj.UserLogon "admin", "", "SAMPL", AD_PRIVATE_SESSION, ""
Dim NewValues(2)
    NewValues(0) = "ABC"
    NewValues(1) = "123"
    NewValues(2) = "XYZ"
DynamicListNamesRef = sessionObj.GetListDefNames
set ListName = DynamicListNamesRef(0)
print ListName
sessionObj.SetListMembers ListName, NewValues

members = sessionObj.GetListMembers(ListName)
' print out the list members...
For Each member In members
   print member
Next
```

Perl

```
# This example assumes there is at least 1 dynamic list
# in the current database-access session.
$sessionObj = $entity->GetSession();
$sessionObj->UserLogon("admin","","SAMPL","");

# Get a list of the names of Dynamic Lists that exist in this database...
$ListDefNamesREF = $sessionObj->GetListDefNames();
$ListName = @$ListDefNamesREF[0];
# Use SetListMembers() to set the list to a specific list of values...
print "\nSetting list '$ListName' to ('ABC', '123', 'XYZ')...\n";
@NewValues = ('ABC', '123', 'XYZ');
$sessionObj->SetListMembers($ListName, \@NewValues);
$members = $sessionObj->GetListMembers($ListName);
#print out the list members
foreach $member (@$members){
   print $member, "\n";
   }
```

See Also

GetListMembers
AddListMember
DeleteListMember
GetListDefNames

SetRestrictedUser

Description

For current user, restricts access and action privileges related to lists, public folders, security, user information, and multisite administration. This method returns Boolean True if privileges are restricted and otherwise returns False.

Note: This method is for COM only. It is not available in the Perl API.

Syntax

VBScript

```
user_restr = session.SetRestrictedUser
```

Identifier	Description
session	The Session object that represents the current database-access session.
Return value	A Boolean True if privileges are restricted and otherwise returns False.

See Also

IsRestrictedUser
"UserPrivilegeMaskType Constants" on page 801
HasUserPrivilege
IsUserAppBuilder
IsUserSuperUser

StringIdToDbId

Description

Returns the dbid number translated from string ID.

Dbid is a unique number assigned to every record by ClearQuest.

For stateful records, the string id is the display name (for example, RAMBU00001234). For stateless records, the display name is composed of a concatenation of all the unique key field values with a space character between.

For example, if a project record type has two key fields, name and department and they are both designated as unique key fields, the string id would be "<name> <department>".

For a project with name "ACME" and department "Finance" the string ID is "ACME Finance".

Syntax

VBScript

*session.*__StringIdToDbId__ string_id

Perl

*$session->*__StringIdToDbId__*(string_id);*

Identifier	Description
session	The Session object that represents the current database-access session.
string_id	A String containing a record string id (display name)
Return value	Returns a Long containing the record dbid.

Examples

VBScript

```
dbid = session.StringIdToDbId "RAMBU00001234"
```

Perl

```
$dbid = $session->StringIdToDbId ("RAMBU00001234");
```

See Also

DbIdToStringId
GetEntityDefNames

Unbuild

Description Deletes the Session object you explicitly created with the Build method, when you do not need it anymore.

Note: This method is for Perl only.

Syntax

Perl

```
CQSession::Unbuild(SessionObj);
```

Identifier	Description
CQSession	A static function specifier for the Session object.
SessionObj	The Session object that you are deleting.
Return value	None.

Example

Perl

```
use CQPerlExt;

my $sessionObj = CQSession::Build();

CQSession::Unbuild($sessionObj);
```

See Also

Build
AdminSession Object

UnmarkEntityAsDuplicate

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`** and **`Commit`** methods, respectively.

Syntax

VBScript

session.**UnmarkEntityAsDuplicate** *duplicate, action_name*

Perl

*$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.

Examples

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
```

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`
`Validate` of the **Entity Object**
"Notation Conventions for VBScript" on page 3
"Notation Conventions for Perl" on page 2

UserLogon

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.

Note: You must log in with superuser privilege or an error will be generated by the DatabaseDesc object's GetDatabaseConnectString 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.

Syntax

VBScript

```
session.UserLogon login_name, password, database_name,
   session_type, database_set
```

Perl

```
$session->UserLogon(login_name, password, database_name,
   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.)
session_type	(VBScript Only) A SessionType enumerated constant (use `AD_PRIVATE_SESSION`). Perl does not recognize SessionType constants.
database_set	A String that specifies the name of the master database. You should set this string to the empty string ("") unless you have more than one database set. 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.
Return value	None.

Examples

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.
databases = sessionObj.GetAccessibleDatabases("MASTR","admin","")
For Each db in databases
   dbName = db.GetDatabaseName
   sessionObj.UserLogon "admin", "", dbName, AD_PRIVATE_SESSION, ""
```

```
' Access the database
   ' ...
Next
```

Perl

```
use CQPerlExt;

#Start a ClearQuest session

$sessionObj = CQSession::Build();

#Get a list of accessible databases

$databases = $sessionObj->GetAccessibleDatabases("MASTR", "admin", "");

$count = $databases->Count();

#Foreach accessible database, login as joe with password gh36ak3

for($x=0;$x<$count;$x++){
   $db = $databases->Item($x);
   $dbName = $db->GetDatabaseName();
   # Logon to the database
   $sessionObj->UserLogon( "joe", "gh36ak3", $dbName, "" );
   #...
   }

CQSession::Unbuild($sessionObj);
```

See Also

`GetDatabaseConnectString` of the DatabaseDesc Object
`GetAccessibleDatabases`
`GetSessionDatabase`
"Getting Session and Database Information" on page 850
`SessionType Constants`
"Notation Conventions for VBScript" on page 3

ValidateStringInCQDataCodePage

Description

Checks to see if a given String is in the ClearQuest data code page for the session's schema-repository. If the String is not in the code page, it returns an error message for display to the user. The error message includes which characters (up to the first five characters) were not in the ClearQuest data code page.

Syntax

VBScript

session.**ValidateStringInCQDataCodePage** *stringToCheck*

Perl

*$session->***ValidateStringInCQDataCodePage** *($stringToCheck);*

Identifier	Description
session	The Session object that represents the current database-access session.
stringToCheck	A String that specifies what you are checking to see is in the ClearQuest data code page.
Return value	Returns an empty String if the *stringToCheck* is valid, otherwise, it returns a displayable error message

Examples

VBScript

```
validation = session.ValidateStringInCQDataCodePage stringToCheck
```

Perl

```
$validation = $session->ValidateStringInCQDataCodePage ($stringToCheck);
```

See Also

CQDataCodePageIsSet
GetClientCodePage
GetCQDataCodePage
IsClientCodePageCompatibleWithCQDataCodePage
IsStringInCQDataCodePage
IsUnsupportedClientCodePage
CQDataCodePageIsSet of the **AdminSession Object**
Validate of the **Entity Object**

User Object

43

User (OAdUser) contains information about a user in the master database. A user's login, access privilege and some other information can be retrieved and specified through this object. You can also subscribe and unsubscribe a User to databases and upgrade user information to all the user databases that the user is currently subscribed to.

You can get the number of items in the collection by accessing the value in the **`Count`** method. 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	Read/Write	Indicates whether or not the user's account is active.
AppBuilder	Read/Write	Indicates whether or not the user has AppBuilder privileges.
Email	Read/Write	Sets or returns the user's e-mail address.
Fullname	Read/Write	Sets or returns the user's full name.
Groups	Read-only	Returns a collection of groups to which the user belongs.
MiscInfo	Read/Write	Sets or returns the user's miscellaneous information.
Name	Read-only	Returns the user's login name.
Password	Read/Write	Sets or returns the user’s password.
Phone	Read/Write	Sets or returns the user's phone number.
SubscribedDatabases	Read-only	Returns the collection of databases to which the user is subscribed.
SuperUser	Read/Write	Indicates whether or not the user has SuperUser privileges.
UserMaintainer	Read/Write	Indicates whether or not the user has user administration privileges.

Active

Description

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.

Syntax

VBScript

user.**Active**

user.**Active** *boolean_value*

Perl

$user->**GetActive();**
$user->**SetActive(***boolean_value***);**

Identifier	Description
user	A User object.
boolean_value	True if setting the user's account to active; False if not active.
Return value	A Bool indicating whether the user's account is active.

See Also

SubscribeDatabase
UnsubscribeDatabase
SubscribedDatabases
AppBuilder
SuperUser
UserMaintainer

AppBuilder

Description

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 box.)

Syntax

VBScript

user.**AppBuilder**

user.**AppBuilder** *boolean_value*

Perl

$user->**GetAppBuilder();**
$user->**SetAppBuilder(***boolean_value***);**

Identifier	Description
user	User object.
boolean_value	True if setting the user's account to active; False if not active.
Return value	A Bool indicating whether or not the user's account has AppBuilder privileges.

See Also

Active
SuperUser
UserMaintainer

Email

Description Sets or returns the user's e-mail address.

Syntax

VBScript

user.**Email**

user.**Email** *email_address_string*

Perl

```
$user->GetEmail();
$user->SetEmail(email_address_string);
```

Identifier	Description
user	A User object.
email_address_string	A String containing the user's e-mail address.
Return value	A String containing the user's e-mail address.

See Also

See "Upgrading User Information" on page 860.

Fullname

Name

Fullname

Description Sets or returns the user's full name.

Syntax

VBScript

user.**Fullname**

user.**Fullname** *full_name_string*

Perl

```
$user->GetFullName();
$user->SetFullName(full_name_string);
```

Identifier	Description
user	A User object.
full_name_string	A String containing the user's fullname.
Return value	A String containing the user's full name, as opposed to the user's login name.

See Also

See "Upgrading User Information" on page 860.

Email

Name

Groups

Description

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.

Syntax

VBScript

user.**Groups**

Perl

*$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

Description

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.

Syntax

VBScript

user.**MiscInfo**

user.**MiscInfo** *user_info_string*

Perl

$user->**Get**MiscInfo();
$user->**Set**MiscInfo(*user_info_string*);

Identifier	Description
user	A User object.
user_info_string	A String containing the user's miscellaneous information.
Return value	A String containing miscellaneous information about the user.

See Also

Fullname
Name

Name

Description

Returns the user's login name.

Syntax

VBScript

user.**Name**

Perl

$user->**GetName();**

Identifier	Description
user	A User object.
Return value	A String containing the user's login name.

See Also

Fullname

See "Upgrading User Information" on page 860.

Password

Description

Sets or returns the user's ClearQuest user password. GetPassword returns the encrypted password. You can also use **SetLoginName** to set a user password.

Syntax

VBScript

user.**Password**

user.**Password** *passwd_string*

Perl

$user->**GetPassword();**
$user->**SetPassword(***passwd_string***);**

Identifier	Description
user	A User object.
passwd_string	A String specifying the User's new password.
Return value	A String containing the User's password.

Example

Perl

```
use CQPerlExt;
# Create a ClearQuest admin session
my $adminSession = CQAdminSession::Build();

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

# Create the user "jsmith" object
my $newUserObj = $adminSession->CreateUser( "jsmith" );
die "Unable to create the user!\n" unless $newUserObj;

# Set the new user's password to secret
$newUserObj->SetPassword("secret");

# All done.
CQAdminSession::Unbuild($adminSession);
```

See Also

See "Upgrading User Information" on page 860.
Fullname
Name
CreateUser of the **AdminSession Object**
SetLoginName

Phone

Description

Sets or returns the user's telephone number.

Syntax

VBScript

user.**Phone**

user.**Phone** *phone_number_string*

Perl

```
$user->GetPhone();
$user->SetPhone(phone_number_string);
```

Identifier	Description
user	A User object.
phone_number_string	A String containing the User's telephone number.
Return value	A String containing the user's telephone number.

See Also

See "Upgrading User Information" on page 860.

Fullname

Name

SubscribedDatabases

Description

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.

Syntax

VBScript

user.**SubscribedDatabases**

Perl

$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
UnsubscribeAllDatabases
UnsubscribeDatabase
Databases Object
See "Upgrading User Information" on page 860.

SuperUser

Description

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.

Syntax

VBScript

user.**SuperUser**

user.**SuperUser** *boolean_value*

Perl

*$user->***GetSuperUser();**
*$user->***SetSuperUser(***boolean_value***);**

Identifier	Description
user	A User object.
boolean_value	Set to True if authorizing SuperUser privileges to the user's account; False if not allowing SuperUser privileges.
Return value	A Bool indicating whether or not the user's account has SuperUser privileges.

See Also

Active
AppBuilder
UserMaintainer

UserMaintainer

Description

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.

Syntax

VBScript

user.**UserMaintainer**

user.**UserMaintainer** *boolean_value*

Perl

*$user->***GetUserMaintainer();**
*$user->***SetUserMaintainer(***boolean_value***);**

Identifier	Description
user	A User object.
boolean_value	Set to True if authorizing administration privileges to the user's account; False if not allowing administration privileges.
Return value	A Bool indicating whether or not the user's account has user administration privileges.

See Also

Active
AppBuilder
SuperUser

User Object Methods

The following list summarizes the User object methods:

Note: For all Perl Get and Set methods that map to Visual Basic Properties, see the Properties section of this object.

Method Name	Description
SetLoginName	Change a login name and/or password.
SiteHasMastership	Tests whether this object is mastered in the session database.
SubscribeDatabase	Subscribes this user to the specified database.
UnsubscribeAllDatabases	Unsubscribes the user from all databases.
UnsubscribeDatabase	Unsubscribes the user from the specified database.
UpgradeInfo	Upgrades the user profile related information to all the user databases that the user is currently subscribed to.

Additional Perl Get and Set Methods that map to Visual Basic properties:

Method Name	Description
GetActive	Indicates whether or not the user's account is active.
GetAppBuilder	Indicates whether or not the user has AppBuilder privileges.
GetEmail	Returns the user's e-mail address.
GetFullName (See Fullname)	Returns the user's full name.
GetGroups	Returns a collection of groups to which the user belongs.
GetMiscInfo	Returns the user's miscellaneous information.
GetName	Returns the user's login name.
GetPassword	Returns the user's password.
GetPhone	Returns the user's phone number.
GetSubscribedDatabases	Returns the collection of databases to which the user is subscribed.
GetSuperUser	Indicates whether or not the user has SuperUser privileges.
GetUserMaintainer	Indicates whether or not the user has user administration privileges.
SetActive	Specifies whether or not the user's account is active.

Method Name	Description
SetAppBuilder	Specifies whether or not the user has AppBuilder privileges.
SetEmail	Sets the user's e-mail address.
SetFullName. (**See Fullname**)	Sets the user's full name.
SetMiscInfo	Sets the user's miscellaneous information.
SetPassword	Sets the user's password.
SetPhone	Sets the user's phone number.
SetSuperUser	Specifies whether or not the user has SuperUser privileges.
SetUserMaintainer	Specifies whether or not the user has user administration privileges.

SetLoginName

Description

Changes the login name and/or password of the current user.

This method can be used to support MultiSite operations, as it can be used to resolve ambiguous names.

To detect whether there are multiple users with the same name on other sites, you can use the **`GetDisplayNamesNeedingSiteExtension`** method in Session. For example, a user named "Tom" might have been created on more than one site.

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

There is no return value. Changes will take effect at the next login.

Syntax

VBScript

user.**SetLoginName** *new_name*, *new_password*

Perl

user->**SetLoginName**(*new_name*, *new_password*);

Identifier	Description
user	A User object.
new_name	A String containing a new or existing user name.
new_password	A String containing a new password.
Return value	None.

Example

Perl

```
# change a user password using SetLoginName
use CQPerlExt;
my $adminSession = CQAdminSession::Build();
   ($user, $newpasswd, $cqdb) = @ARGV;
   $adminUser = "admin";
   $adminPasswd = "";
   $adminSession->Logon($adminUser, $adminPasswd, "");
   $userobj = $adminSession->GetUser($user);
   $userobj->SetLoginName($user, $newpasswd);
   $dbobj= $adminSession->GetDatabase($cqdb);
   $dbobj->UpgradeMasterUserInfo();
CQAdminSession::Unbuild($adminSession);
```

See Also

`GetDisplayNamesNeedingSiteExtension`

SiteHasMastership

Description

Tests whether this User object is mastered in the local, session database and returns True if it is mastered in the local site and otherwise returns False.

This method supports MultiSite operations. An object can be modified or deleted only in its master database. An object's initial master database is the database in which it is first created, but the master database can be changed by using the MultiUtil tool.

Syntax

VBScript

user.**SiteHasMastership**

Perl

$*user*->**SiteHasMastership()**;

Identifier	Description
user	A User object.
Return value	A Boolean True or False.

See Also

SiteHasMastership in Entity
SiteHasMastership in Group
SiteHasMastership in Workspace

SubscribeDatabase

Description

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.

When calling SubscribeDatabase :

```
$UserObject->SubscribeDatabase($DBObject);
```

the user is subsequently required to execute UpgradeMasterUserInfo in order for the SubscribeDatabase method to have an effect on the user database represented by $DBObject:

```
$DBObject->UpgradeMasterUserInfo();
```

Syntax

VBScript

user.**SubscribeDatabase** *database*

Perl

*$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
UnsubscribeDatabase
SubscribedDatabases

UnsubscribeAllDatabases

Description

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.

Syntax

VBScript

user.**UnsubscribeAllDatabases**

Perl

$user->**UnsubscribeAllDatabases();**

Identifier	Description
user	A User object.
Return value	None.

See Also

SubscribeDatabase
UnsubscribeDatabase
Active
SubscribedDatabases

UnsubscribeDatabase

Description

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.

Syntax

VBScript

user.**UnsubscribeDatabase** *database*

Perl

$*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
UnsubscribeAllDatabases
SubscribedDatabases

UpgradeInfo

Description

Upgrades the user profile related information to all the user databases that the user is currently subscribed to. Returns the database names that have failed in upgrading.

This method does not upgrade the schema repository, nor does it upgrade the user's links.

Syntax

VBScript

user.**UpgradeInfo**

Perl

$*user*->**UpgradeInfo()**;

Identifier	Description
user	A User object.
Return value	For VBScript, returns a VARIANT array containing the database names that have failed in upgrading. For Perl, returns a reference to an array of strings containing the database names that have failed in upgrading.

See Also

See "Upgrading User Information" on page 860.

`SubscribeDatabase`

`UnsubscribeAllDatabases`

Users Object

44

A Users object is a collection object for User objects.

For example a Database object's `SubscribedUser` property may return a Users object.

This object can only be instantiated when you are in an AdminSession.

See Also **User Object**

Users Object Properties

The following list summarizes the Users object properties:

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

Count

Description

Returns the number of items in the collection.

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

Syntax

VBScript

collection.**Count**

Perl

$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`

"Adding and Removing Users in a Group" on page 857.

Users Object Methods

The following list summarizes the Users object methods:

Method Name	Description
`Item`	Returns the item at the specified index in the collection.
`ItemByName`	(Perl only) Returns the specified item in the collection.

Note: For Perl methods that map to Visual Basic Properties, see the Properties section of this object.

The following list summarizes additional Perl Users object methods:

Method Name	Access	Description
`Count`	Read-only	Returns the number of items in the collection.

Item

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

VBScript

collection.**Item(*itemNum*)**
collection.**Item(*name*)**

Perl

***$collection*->Item(*itemNum*);**
***$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`

"Adding and Removing Users in a Group" on page 857.

Workspace Object

45

The Rational ClearQuest workspace consists of a folder hierarchy where queries, charts and reports are stored. The Workspace (WorkSpaceMgr for Perl) 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.

For each folder type, it is designated as either a public or user (personal) folder, which is enumerated under the WorkspaceFolderType. The two top folders for a workspace are always a public and a personal folder, which are created automatically when a ClearQuest user database is created.

You can also use this object to save queries back to the workspace.

Each workspace item has a dbid assigned to it and its type is enumerated under WorkspaceItemType in clearquest.bas.

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`.

You can create nested folders explicitly using **`CreateWorkspaceFolder`** or 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 in a 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` of the Session Object
ChartMgr Object
ReportMgr Object
`QueryDef Object`

Workspace Object Methods

The following list summarizes the Workspace object methods:

Method Name	Description
CreateWorkspaceFolder	Creates a new workspace folder and returns the new dbid.
DeleteWorkspaceItemByDbId	Deletes the workspace item designated by dbid.
GetAllQueriesList	Returns the complete list of queries in the workspace.
GetChartDbIdList	Returns the list of dbids parallel to the list of names returned from GetChartList.
GetChartDef	Returns the QueryDef object associated with the specified chart.
GetChartDefByDbId	Returns the QueryDef object associated with the specified dbid. Same as GetChartDef, except the lookup is by dbid.
GetChartList	Returns the specified list of charts.
GetChartMgr	Returns the CHARTMGR object associated with the current session.
GetPersonalFolderName	Returns name of the personal queries folder from the resource file.
GetPublicFolderName	Returns name of the public queries folder from the resource file.
GetQueryDbIdList	Returns the list of dbids parallel to the names returned from GetQueryList.
GetQueryDef	Returns the QueryDef object associated with the specified workspace query.
GetQueryDefByDbId	Returns the QueryDef object associated with the specified workspace query. Same as GetQueryDef, except the lookup is by dbid.
GetQueryList	Returns the specified list of workspace queries.
GetReportDbIdList	Returns the list of dbids parallel to the list returned from GetReportList.
GetReportList	Returns the specified list of reports.
GetReportMgr	Returns the ReportMgr object associated with the current session.
GetReportMgrByReportDbId	Returns the ReportMgr object associated with the current session. Same as GetReportMgr, except the report is designated by dbid.
GetSiteExtendedNames	Gets extended names of workspace items.

Method Name	Description
GetWorkspaceItemDbIdList	Returns a list of dbids of workspace items based on the input criteria.
GetWorkspaceItemName	Returns name of a workspace item.
GetWorkspaceItemParentDbId	Returns the parent dbid of the given workspace item.
GetWorkspaceItemPathName	Returns a list of path names for the workspace item, including the name of the workspace item itself.
GetWorkspaceItemSiteExtendedName	Returns site extended name of a workspace item, whether it needs it or not.
GetWorkspaceItemType	Returns workspace item type, as enumerated in WorkspaceItemType.
InsertNewChartDef	Inserts a new chart into the workspace, under the workspace folder specified by parent dbid.
InsertNewQueryDef	Inserts a new query into the workspace, under the workspace folder specified by parent dbid.
RenameWorkspaceItem	Rename a workspace item.
RenameWorkspaceItemByDbId	Rename a workspace item. Same as RenameWorkspaceItem except lookup is by workspace item dbid rather than name.
SaveQueryDef	Saves the query to the specified location in the workspace.
SetSession	Associates the specified Session object with this object.
SetUserName	Sets the current user name when searching for queries, charts, or reports.
SiteExtendedNameRequired	Returns whether a site extended name is required for the given workspace item.
SiteHasMastership	Tests whether this object is mastered in the session database.
UpdateChartDef	Overwrites an existing chart workspace item specified by dbid, with the QueryDef object.
UpdateQueryDef	Overwrites an existing query workspace item specified by dbid, with the given QueryDef object.
ValidateQueryDefName	Verifies that the specified query name and path info are correct.

CreateWorkspaceFolder

Description Creates a new workspace folder and returns the new dbid.

Note: This method is for Windows only.

Syntax

VBScript

*workspace.*__CreateWorkspaceFolder__ *user_id, folder_type, new_name, parent_dbid*

Perl

*$workspace->*__CreateWorkspaceFolder__*(user_id, folder_type, new_name, parent_dbid);*

Identifier	Description
workspace	The Workspace object obtained from the current session.
user_id	A Long. Set this to 0.
folder_type	A Long containing the folder type as enumerated by WorkspaceFolderType. The workspace folder types are: Public folder items (_WORKSPACE_PUBLIC_FOLDER = 1) Personal folder items (_WORKSPACE_USER_FOLDER = 2)
new_name	A String containing the name of new folder.
parent_dbid	A Long containing the parent folder dbid from which to create the new workspace folder (should never be 0).
Return value	Returns a Long containing the new dbid.

See Also **GetWorkspaceItemParentDbId**

DeleteWorkspaceItemByDbId

Description Deletes the workspace item designated by dbid.

Syntax

VBScript

*workspace.*__DeleteWorkspaceItemByDbId__ *dbid*

Perl

*$workspace->*__DeleteWorkspaceItemByDbId__*(dbid);*

Identifier	Description
workspace	The Workspace object obtained from the current session.
dbid	A Long containing the dbid of the workspace item to delete.
Return value	Returns Boolean True is if successful; False if failed.

See Also `GetWorkspaceItemDbIdList`

GetAllQueriesList

Description

Returns the complete list of queries (queries, charts, reports) in the workspace.

This method returns both the public queries defined by the ClearQuest administrator and personal queries created by individual users.

Syntax

VBScript

variant_queries = *workspace*.**GetAllQueriesList**

Perl

$*ref_queries* = $*workspace*->**GetAllQueriesList()**;

Identifier	Description
workspace	The Workspace object obtained from the current session.
Return value	In Visual Basic, an array of Variants (each containing a string) that make up the list of all queries (queries, charts, reports). Each string contains the pathname of a query. In Perl, a reference to an array of strings containing the query pathnames.

Example

Perl

```
$MyWorkSpace = $Session->GetWorkSpace();
$MyQueriesListREF = $MyWorkSpace->GetAllQueriesList();
foreach (@$MyQueriesListREF) {
    print ("Query name: $_\n");
}
```

See Also

GetQueryDef
GetQueryList
QueryDef Object

GetChartDbIdList

Description

Returns the list of dbids parallel to the list of names returned from GetChartList, when specifying the same *charttype* argument.

Note: This method is for Windows only.

Syntax

VBScript

*workspace.*__GetChartDbIdList__ *charttype*

Perl

*$workspace->*__GetChartDbIdList__*(charttype);*

Identifier	Description
workspace	The Workspace object obtained from the current session.
charttype	The type of chart dbids to return. 1 - public charts 2 - personal charts 3 - all public and personal charts
Return value	In Visual Basic, an array of Variants (each containing a string) that make up the list of dbids parallel to the list returned from GetChartList. In Perl, a reference to an array of strings containing the list of chart dbids.

See Also

GetChartDef
GetChartList
GetChartMgr
ChartMgr Object

GetChartDef

Description

Returns the QueryDef object associated with the specified chart.

Note: This method is for Windows only.

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.

Syntax

VBScript

workspace.**GetChartDef** *chartName*

Perl

$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	Returns a reference to the QueryDef object associated with the chart.

See Also

GetChartMgr
GetChartList
ChartMgr Object
QueryDef Object

GetChartDefByDbId

Description Returns the QueryDef object associated with the specified dbid. GetChartDefByDbId is the same as GetChartDef, except the lookup is by dbid.

Note: This method is for Windows only.

Syntax

VBScript

*workspace.***GetChartDefByDbId** *dbid*

Perl

*$workspace->***GetChartDefByDbId** *(dbid);*

Identifier	Description
workspace	The Workspace object obtained from the current session.
dbid	A Long containing the dbid of the chart.
Return value	A reference to a QueryDef object.

See Also

GetChartDef
GetChartList
GetChartMgr
ChartMgr Object

GetChartList

Description

Returns the specified list of charts.

Note: This method is for Windows only.

You must first call **SetSession** on the Workspace object, if you have not created a Session object.

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 (1 for Perl) returns only the public charts defined by the ClearQuest administrator. Specifying the constant OLEWKSPCBOTHQUERIES (3 for Perl) 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 (2 for Perl) for the *typeOfCharts* parameter.

Syntax

VBScript

workspace.**GetChartList** *typeOfCharts*

Perl

$workspace->**GetChartList**(*typeOfCharts*);

Identifier	Description
workspace	The Workspace object obtained from the current session.
typeOfCharts	The type of charts to return. 1 - public charts 2 - personal charts 3 - all public and personal charts For Visual Basic, this value corresponds to one of the *Workspace Query Type Constants* enumerated constants.
Return value	In Visual Basic, an array of Variants (each containing a string) that make up the list of chart definition names known to the database. Each String contains the pathname of a single chart. For Perl, a reference to an array of strings, each of which contains the pathname of a single chart.

See Also

GetChartDef
GetChartMgr
SetUserName
ChartMgr Object

GetChartMgr

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.

Note: This method is for Windows only.

Syntax

VBScript

workspace.**GetChartMgr**

Perl

$workspace->**GetChartMgr**();

Identifier	Description
workspace	The Workspace object obtained from the current session.
Return value	The CHARTMGR object associated with the current session.

Examples

VBScript

```
Set oSession = CreateObject("CLEARQUEST.SESSION")
oSession.UserLogon "admin", "", "RUC", AD_PRIVATE_SESSION, ""

Set oWorkSpace = oSession.GetWorkSpace
querylist = oWorkSpace.GetChartList(OLEWKSPCSYSTEMQUERIES)
For Each querystr In querylist
  Set cq_query_def = oWorkspace.GetChartDef(querystr)
  Set cq_resultset = oSession.BuildResultSet(cq_query_def)
  filename = "c:\test.jpg"
  Call cq_resultset.Execute
  Set oChartMgr = oWorkSpace.GetChartMgr
  Call oChartMgr.SetResultSet(cq_resultset)
  oChartMgr.Width = 600
  oChartMgr.Height = 600
  oChartMgr.MakeJPEG(filename)

Next
```

Perl

```
use CQPerlExt;

$session = CQSession::Build();
$user = "admin";
```

```
$pass = "";
$db = "SAMPL";
$session->UserLogon($user, $pass, $db, "");

$wkSpc = $session->GetWorkSpace();
$chartDef = $wkSpc->GetChartDef("Personal Queries/Sample_Chart");
$resultSet = $session->BuildResultSet($chartDef);
$resultSet->SetMaxRowsInMemory(2000);
$resultSet->Execute();

$chartMgr = $wkSpc->GetChartMgr();
$chartMgr->SetResultSet($resultSet);
$chartMgr->MakeJPEG("C:\\temp\\BBChart.jpg");

CQSession::Unbuild($session);
```

See Also

GetChartDef
GetChartList
ChartMgr Object

GetPersonalFolderName

Description Returns name of the personal queries folder from the resource file.

Syntax

VBScript

workspace.**GetPersonalFolderName**

Perl

*$workspace->***GetPersonalFolderName***();*

Identifier	Description
workspace	The Workspace object obtained from the current session.
Return value	The string containing the name of the folder.

Examples

VBScript

```
folderName = workspace.GetPersonalFolderName
```

Perl

```
$folderName = $workspace->GetPersonalFolderName();
```

See Also **GetPublicFolderName**

GetPublicFolderName

Description Returns name of the public queries folder from the resource file.

Syntax

VBScript

workspace.**GetPublicFolderName**

Perl

*$workspace->***GetPublicFolderName***();*

Identifier	Description
workspace	The Workspace object obtained from the current session.
Return value	The string containing the name of the folder.

Examples

VBScript

```
folderName = workspace.GetPublicFolderName
```

Perl

```
$folderName = $workspace->GetPublicFolderName();
```

See Also **GetPersonalFolderName**

GetQueryDbIdList

Description Returns the list of dbids parallel to the names returned from GetQueryList, with the same querytype argument value.

Note: This method is for Windows only.

Syntax

VBScript

workspace.`GetQueryDbIdList` *querytype*

Perl

$workspace->`GetQueryDbIdList`*(querytype);*

Identifier	Description
workspace	The Workspace object obtained from the current session.
querytype	The type of queries to return. 1 - public queries 2 - personal queries 3 - all public and personal queries For Visual Basic, this value corresponds to one of the *Workspace Query Type Constants* enumerated constants.
Return value	For Visual Basic, an array of Variants (each containing a string) that make up the list of dbids parallel to the names returned from GetQueryList. Each String contains one dbid. For Perl, returns a reference to an array of strings.

See Also `GetQueryList`
QueryDef Object

GetQueryDef

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.

Note: You must call SetSession on the Workspace object before calling GetQueryDef, if you have not created a Session object.

Syntax

VBScript

workspace.**GetQueryDef** *queryName*

Perl

$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	Returns a reference to the QueryDef object associated with the query.

See Also

GetQueryList
QueryDef Object

GetQueryDefByDbId

Description Returns the QueryDef object associated with the specified workspace query. This method is the same as GetQueryDef, except the lookup is by dbid.

Note: You must call SetSession on the Workspace object before calling GetQueryDef, if you have not created a Session object.

Syntax

VBScript

workspace.`GetQueryDefByDbId` *dbid*

Perl

$workspace->`GetQueryDefByDbId` *(dbid);*

Identifier	Description
workspace	The Workspace object obtained from the current session.
dbid	A Long containing the dbid of the query.
Return value	Returns a reference to the QueryDef object associated with the query.

See Also `GetQueryDef`

GetQueryList

Description

Returns the specified list of workspace queries known to the database.

This method returns the pathnames of the public or personal queries defined in the ClearQuest workspace. The *querytype* parameter lets you specify the type of queries to return. Specifying the constant `OLEWKSPCSYSTEMQUERIES (1)` returns only the public queries defined by the ClearQuest administrator. Specifying the constant `OLEWKSPCBOTHQUERIES (3)` 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` (2) for the *querytype* parameter.

Note: You must first call the **`SetSession`** method of the **`Workspace Object`**, if you have not created a Session object.

Syntax

VBScript

`workspace``.`**GetQueryList** *`querytype`*

Perl

`$workspace->``GetQueryList(`*`querytype`*`);`

Identifier	Description
workspace	The Workspace object obtained from the current session.
querytype	The type of queries to return. 1 - public queries 2 - personal queries 3 - all public and personal queries For Visual Basic, this value corresponds to one of the *Workspace Query Type Constants* enumerated constants.
Return value	For Visual Basic, an array of Variants (each containing a string) that make up the list of query definition names known to the database. Each item contains the pathname of a single query. For Perl, a reference to an array of strings, each of which contains the pathname of a single query.

See Also

`GetQueryDef`
`SetUserName`
QueryDef Object

GetReportDbIdList

Description Returns the list of dbids parallel to the list returned from GetReportList.

Note: This method is for Windows only.

Syntax

VBScript

workspace.**GetReportDbIdList** reporttype

Perl

$workspace->**GetReportDbIdList** (reporttype);

Identifier	Description
workspace	The Workspace object obtained from the current session.
reporttype	The type of reports to return. 1 - public reports 2 - personal reports 3 - all public and personal reports
Return value	For Visual Basic, returns an array of Variants (each containing a string) that make up the list of dbids parallel to the list returned from GetReportList. Each string names one dbid. For Perl, returns a reference to an array of strings. Each string names one dbid.

See Also `GetReportList`

GetReportList

Description

Returns the specified list of reports.

Note: This method is for Windows only.

This method returns the pathnames of the public or personal reports defined in the ClearQuest workspace. The *reporttype* parameter lets you specify the type of reports to return. Specifying the constant `OLEWKSPCSYSTEMREPORTS (1)` returns only the public reports defined by the ClearQuest administrator. Specifying the constant `OLEWKSPCBOTHREPORTS (3)` 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 (2)` for the *reporttype* parameter.

Syntax

VBScript

workspace.**GetReportList** *reporttype*

Perl

$workspace->**GetReportList**(*reporttype*);

Identifier	Description
workspace	The Workspace object obtained from the current session.
reporttype	The type of reports to return. 1 - public reports 2 - personal reports 3 - all public and personal reports For Visual Basic, this value corresponds to one of the *Workspace Query Type Constants* enumerated constants.
Return value	For Visual Basic, returns an array of Variants (each containing a string) that make up the list of report definition names known to the database. Each string contains the pathname of a single report. For Perl, a reference to an array of strings. Each string contains the pathname of a single report.

See Also

`GetReportMgr`
ReportMgr Object

GetReportMgr

Description

Returns the ReportMgr object associated with the current session. Report to be generated is designated by the reportName parameter.

Note: This method is for Windows only.

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.

Syntax

VBScript

workspace.**GetReportMgr** *reportName*

Perl

$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	Returns a reference to a ReportMgr object.

Example

VBScript

```
Set oSession = CreateObject("CLEARQUEST.SESSION")
oSession.UserLogon "admin", "", "RUC", AD_PRIVATE_SESSION, ""

Set oWorkSpace = oSession.GetWorkSpace
querylist = oWorkSpace.GetReportList(OLEWKSPCSYSTEMQUERIES)
For Each querystr In querylist
  filename = "c:\test.html"
  Set oReportMgr = oWorkSpace.GetReportMgr(querystr)
  oReportMgr.SetHTMLFileName filename
  Call oReportMgr.ExecuteReport
Next
```

Perl

```
use CQPerlExt;
my $session;
  my $workspace;
  my $reportMgr;
  my $reportName = "Personal Queries/Sample_report";
  my $htmlPath = "c:\\temp\\my-report.html";
```

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

CQSession::UserLogon ("admin", "", "SAMPL", "");

$workspace = $session->GetWorkSpace();
$reportMgr = $workspace->GetReportMgr ( $reportName );
$reportMgr->SetHTMLFileName($htmlPath);
$reportMgr->ExecuteReport();

CQSession::Unbuild($session);
```

See Also

ReportMgr Object

GetReportList

SetHTMLFileName of the **ReportMgr Object**

ExecuteReport of the **ReportMgr Object**

GetReportMgrByReportDbId

Description

Returns the ReportMgr object associated with the current session. This method is the same as GetReportMgr, except the report is designated by dbid, rather than report name. Report to be generated is designated by the report_dbid parameter.

Note: This method is for Windows only.

Syntax

VBScript

workspace.**GetReportMgrByReportDbId** report_dbid

Perl

*$workspace->***GetReportMgrByReportDbId** *(*report_dbid*);*

Identifier	Description
workspace	The Workspace object obtained from the current session.
report_dbid	A String containing the dbid of the report to run with the returned ReportMgr object.
Return value	Returns a reference to a ReportMgr object.

See Also

`GetReportMgr`

GetSiteExtendedNames

Description

Gets site-extended names for a given workspace item.

This method supports MultiSite operations and may be useful in detected or resolving name conflicts. For example, a replica site might have charts, queries, reports, and report formats with the same pathnames as the local site. This means that there is a potential for duplicate names. This method allows you to get site names that uniquely identify a workspace item. An extended name can be used in APIs wherever unextended names are used.

If the specified item is already an extended name, no error will occur and extended names will still be returned. If the specified name is invalid, the returned array will be empty. You must have access to the specified locations. If you do not have access or if the specified location does not exist, an empty array will be returned. Folder names must be separated with a forward slash (/) character.

Syntax

VBScript

```
var_site_names = workspace.GetSiteExtendedNames item_path
```

Perl

```
$ref_site_names = $workspace->GetSiteExtendedNames(item_path);
```

Identifier	Description
workspace	The Workspace object obtained from the current session.
item_path	The path to a workspace item.
Return value	In Visual Basic, the return value is a Variant containing extended names. In Perl, the return value is a reference to an array of strings.

See Also

RenameWorkspaceItem
GetDisplayNamesNeedingSiteExtension
GetSiteExtendedNames
GetSiteExtension
GetUnextendedName
IsSiteExtendedName
ParseSiteExtendedName

GetWorkspaceItemDbIdList

Description Returns a list of dbids of workspace items based on the input criteria.

Syntax

VBScript

*workspace.*__GetWorkspaceItemDbIdList__ *folder_type, item_type, parent_dbid, entdef_name*

Perl

*$workspace->*__GetWorkspaceItemDbIdList__*(folder_type, item_type, parent_dbid, entdef_name);*

Identifier	Description
workspace	The Workspace object obtained from the current session.
folder_type	A Long containing the folder type as enumerated by WorkspaceFolderType. The workspace folder types are: Public folder items (_WORKSPACE_PUBLIC_FOLDER = 1) Personal folder items (_WORKSPACE_USER_FOLDER = 2)
item_type	A Long containing a WorkspaceItemType enumerated constant.
parent_dbid	A Long corresponding to the dbid of the parent folder. Set this to 0 for retrieving top folders.
entdef_name	A String containing the EntityDef name associated with the workspace item. This argument can be empty.
Return value	For Visual Basic, returns an array of Variants (each containing a String) that make up the list of dbids of workspace items. Each String names one dbid. For Perl, returns a reference to an array of strings containing the dbid list.

Examples

VBScript

```
// Retrieves dbid for top public folder (returns a list of 1 item)
GetWorkspaceItemDbIdList(AD_WORKSPACE_PUBLIC_FOLDER, AD_WORKSPACE_FOLDER, 0, "")
// Retrieves dbid for children folders of public query folder
GetWorkspaceItemDbIdList(AD_WORKSPACE_PUBLIC_FOLDER, AD_WORKSPACE_FOLDER, 33554440, "")
```

See Also **WorkspaceItemType Constants**

GetName of the **EntityDef Object**

GetWorkspaceItemName

Description Returns name of a workspace item.

Syntax

VBScript

workspace.`GetWorkspaceItemName` *dbid, extend_option*

Perl

$workspace->`GetWorkspaceItemName`*(dbid, extend_option);*

Identifier	Description
workspace	The Workspace object obtained from the current session.
dbid	A Long containing the dbid of the workspace item.
extend_option	A Long containing the one of the following WorkspaceNameOption enumerated constants: _WORKSPACE_NAME_NOT_EXTENDED = 1 _WORKSPACE_NAME_EXTENDED = 2 _WORKSPACE_NAME_EXTEND_WHEN_NEEDED = 3
Return value	Returns a string containing the name of the workspace item.

See Also `RenameWorkspaceItem`

GetWorkspaceItemParentDbId

Description Returns the parent dbid of the given workspace item.

Note: This method is for Windows only.

Syntax

VBScript

```
workspace.GetWorkspaceItemParentDbId dbid
```

Perl

```
$workspace->GetWorkspaceItemParentDbId(dbid);
```

Identifier	Description
workspace	The Workspace object obtained from the current session.
dbid	A String containing the dbid of the item for which you want the parent dbid.
Return value	A Long containing the parent dbid.

See Also

GetWorkspaceItemDbIdList

InsertNewChartDef

InsertNewQueryDef

GetWorkspaceItemPathName

Description Returns a list of path names for the workspace item, including the name of the workspace item itself. Each string contains a path name for the workspace item.

Syntax

VBScript

workspace.`GetWorkspaceItemPathName` *dbid, extend_option*

Perl

$workspace->`GetWorkspaceItemPathName`*(dbid, extend_option);*

Identifier	Description
workspace	The Workspace object obtained from the current session.
dbid	A Long containing the dbid of the workspace item.
extend_option	A Long containing the one of the following WorkspaceNameOption enumerated constants: _WORKSPACE_NAME_NOT_EXTENDED = 1 _WORKSPACE_NAME_EXTENDED = 2 _WORKSPACE_NAME_EXTEND_WHEN_NEEDED = 3
Return value	For Visual Basic, returns an array of strings. For Perl, returns a reference to an array of strings. Each string contains a path name for the workspace item, including the name of the workspace item itself.

See Also `GetWorkspaceItemDbIdList`

GetWorkspaceItemSiteExtendedName

Description Returns site extended name of a workspace item, whether it needs it or not.

Syntax

VBScript

workspace.`GetWorkspaceItemSiteExtendedName` *dbid*

Perl

$workspace->`GetWorkspaceItemSiteExtendedName` *(dbid);*

Identifier	Description
workspace	The Workspace object obtained from the current session.
dbid	A Long containing the dbid of the workspace item.
Return value	Returns a String containing the site extended name of the workspace item.

See Also `GetSiteExtendedNames`

`SiteExtendedNameRequired`

GetWorkspaceItemType

Description Returns workspace item type, as enumerated in WorkspaceItemType.

Syntax

VBScript

workspace.**GetWorkspaceItemType** *dbid*

Perl

*$workspace->***GetWorkspaceItemType** *(dbid);*

Identifier	Description
workspace	The Workspace object obtained from the current session.
dbid	A Long containing the dbid of the workspace item.
Return value	Returns a Long containing an enumerated WorkspaceItemType.

See Also **WorkspaceItemType Constants**

InsertNewChartDef

Description Inserts a new chart into the workspace, under the workspace folder specified by parent dbid.

Note: This method is for Windows only.

Syntax

VBScript

workspace.`InsertNewChartDef` *newName, parent_dbid, QueryDefObj*

Perl

$workspace->`InsertNewChartDef` *(newName, parent_dbid, QueryDefObj);*

Identifier	Description
workspace	The Workspace object obtained from the current session.
newName	A String containing the name you want to give to the new ChartDef.
parent_dbid	A Long containing the dbid of the parent workspace folder from which to insert the new chart.
QueryDefObj	A reference to the new QueryDef object you are inserting.
Return value	Returns a Long containing the dbid of the new chart.

See Also

`QueryDef Object`

`GetChartDef`

`GetWorkspaceItemParentDbId`

`GetChartDefByDbId`

InsertNewQueryDef

Description Inserts a new query into the workspace, under the workspace folder specified by parent dbid.

Syntax

VBScript

workspace.**InsertNewQueryDef** *newName, parent_dbid, QueryDefObj*

Perl

*$workspace->***InsertNewQueryDef***(newName, parent_dbid, QueryDefObj);*

Identifier	Description
workspace	The Workspace object obtained from the current session.
newName	A String containing the name you want to give to the new query.
parent_dbid	A Long containing the dbid of the parent workspace folder from which to insert the new query.
QueryDefObj	A reference to the new QueryDef object.
Return value	Returns the dbid of the new query.

See Also

QueryDef Object

GetQueryDef

GetWorkspaceItemParentDbId

RenameWorkspaceItem

Description

Renames a workspace item and returns Boolean True if successful and otherwise returns False.

This method supports MultiSite operations and may be useful in resolving naming conflicts. You can use this method to change an ambiguous, workspace item name to an unambiguous name. You must have access to the specified locations. If you do not have access or if the specified location does not exist, an empty array will be returned. Folder names must be separated with a forward slash (/) character.

For example, a replica site might have charts, queries, reports, and report formats with the same pathnames as the local site. This means that there is a potential for duplicate names. This method allows you to change site names so that they uniquely identify a workspace item.

Syntax

VBScript

```
success = workspace.RenameWorkspaceItem old_path, new_name
```

Perl

```
$success = $workspace->RenameWorkspaceItem(old_path, new_name);
```

Identifier	Description
workspace	The Workspace object obtained from the current session.
old_path	The existing path to a workspace item.
new_name	A String containing the new name of the workspace item.
Return value	Returns Boolean True if successful; False if failed.

See Also

GetSiteExtendedNames

The following related methods are in Session:

GetDisplayNamesNeedingSiteExtension
GetSiteExtendedNames
GetSiteExtension
GetUnextendedName
IsSiteExtendedName
ParseSiteExtendedName

RenameWorkspaceItemByDbId

Description Renames a workspace item with a new name and returns Boolean True if successful and otherwise returns False. This method is the same as RenameWorkspaceItem except lookup is by workspace item dbid rather than name.

Syntax

VBScript

workspace.**RenameWorkspaceItemByDbId** *dbid, newName*

Perl

*$workspace->***RenameWorkspaceItemByDbId** *(dbid, newName);*

Identifier	Description
workspace	The Workspace object obtained from the current session.
dbid	A Long containing the existing dbid to a workspace item.
newName	A String containing the new name of the workspace item.
Return value	Returns Boolean True if successful; False if failed.

See Also `RenameWorkspaceItem`

SaveQueryDef

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.

Syntax

VBScript

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

Perl

*$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.

Example

Perl

```
use CQPerlExt;
my $CQSession = CQSession::Build();
$CQSession->UserLogon($ologon, $opw, $odb, "");
$workspace = $CQSession->GetWorkSpace();
$QueryDef = $CQSession->BuildQuery("Defect");
@owner = ("jswift");
@state = ("Closed");
@dbfields = ("ID","State","Headline");
foreach $field (@dbfields) {
      $QueryDef->BuildField($field);
      }
$FilterNode1 = $QueryDef->BuildFilterOperator($CQPerlExt::CQ_BOOL_OP_AND);
$FilterNode1->BuildFilter("Owner", $CQPerlExt::CQ_COMP_OP_EQ, \@owner);
$FilterNode1->BuildFilter('State', $CQPerlExt::CQ_COMP_OP_NOT_IN, \@state);
$ResultSet = $CQSession->BuildResultSet($QueryDef);
$ResultSet->Execute();
$workspace->SaveQueryDef("delete me", $RootFolder, $QueryDef, 1);
```

```
print "'$RootFolder/delete me' copied\n";
  }
CQSession::Unbuild($CQSession);
```

See Also

GetQueryDef
GetQueryList
QueryDef Object

SetSession

Description

Associates the specified Session object with the Workspace 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.

Syntax

VBScript

workspace.**SetSession** *sessionObj*

Perl

$workspace->**SetSession**(*sessionObj*);

Identifier	Description
workspace	The Workspace object obtained from the current session.
sessionObj	A reference to the Session object created previously (as in, set sessionObj = CreateObject("CLEARQUEST.SESSION") for Visual Basic).
Return value	None. For Visual Basic, if the method fails it sets the error object's error code to OLEWKSPC_E_CANTCREATESESSION

See Also

Session Object

SetUserName

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.

Syntax

VBScript

workspace.**SetUserName** *userName*

Perl

$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
GetQueryList
GetReportList

SiteExtendedNameRequired

Description Returns whether a site extended name is required for the given workspace item. This also applies to replicated databases.

Syntax

VBScript

workspace.**SiteExtendedNameRequired** *dbid*

Perl

*$workspace->***SiteExtendedNameRequired** *(dbid);*

Identifier	Description
workspace	The Workspace object obtained from the current session.
dbid	A Long containing the dbid of the workspace item.
Return value	Returns Boolean True if a site extended name is required for the given workspace item; False otherwise.

See Also **GetSiteExtendedNames**

GetWorkspaceItemSiteExtendedName

SiteHasMastership

Description

Tests whether this Workspace object is mastered in the local, session database and returns True if it is mastered in the local site and otherwise returns False.

This method supports MultiSite operations. An object can be modified or deleted only in its master database. An object's initial master database is the database in which it is first created, but the master database can be changed by using the MultiUtil tool.

Syntax

VBScript

```
is_mastered_locally = workspace.SiteHasMastership
```

Perl

```
$is_mastered_locally = $workspace->SiteHasMastership();
```

Identifier	Description
workspace	The Workspace object obtained from the current session.
Return value	The return value is Boolean True if this object is mastered in the session database, and otherwise is False.

See Also

`SiteHasMastership` in Entity
`SiteHasMastership` in Group
`SiteHasMastership` in User

UpdateChartDef

Description Overwrites an existing chart workspace item specified by dbid, with the QueryDef object.

Note: This method is for Windows only.

Syntax

VBScript

*workspace.***UpdateChartDef** *dbid, QueryDefObj*

Perl

*$workspace->***UpdateChartDef***(dbid, QueryDefObj);*

Identifier	Description
workspace	The Workspace object obtained from the current session.
dbid	A Long containing the dbid of the existing chart workspace item.
QueryDefObj	A reference to the QueryDef object you are updating the chart workspace item with.
Return value	None.

See Also

QueryDef Object

GetChartDbIdList

GetChartDefByDbId

UpdateQueryDef

Description Overwrites an existing query workspace item specified by dbid, with the given QueryDef object.

Syntax

VBScript

*workspace.*__UpdateQueryDef__ *dbid, QueryDefObj*

Perl

*$workspace->*__UpdateQueryDef__*(dbid, QueryDefObj);*

Identifier	Description
workspace	The Workspace object obtained from the current session.
dbid	The dbid of the existing workspace item you want to overwrite.
QueryDefObj	A reference to the QueryDef object you are updating the workspace item with.
Return value	None.

See Also

QueryDef Object

GetQueryDbIdList

GetQueryDefByDbId

ValidateQueryDefName

Description Verifies that the specified query name and path information are correct.

Syntax

VBScript

workspace.**ValidateQueryDefName** *qdefName, qdefPath*

Perl

*$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**

Enumerated Constants

46

This topic lists all the constants used as arguments or return values by the methods and properties in the Rational ClearQuest API, except as otherwise noted. The constants are grouped into the following categories:

- **ActionType Constants**
- **Behavior Constants**
- **BoolOp Constants**
- **ChoiceType Constants**
- **CompOp Constants**
- **CType Constants**
- **DatabaseVendor Constants**
- **DbAggregate Constants**
- **DbFunction Constants**
- **EntityStatus Constants**
- **EntityType Constants**
- **EventType Constants**
- **FetchStatus Constants**
- **FieldType Constants**
- **FieldValidationStatus Constants**
- **QueryType Constants**
- **SessionType Constants**
- **Sort Constants**
- **UserPrivilegeMaskType Constants**
- **ValueStatus Constants**
- **WorkspaceFolderType Constants**
- **WorkspaceItemType Constants**
- **Workspace Query Type Constants**
- **OLEWKSPCERROR Constants**
- **OLEWKSPCERROR Constants**

You use the following prefixes for these constants:

- For VBScript, use AD. For example: AD_ORACLE.

- For Perl, use $CQPerlExt::CQ. For example, $CQPerlExt::CQ_ORACLE.

Note: For the difference between VBScript and Perl constants, see “Notation Conventions for VBScript” on page 3 and “Notation Conventions for Perl” on page 2.

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.
_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`**, instead of having to call three: **`EditEntity`**, **`Validate`**, and **`Commit`**.

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

ChoiceType Constants

The ChoiceType constants identify the choice list type for a field.

Constant	Value	Description
_CLOSED_CHOICE	1	Must use a choice list option. This means that the valid values for the field are limited to those specified in a choice list.
_OPEN_CHOICE	2	Can enter something besides the choice list. The user may select an item from the choice list or type in a new value.

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)

CType Constants

The CType constants identify the underlying column datatypes of the fields in a schema.

These constants distinguish what kind of SQL datatype the fields came from.

Note: The CType constant uses a different notation for the standard convention of the ClearQuest API enumerations. You use the following prefixes for CType constants:

- For VBScript, use PD. For example: PD_C_CHAR.
- For Perl, use $CQPerlExt::PD. For example, $CQPerlExt::PD_C_CHAR.

Constant	Value	Description
PD_C_CHAR	1	A CHAR data type.
PD_C_LONGVARCHAR	2	A CHAR * for a null terminated string as long as 2 Gigabytes. The actual limit depends on the database limit.
PD_C_LONGVARBINARY	3	A CHAR * but not null terminated. This comes from a database type of binary large objects and contains as much as 2 Gigabytes.
PD_C_SLONG	4	A LONG integer.
PD_C_TIMESTAMP	5	A CHAR * that holds a datetime "value.as." For example, yyyy-mm-dd or hh:mm:ss.
PD_C_DOUBLE	6	A double data type.

DatabaseVendor Constants

The DatabaseVendor constants identify the supported database types.

Constant	Value	Description
_SQL_SERVER	1	A SQL Server database.
_MS_ACCESS	2	An MS Access database.
_SQL_ANYWHERE	3	A SQL Anywhere database.
_ORACLE	4	An Oracle database using Oracle client networking software.
_DB2	5	A DB2 database.

DbAggregate Constants

The DbAggregate constants identify the SQL aggregate functions for a field in a schema.

Constant	Value	Description
_DB_AGGR_COUNT	1	Returns the count of records that have some value in the field.
_DB_AGGR_SUM	2	Returns the sum of the values of a field.
_DB_AGGR_AVG	3	Returns the average value of a field.
_DB_AGGR_MIN	4	Returns the minimum value of a field.
_DB_AGGR_MAX	5	Returns the maximum value of a field.

Note: SUM & AVG are supported only for numeric fields (that is, integer or float field types). The other functions work for for any field type.

DbFunction Constants

The DbFunction constants identify a function type for a field type (QueryFieldDef). These are date functions that can only be applied to datetime fields.

Constant	Value	Description
_DB_DAY_FUNC	1	The current day iteself. For example, 8/29/2002.
_DB_WEEK_FUNC	2	The week number of the year and the year, separated by a comma. For example, 35,2002 for 8/29/2002 (the 35th week of the year).
_DB_MONTH_FUNC	3	The first of the month. For example, 8/1/2002 for the current day, 8/29/2002.
_DB_YEAR_FUNC	4	The date that is the first of the year. For example, 1/1/2002 for the current day, 8/29/2002.

EntityStatus Constants

The EntityStatus constants identify the possible exceptions generated for security conditions.

Constant	Value	Description
_ENTITY_NOT_FOUND	1	Entity does not exist in the database
_ENTITY_VISIBLE	2	Entity exists and is visible to current user
_ENTITY_HIDDEN	3	Entity exists but hidden from current user

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 EventType 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 subdialog button click.
_ITEM_SELECTION	3	The hook invocation is triggered by an item selection.
_ITEM_DBLCLICK	4	The hook invocation is triggered by a point device double-click.
_CONTEXMENU_ITEM_SELECTION	5	The hook invocation is triggered by a contextual menu selection.
_CONTEXMENU_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	Arbitrarily long text
_INT	3	Integer
_DATE_TIME	4	Timestamp information
_REFERENCE	5	A pointer to a stateless record type.
_REFERENCE_LIST	6	A list of references
_ATTACHMENT_LIST	7	A list of attached files
_ID	8	A string ID for records (Entity objects)
_STATE	9	The current state of a state-based record (that is, a request entity).
_JOURNAL	10	A list of rows in a subtable that belongs exclusively to this record (Entity)
_DBID	11	An internal numeric ID
_STATETYPE	12	State type of record (entity). State types are defined by schema packages and are assigned to states within your schema. For example, UCM uses state types to determine when hooks should run. For more information about state types, see Administering Rational ClearQuest.
_RECORDTYPE	13	The name of the record type (EntityDef) of the current record (Entity). For example, “Defect” or “Customer”.

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. You use this constant to specify session type for a user login.

Note: Perl does not recognize SessionType constants. The session_type argument is for VBScript only.

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.
_SHARED_METADATA_SESSION	4	The session owns only a read-only copy of the metadata to be shared by other shared or private sessions.

Sort Constants

The Sort constant is for specifying the sort type for the a field in a ResultSet.

Constant	Value	Description
_SORT_ASC	1	Sort in ascending order.
_SORT_DESC	2	Sort in descending order.

UserPrivilegeMaskType Constants

UserPrivilegeMaskType constants specify privileges in a security context.

Constant	Value	Description
_DYNAMIC_LIST_ADMIN	1	Can create and manage dynamic lists.
_PUBLIC_FOLDER_ADMIN	2	Can create / delete / read / write public folders used for queries, reports, and charts.
_SECURITY_ADMIN	3	Can access and manage secure records and fields. Also can edit the context group list field for a security context record and view all records.
_RAW_SQL_WRITER	4	Can create and use a SQL query using a raw SQL string.
_ALL_USERS_VISIBLE	5	Can view information for all users and groups from user databases.
_MULTI_SITE_ADMIN	6	MultiSite administrator privilege.
_SUPER_USER	7	Can perform all Active User, Schema Designer, User Administrator, Security Administrator, Public Folder Administrator, Dynamic List Administrator, and SQL Editor tasks. Can also create and delete databases and schemas and edit ClearQuest Web settings. The **admin** user account has Super User privilege.
_APP_BUILDER	8	Can create and modify schemas. Add record types, define and modify fields, create and modify states and actions, add hooks to the schema, and update existing databases. Create, modify, and save public queries, charts, and reports. Cannot perform User Administrator tasks.
_USER_ADMIN	9	Can create users and user groups and assign and modify their user-access privileges.

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.

WorkspaceFolderType Constants

The WorkspaceFolderType constants identify the type of a Workspace folder.

Constant	Value	Description
_WORKSPACE_PUBLIC_FOLDER	1	A Public folder.
_WORKSPACE_USER_FOLDER	2	A Personal folder.

WorkspaceItemType Constants

The WorkspaceItemType constants identify the type of a workspace item.

Constant	Value	Description
_WORKSPACE_QUERY	1	A query.
_WORKSPACE_CHART	2	A chart.
_WORKSPACE_FOLDER	3	A public or personal folder.
_WORKSPACE_FAVORITES	5	An item in Favorites.
_WORKSPACE_QUERY_PARAMETERS	6	A query parameter.
_WORKSPACE_PREFERENCES	7	A user preference.
_WORKSPACE_REPORT	9	A report.
_WORKSPACE_REPORT_FMT	10	A report format.
_WORKSPACE_STARTUP_BUCKET_ARRAY	11	An item that runs at startup.

Workspace Query Type Constants

Note: The following constants do not use the notational convention.

For VBScript, 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.

For Perl, the CQWKSPCQUERYTYPE constants identify the desired source of a query.

Constant	Value	Description
CQ_WKSPC_QUERIES_NONE	0	Do not return reports
CQ_WKSPC_SYSTEM_QUERIES	1	Return system reports only.
CQ_WKSPC_USER_QUERIES	2	Return user reports only.
CQ_WKSPC_BOTH_QUERIES	3	Return either system or user reports.

OLEWKSPCERROR Constants

The OLEWKSPCERROR constants identify errors that can be returned from Workspace-related operations.

Note: These error messages are for VBScript only.

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
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

Constant	Value
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

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 47

In addition to the examples of hooks and external applications provided in this chapter, see the following resources:

- Rational 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.

Attachments are files saved in a database. You can add any kind of file to a database using ClearQuest. For example, you can save text, word processing, spreadsheet, image, and diagram files.

When you save a file as part of a change request entity (record), you can also add a description of the file, which will make it easier to identify the file at a later time. Other information, such as the original file name, will also be stored, and the database will automatically create a unique identifier for the file.

Attachments, like other types of values, are held in a database field. The data type of a field holding attachments is AttachmentField. Because attachments are usually stored in logical groups, as when a defect is being discussed, an attachment field is a collection. The instances of the collection, each of which holds a single file, are of data type Attachment.

Methods to find and access attachment fields are available in a special object of type AttachmentFields. Each Entity always has an AttachmentFields object, even if no attachments are actually stored in it. Then, in each attachment field, there is an Attachments object that allows you to manage individual Attachment objects.

For other kinds of fields, you get field values by getting a FieldInfo object and then invoking GetValue() or GetValueAsList(). For GetValue(), a single string will be returned. If invoked on an attachment field, GetValue() might produce something meaningful if, say, your attachment is a one line text file. However, in general, using GetValue() on an attachment field will not produce a useful result. Instead, for attachments, you usually get values by first traversing to an Attachment object, then writing the file to disk and opening it with an application program.

Suppose that you have defined an Entity with two attachment fields. When you traverse this structure, you will iterate over two AttachmentField collections. To distinguish the collections, you can use the field names of each AttachmentField. To identify individual attachment files, you can use the descriptions in each instance of Attachment.

Schema Overview

Storage — In brief, an Entity has fields, each with a name and data type. A field can be of type AttachmentField. An AttachmentField can hold a collection of files, each stored in an individual Attachment object.

Management — To manage attachment fields, each Entity object always has an AttachmentFields object. Similarly, to manage individual attachments, each AttachmentField always has an Attachments object.

See “Attachments and Histories” on page 7 for more information and a diagram that illustrates hierarchy of objects.

- AttachmentFields

  An access object. There is exactly one per Entity, with methods that provide access to the fields containing attachments.

- AttachmentField

  A field that can hold collections of attachments. None or many attachment fields may be defined for an Entity. Each field can contain a collection of attachments. Each AttachmentField, as with other types of fields, has a name.

- Attachments

  An access object. There is exactly one per AttachmentField, with methods to count, get, add, and delete attachments.

- Attachment

  An object that contains a file and information about the file. There may be none or many in an attachment field. Information about the file includes a description, the original file name, and the file size.

To traverse from an Entity object to an Attachment, you must first get the AttachmentFields object. As you traverse, you can distinguish a general path using the AttachmentField names. Once you are at the level of an actual collection of attachments, you can identify individual attachments using Description and FileName values.

Schema Detail

Following is an overview of object members used to store and manage attachments. Actual method names and parameters will vary depending on whether you use Visual Basic or Perl.

Entity

An object corresponding to a database record. Members relevant to attachments are:

- AttachmentFields GetAttachmentFields()

  Get the access object. Every Entity object has exactly one AttachmentFields object with methods that manage attachment fields.

- GetFieldNames()

  Get the field names defined for this object. When you use the AttachmentFields object to make a traversal of attachment collections, you may want to know the field names in order to identify which path you want to follow. (Alternately, you can use indexes to get each AttachmentField.)

- long GetFieldType(fieldName)

  Get the data type of a field. If the return value is 7 (_ATTACHMENT_LIST), then the field is of type AttachmentField. This is useful if you have a field name but are not sure of its data type.

- GetAllFieldValues()

  Get information about all fields in this object. This is something you might normally do, because you fetch other kinds of values using FieldInfo.GetValue() and FieldInfo.GetValueAsList(). If you do have an array of FieldInfo objects, you can get each name with FieldInfo.GetName() and check each data type with FieldInfo.GetType().

Entity.AttachmentFields

An access object with useful methods. Members are:

- long Count()

  Get the number of fields of type AttachmentField, none or many.

- AttachmentField Item(index)

  Get an AttachmentField object using index number.

- AttachmentField ItemByName(name)

  Get an AttachmentField object using a FieldName.

Entity.AttachmentField

A field holding a collection of attachments. Members are:

- string GetFieldName()

  Get the name of the AttachmentField.

- collection GetAttachments()

  Get the collection of Attachment objects in this attachment field.

- array GetDisplayNameHeader()

  Get the unique keys for the Attachment objects.These keys are assigned by the database.

AttachmentField.Attachments

An access object containing useful methods. Members are:

- long Count()

  Get the number of attachments, none or many

- boolean Add(fileName, description)

  Add an attachment.

- boolean Delete(item)

  Delete an attachment using an index number.

- boolean Delete(name)

  Delete an attachment using the name held in the DisplayName property.

- Attachment Item(index)

  Get an attachment using an index number.

- Attachment ItemByName(name)

  Get an attachment using the name held in the DisplayName property.

AttachmentField.Attachment

An object containing a file. Members are:

- string GetDescription()

  Get a description of the attached file.

- void SetDescription(string)

  Set a description of the attached file.

- string GetDisplayName()

  Get the unique key assigned by the database.

- string GetFileName()

  Get the original file name.

- long GetFileSize()

  Get the file size.

- boolean Load(pathname)

  Write the file to a local file system.

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`**).
- 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.

Examples

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(Msg)

   Dim MySession ' As Session

   set MySession = GetSession()

   MySession.OutputDebugString & Msg & VbCrlf
```

```
end sub
REM End of Global Script DBGOUT
```

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).

Examples

VBScript

The following example selects all defects that belong to the *defect* record type.

```
set session = GetSession

set querydef = session.BuildQuery("defect")
querydef.BuildField("id")
querydef.BuildField("headline")


set resultset = session.BuildResultSet(querydef)
```

VBScript

The following example selects defects that match these criteria:

- "assigned to" user "johndoe"
- "beta2" planned release

This translates to:

(assigned_to = "johndoe") AND (planned_release = "beta2")

```
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)
```

VBScript

The following example selects defects that match these criteria:

- Planned release of beta
- Not in resolved or verified states
- Priority levels 1 or 2
- Assigned to a certain set of users

This translates to:

((planned_release = "beta") AND (state != resolved OR verified) AND (priority = 1 OR 2))
OR
(assigned_to = lihong OR gonzales OR nougareau OR akamoto)

```
set session = GetSession

Dim users(4)
users(0) = "lihong"
users(1) = "gonzales"
users(2) = "nougareau"
users(3) = "akamoto"

Dim priority_levels(2)
priority_levels(0)  = "1"
priority_levels(1)  = "2"

Dim states(2)
states(0)  = "resolved"
states(1)  = "verified"

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_OR)
set operator2 = operator.BuildFilterOperator(AD_BOOL_OP_AND)
operator2.BuildFilter "planned_release", AD_COMP_OP_EQ, "beta"
operator2.BuildFilter "state", AD_COMP_OP_NOT_IN, states
operator2.BuildFilter "priority", AD_COMP_OP_IN, priority_levels

operator.BuildFilter "assigned_to",AD_COMP_OP_IN, users

set resultset = session.BuildResultSet(querydef)
```

Perl

```
# ((planned_release = "beta") AND (state != resolved OR verified) AND (priority = 1 OR 2))
# OR
# (assigned_to = lihong OR gonzales OR nougareau OR akamoto)

$session = $entity->GetSession();

@users = ("lihong", "gonzales", "nougareau", "akamoto");
@priority_levels = ("1", "2");
@states = ("resolved", "verified");
@planned_release = ("beta");

$querydef = $session->BuildQuery("defect");
$querydef->BuildField("id");
$querydef->BuildField("component");
$querydef->BuildField("priority");
$querydef->BuildField("headline");

$operator = $querydef->BuildFilterOperator($CQPerlExt::CQ_BOOL_OP_OR);
$operator2 = $operator->BuildFilterOperator($CQPerlExt::CQ_BOOL_OP_AND);
$operator2->BuildFilter ("planned_release", $CQPerlExt::CQ_COMP_OP_EQ,
```

```
\@planned_release);
$operator2->BuildFilter ("state", $CQPerlExt::CQ_COMP_OP_NOT_IN, \@states);
$operator2->BuildFilter ("priority", $CQPerlExt::CQ_COMP_OP_IN,
\@priority_levels);

$operator->BuildFilter ("assigned_to",$CQPerlExt::CQ_COMP_OP_IN, \@users);

$resultset = $session->BuildResultSet(querydef);
```

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)
```

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 duplicate 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.

Examples

VBScript

```
Dim status
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 = entity.Validate
      if status = "" then
         entity.Commit
      else
         entity.Revert
      End If
   Next
End If
```

Perl

```
my($session);    # The current Session object
my($links); # The reference to the links collection object
my($link);
my($cnt);
```

```
my($itm);
my($childID);

if ($entity->HasDuplicates()) {
    $session = $entity->GetSession();
    $links = $entity->GetDuplicates();
  $session->OutputDebugString("links is " . $links . "(" . ref ($links) .
")\n" );
  $cnt = $links->Count();
  $session->OutputDebugString("count is " . $cnt . "(" . ref ($cnt) .
")\nchildren:\n" );
  for ($i = 0; $i<$cnt; $i++) {
    $itm = $links->Item($i);
   $session->OutputDebugString("Item is " . $itm . "(" . ref ($itm) . ")\n";
    $childID = $itm->GetChildEntityId();
    $session->OutputDebugString($childID . "\n" );
    }
   $session->OutputDebugString("done");
  }
else {
}
```

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:

- **`DumpFields`**, which prints out an entity's fields to the standard output
- **`ValidateAndCommit`**, which calls the Entity object's **`Validate`** and **`Commit`** methods.

Examples

VBScript

```
' subroutine for stateless records

Sub No_state(session) ' the Session Object
   Dim entity ' the Entity Object
   Dim failure ' a 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) 'session As Session
   Dim entity ' the Entity that is stateful
  ' failure message from functions that return strings
   Dim failure ' a String
   Dim failures ' an iterator containing list of failure reasons
   Dim id ' a 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

   StdOut "Test of stateful entities is done"
End Sub

REM Start of Global Script StdOut

sub StdOut(Msg)

   msgbox Msg

end sub

REM End of Global Script StdOut
```

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

    $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");
     $entity->Validate();
     $entity->Commit();
    $entity = "";

    print "Reload, and show final result";
    $entity = $session->GetEntity("project", "final project name");
    DumpFields($entity);
    $entity = "";

    print "Test for stateless entities is done";
}
```

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);
     $entity->Validate();
     $entity->Commit();
    $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!");
     $entity->Validate();
     $entity->Commit();

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

    $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 a routine called **`StdOut`**, which prints its arguments to a message box.

Examples

VBScript

```
Sub DumpOneEntityDef(edef) ' 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
REM Start of Global Script StdOut
sub StdOut(Msg)
   msgbox Msg
end sub
REM End of Global Script StdOut
```

Perl

```
use strict;
use CQPerlExt;

my $sessionObj = CQSession::Build();
$sessionObj->UserLogon("admin", "", "SAMPL", "");

my $entityDefNames = $sessionObj->GetEntityDefNames();

#Iterate over the record types
foreach my $edef_name (@$entityDefNames) {
    my $entityDefObj = $sessionObj->GetEntityDef($edef_name);
    print_edef($entityDefObj);
}

sub print_edef {
    my($edef)=@_;
    #  The parameter is an EntityDef object.

    my($names, $name);

    print "Dumping EntityDef ", $edef->GetName;
    print "\nFieldDefs:";

    $names = $edef->GetFieldDefNames;
    foreach $name (@$names) {
        print " " , $name , " type=" ,
        $edef->GetFieldDefType($name);
```

```
    }

    print "\nActionDefs: ";
    $names = $edef->GetActionDefNames;
    foreach $name (@$names) {
        print " " , $name , " type=" ,
        $edef->GetActionDefType($name);
    }

    if ($edef->GetType == $CQPerlExt::CQ_REQ_ENTITY) {
        #  stated record type
        print "\nEntityDef is a REQ entity def";
        print "\nStateDefs:";
        $names = $edef->GetStateDefNames;
        foreach $name (@$names) {
            print " " , $name;
        }
    }
    else {
        #  stateless record type
        print "\nEntityDef is an AUX entity def";
    }
    print "\n\n";
}
CQSession::Unbuild($sessionObj);
```

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 **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 message box.

Examples

VBScript

```
Sub DumpFieldInfo(info) ' 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

REM Start of Global Script StdOut

sub StdOut(Msg)

   msgbox Msg

end sub

REM End of Global Script StdOut
```

Perl

```
use CQPerlExt;


$CQsession = CQSession::Build();

$CQsession->UserLogon("admin", "", "perl", "");

$record = $CQsession->GetEntity("Defect", "perl00000001");

$fieldInfo = $record->GetFieldValue("id");


$temp = $fieldInfo->GetValueStatus();

if ($temp == $CQPerlExt::CQ_VALUE_NOT_AVAILABLE) {

   $status = "VALUE_NOT_AVAILABLE";

} elsif ($temp == $CQPerlExt::CQ_HAS_VALUE) {

   $status = "HAS_VALUE";

   $value = "'" & fieldinfo.GetValue() & "'";

} elsif ($temp == $CQPerlExt::CQ_HAS_NO_VALUE) {

   $status = "NO_VALUE";
```

```
} else {
   $status = "<invalid value status: " & temp & ">";
}
$temp = $fieldInfo->GetValidationStatus();
if ($temp == $CQPerlExt::CQ_KNOWN_INVALID) {
   $validity = "INVALID";
} elsif ($temp == $CQPerlExt::CQ_KNOWN_VALID) {
   $validity = "VALID";
} elsif ($temp == $CQPerlExt::CQ_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 = '", $fieldInfo->GetMessageText(), "'";
CQSession::Unbuild($CQsession);
```

Using Field Path Names to Retrieve Field Values

A field path name provides the path to a named Entity. You can use GetLocalFieldPathNames for a given record type and then use the returned fieldpaths to retrieve FieldInfo objects and their contents. These field paths use a dotted path notation (for example "owner.fullname").

When you call GetFieldValue to get a FieldInfo object, you normally do something like this, to get the value of the object:

```
Dim Owner
Owner = GetFieldValue("owner").GetValue()
```

If you wanted to get the full name of the owner and not the login name, you could write the following:

```
Dim MySession
Set MySession = GetSession()
Dim Owner
Owner = GetFieldValue("owner").GetValue()
Dim UserEntity
Set UserEntity = MySession.GetEntity("users", Owner)
Dim FullName
FullName = UserEntity.GetFieldValue("fullname").GetValue()
```

Using field path names, you can achieve the same result as follows:

```
Dim FullName
FullName = GetFieldValue("owner.fullname").GetValue()
```

For example, if a record type named Defect has a reference field Cfield to a record type named Customer and that record type has a reference field Ufield to a User record type with a field Name, then the field path of Name is:

```
"Defect\Cfield\Ufield\Name"
```

The field path name (or "dotted name) of Name is:

```
Defect.Cfield.Ufield.Name
```

You can use this path name to retrieve the value of Name. For example, using Perl:

```
$defect->GetFieldValue("Cfield.Ufield.Name")->GetValue();
```

You do not need the initial Defect if you already have a variable ($defect) referencing the Defect.

Showing Changes to a FieldInfo (Field)

The following example illustrates how to use the ClearQuest API to get information about field values that have changed for a given record.

Example

Perl

```
# entity->GetFieldsUpdatedThisAction returns all fields that have changed...
$oSess = $entity->GetSession();
$username = $oSess->GetUserLoginName();
$action = $entity->GetActionName();
$state = $entity->GetFieldValue("State")->GetValue();
my ($sec, $min, $hour, $mday, $mon, $year, $wday, $yday, $time) =
localtime();
$now = sprintf("%2.2d/%2.2d/%4d %2.2d:%2.2d:%2.2d", $mon + 1,$mday, $year +
1900, $hour, $min, $sec);
$body = "Fields Changed by $username, $action->$state, $now\n\n";
$fields = $entity->GetFieldsUpdatedThisAction();
for ($i = 0; $i < $fields->Count(); $i++) {
   $field = $fields->Item($i);
   $fname = $field->GetName();
   $newVal = &GetStrValue($oSess, $field);
   $oldVal = &GetStrValue($oSess, $entity->GetFieldOriginalValue($fname));
   unless($fname eq "Notes_Log"){
      unless($newVal eq $oldVal){
         $body .= "* $fname: $val\n";
      }
   }
}
$body .= "\n";
$fieldlog = $entity->GetFieldValue("FieldLog")->GetValue();
$newLog = $body . $fieldlog;
$entity->SetFieldValue("FieldLog", $newLog);

# Start of Global Script GetStrValue
# This is used to handle different types of fields...
sub GetStrValue {
   my ($session, $field) = @_;
   $val = "";
   $type = $field->GetType();
   if ($type lt 6) { #string, int, date, reference
      $val = $field->GetValue();
   } elsif ($type eq 6) { # Reference_list
      $list = $field->GetValueAsList();
```

```
      foreach $item (@$list) {
         $val .= "\n\t$item";
      }
   } elsif ($type eq 9) { #State
      $val = $field->GetValue();
   }
   return $val;
}
# End of Global Script GetStrValue
```

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.

Examples

VBScript

```
REM Start of Global Script ShowOldNewValues
REM TODO -- put your script code here

Sub ShowOldNewValues (actionname, hookname)
   Dim fieldnames
   Dim FN
   Dim OldFV
   Dim FieldValueStatus
   Dim FieldInfo
   Dim i
   Dim NewFV
   Dim FieldType
   Dim is_short
   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"
         ElseIf FieldType = AD_STATETYPE Then
            FieldType = "STATETYPE"
         ElseIf FieldType = AD_RECORDTYPE Then
            FieldType = "RECORDTYPE"
         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
```

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>";

      } elsif ($FieldValueStatus ==
                        $CQPerlExt::CQ_VALUE_NOT_AVAILABLE) {

         $OldFV = "<value not available>";

      } elsif ($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) {
   $NewFV = "<no value>";
} elsif ($FieldValueStatus ==
         $CQPerlExt::CQ_VALUE_NOT_AVAILABLE) {
   $NewFV = "<value not available>";
} elsif ($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) {
   $FieldType = "Short String";
} elsif ($FieldType == $CQPerlExt::CQ_MULTILINE_STRING) {
   $FieldType = "Multiline String";
   $is_short = 0;
} elsif ($FieldType == $CQPerlExt::CQ_INT) {
   $FieldType = "Integer";
} elsif ($FieldType == $CQPerlExt::CQ_DATE_TIME) {
   $FieldType = "Date Time";
} elsif ($FieldType == $CQPerlExt::CQ_REFERENCE) {
   $FieldType = "Reference";
} elsif ($FieldType == $CQPerlExt::CQ_REFERENCE_LIST) {
   $FieldType = "Reference List";
   $is_short = 0;
} elsif ($FieldType == $CQPerlExt::CQ_ATTACHMENT_LIST) {
   $FieldType = "Attachment List";
   $is_short = 0;
} elsif ($FieldType == $CQPerlExt::CQ_ID) {
   $FieldType = "ID";
} elsif ($FieldType == $CQPerlExt::CQ_STATE) {
   $FieldType = "State";
} elsif ($FieldType == $CQPerlExt::CQ_JOURNAL) {
   $FieldType = "Journal";
   $is_short = 0;
} elsif ($FieldType == $CQPerlExt::CQ_DBID) {
   $FieldType = "DBID";
```

```
      } elsif ($FieldType == $CQPerlExt::CQ_STATETYPE) {
         $FieldType = "STATETYPE";
      } elsif ($FieldType == $CQPerlExt::CQ_RECORDTYPE) {
         $FieldType = "RECORDTYPE";
      } 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 $session->OutputDebugString().
   # Here we call a subroutine 'DBGOUT' which writes the message
   # out to a file and invokes 'notepad' on it...
   DBGOUT($M);
}
```

```
# End of Global Script ShowOldNewValues

# 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
```

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 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: **`StdOut`**, which prints its arguments to a file, and **`ToStr`** (not included here), which converts its argument to a string.

Following the VBScript and Perl code examples are additional Perl code examples that illustrate:

- Listing all the defect records in a ClearQuest user database and modifying one of the records. See, *Getting a List of Defects and Modifying a Record*
- Sorting the result set by using methods of the QueryFieldDef object. See, *Sorting a Result Set*

Examples

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 ' 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
         ' ToStr converts the argument to a string
         StdOut " " & rsltset.GetColumnLabel(column) & "=" & _
            ToStr(rsltset.GetColumnValue(column))
```

```
        column = column + 1
      Loop

      StdOut ""
      status = rsltset.MoveNext
   Loop
End Sub

REM Start of Global Script StdOut

sub StdOut(Msg)

   msgbox Msg

end sub

REM End of Global Script StdOut
```

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 == CQPerlExt::CQ_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 a List of Defects and Modifying a Record

The following Perl example is an external program that lists all the defect records in a ClearQuest user database, and modifies one of the records. The program:

- Lists all the defect records in the SAMPL database, and selects “id,” “Headline,” and “State” as display fields.
- Modifies the “Description” field of the defect record SAMPL00000012 to be “This defect has been modified.”

```
use CQPerlExt;

#Getting the session
my $session = CQSession::Build();
CQSession::UserLogon ($session, "admin", "", "SAMPL", "Lab3");

my $querydef = $session->BuildQuery ("defect");
$querydef->BuildField ("id");
$querydef->BuildField ("headline");
my $resultset = $session->BuildResultSet ($querydef);
$resultset->Execute();

while (($resultset->MoveNext()) == 1)
{
   $id   = $resultset->GetColumnValue(1);
   $rec  = $session->GetEntity("Defect", $id);
   $head = $rec->GetFieldValue("Headline")->GetValue();
   $state= $rec->GetFieldValue("State")->GetValue();
   print "$id, $head, $state. \n";

   if ($id eq "SAMPL00000012") {
     $session->EditEntity($rec, "Modify");
     $rec->SetFieldValue("description", "This defect has been modified.");
     $rec->Validate();
```

```
     $rec->Commit();
   }
}
CQSession::Unbuild($session);
```

Sorting a Result Set

You can use methods of the QueryFieldDef object to customize the sort order or of a result set as the following example illustrates. It sets the sort precedence on the id field and the sort order is ascending.

Example

Perl

```
use CQPerlExt;
#Start a ClearQuest session
#$AdminSession= CQAdminSession::Build();
$SessionObj = CQSession::Build();

$dbsetname = "CQMS.SAMPL.HOME";
#Refresh list of accessible databases
$databases = $SessionObj->GetAccessibleDatabases("MASTR", "", $dbsetname);

#Log into a database
$SessionObj->UserLogon("admin","","SAMPL",$dbsetname);

#Create a Query
$querydef = $SessionObj->BuildQuery("defect") ;
$querydef->BuildField("id") ;
$querydef->BuildField("headline") ;
$querydef->BuildField("owner.login_name") ;
$querydef->BuildField("submit_date") ;
#Create the queryfilternode object:
# where (state not in closed AND (id = 1 OR id = 2))
$where = $querydef->BuildFilterOperator($CQPerlExt::CQ_BOOL_OP_AND);
@states = ("closed");
$where->BuildFilter("state", $CQPerlExt::CQ_COMP_OP_NEQ, \@states);
$subor = $where->BuildFilterOperator($CQPerlExt::CQ_BOOL_OP_OR);
@id1 = ("SAMPL00000001");
$subor->BuildFilter("id", $CQPerlExt::CQ_COMP_OP_EQ, \@id1);
@id2 = ("SAMPL00000002");
$subor->BuildFilter("id", $CQPerlExt::CQ_COMP_OP_EQ, \@id2);

# Get the collection QueryFieldDef objects
```

```
$queryfielddefs = $querydef->GetQueryFieldDefs();
# Select the id field and set the sort type and order
$idfield = $queryfielddefs->ItemByName("id");
$idfield->SetSortType($CQPerlExt::CQ_SORT_DESC);
# this is for if you have mulplte sort columns, which takes precedence:
$idfield->SetSortOrder(1);
# Select the submit_date field and set the week function on it
$datefield = $queryfielddefs->ItemByName("submit_date");
$datefield->SetFunction($CQPerlExt::CQ_DB_WEEK_FUNC);

$resultset = $SessionObj->BuildResultSet($querydef);
$ct = $resultset->ExecuteAndCountRecords();
for ($i = 0; $i < $ct; $i++) {
   $resultset->MoveNext();
   print $resultset->GetColumnValue(1);
   print " ";
   print $resultset->GetColumnValue(2);
   print " ";
   print $resultset->GetColumnValue(3);
   print " ";
   print $resultset->GetColumnValue(4);
   print "\n";
}
CQAdminSession::Unbuild($AdminSession);
CQSession::Unbuild($SessionObj);
```

Getting Session and Database Information

The following code from an **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 called **`StdOut`**, which prints its arguments to a message box.

Examples

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 "joe","", dbname, AD_PRIVATE_SESSION, ""

Set dbDesc = session.GetSessionDatabase
StdOut "DB name = " & dbDesc.GetDatabaseName
StdOut "DB set name = " & dbDesc.GetDatabaseSetName

' You must log in with superuser privilege or an error will be
' generated by GetDatabaseConnectString
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
   for each onename in userGroups
      StdOut " group " & onename
   next
End If

REM Start of Global Script StdOut

sub StdOut(Msg)

   msgbox Msg

end sub

REM End of Global Script StdOut
```

Perl

```
use lib "E:\\Program Files\\Rational\\common\\lib";
```

```
use CQPerlExt;

$CQsession = 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 = $CQsession->GetUserGroups();
if (!@$userGroups) {
   #Code to handle if no user groups exist
   print "This user does not belong to any groups\n";
   }
else {
    # print out all groups
    foreach $groupname (@$userGroups) {
        print "Group $groupname\n";
    }
}
CQSession::Unbuild($CQsession);
```

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* includes *defects*, *enhancement requests*, and *documentation requests*
- *Work orders* includes *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*)

Examples

VBScript

```
Dim qryDef ' a QueryDef object
Dim resultSet ' a Resultset object
Dim familyEntDef ' an EntityDef object
Dim families ' As Variant
Dim session ' a Session object
Dim i ' As String

' 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)
End If
```

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;
}
```

Creating a Dependent Choice List

ClearQuest allows you to specify that the values of one field (**dependent field**) depend upon the values of another field (**parent field**). This is accomplished by defining a Choice_List hook on the dependent field that sets its choice list based upon the value of the parent field.

In this example, you have two fields: **Platform** and **Version**. **Platform** is the parent field and has a constant (enumerated) choice list. **Version** is the dependent field which calculates the appropriate choice list based on the value of **Platform**.

Version's Choice list hook gets *recalculated* every time **Platform** changes because its Recalculate Choice List option is set.

Note that any field change triggers the hook to be rerun.

- If *any* field changes, all fields with Recalculate Choice List set will rerun their Choice List hook, even if the choice list does not depend on the changed field.
- To have the choice list update only when its dependent field actually changes, use the **`GetFieldOriginalValue`** method to check the parent field.

In general, Recalculate Choice List should only be set for fields which have a Choice List Hook defined, and should not be set on those that do not.

This Choice List Hook code determines the enumerated content of the choice list based upon the value of the parent field, Platform.

In the following examples:

- The value of the parent field is a SHORT_STRING data type. For VBScript, during the CASE selection, the enumerated values of Platform must to be *identical* to the string, shown here, including the blank spaces. Otherwise, the related list is not generated.
- Before executing this code, the choice list is defaulted to an empty list. If none of these cases match, there will be *no* choice list.

Examples

VBScript

```
' Add field choices for platforms
Dim platform
platform = GetFieldValue("platform").GetValue ()
select case platform
   case "Windows 2000"
      choices.AddItem ("Professional")
      choices.AddItem ("Professional SP1")
      choices.AddItem ("Server")
      choices.AddItem ("Server SP1")
   case "Windows NT Server"
      choices.AddItem ("4.0")
      choices.AddItem ("4.0 SP6A")
   case "Windows 98"
      choices.AddItem ("Win98")
end select
```

Perl

```
my $platform;
$platform = ($entity->GetFieldValue("platform"))->GetValue();
if ($platform eq "Windows NT Workstation") {
   push(@choices, "3.51", "4.0", "4.0 SP2", "4.0 SP3");
} else {
   if ($platform eq "Windows NT Server") {
               push(@choices, "4.0", "4.0 SP3");
   } else {
        if ($platform eq "Windows 95") {
           push(@choices, "Win95");
        } else {
           push(@choices, " ");
      }
 }
}
```

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 e-mail 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 e-mail 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.

Examples

VBScript

```
Sub Defect_Notification(actionname, actiontype)
   Dim cqSes ' a Session object
   Dim entDef ' an EntityDef object
   Dim actionname ' As String
   Dim 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
```

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}
}
```

Adding and Removing Users in a Group

You can add users to a group and remove users from a group by creating an AdminSession and then logging in as an admin.

The following code examples illustrate adding and removing users from groups.

Examples

VBScript

```
' Initiate an admin session and log on as "admin"...
Dim adminSession ' an AdminSession object
Dim GCol ' a Groups collection object
Dim numGs ' As Long
Dim M ' As String
Dim g ' As Long
Dim GObj ' a Group Object
Dim UCol ' a Users collection object
Dim u ' As Long
Dim UObj ' aUser object
set adminSession = CreateObject("ClearQuest.AdminSession")
adminSession.Logon "admin", "", "LOCALTEST"
Function ShowUsersInGroups(Tag)
set GCol = adminSession.Groups
numGs = GCol.Count
M = Tag & VbCrLf
If numGs = 0 Then
   M = Tag & ", there are no groups" & VbCrLf
Else
   M = Tag & ", there are " & numGs & " group(s)." & VbCrLf
   For g = 0 to (numGs - 1)
      set GObj = GCol.Item(g)
      M = M & "Group=" & GObj.Name & " Users="
      set UCol = GObj.Users
      For u = 0 to (UCol.Count - 1)
         set UObj = UCol.Item(u)
         M = M & UObj.Name & " "
      Next
      M = M & VbCrLf
   Next
End If
MsgBox M
End Function
```

```
' Display the starting state...
call ShowUsersInGroups("At the beginning of this program's execution")
' Get the Group and User object handles...
set GObj = adminSession.GetGroup("MyGroup")
set UObj = adminSession.GetUser("QE")
' Add user to group:
GObj.AddUser UObj
call ShowUsersInGroups("After adding user 'QE' to group 'MyGroup'")
' Remove user from group:
GObj.RemoveUser UObj
call ShowUsersInGroups("After removing user 'QE' from group 'MyGroup'")
```

Perl

```
use CQPerlExt;
$DEBUG = 1;
sub ShowUsersInGroups() {
local($Tag) = shift;
my($GCol) = $adminSession->GetGroups();
my($GCount) = $GCol->Count();
if ($GCount == 0) {
   print "\n$Tag, there are no groups.\n\n";
   }
else {
   print "\n$Tag, there " .
   (($GCount == 1) ? "is $GCount group.\n":"are $GCount groups.\n");
   print "Group Users in group\n" .
   "========== ========================================\n";
   for ($i = 0; $i < $GCount; $i++) {
      my($GObj) = $GCol->Item($i);
      printf("%-10.10s ", $GObj->GetName());
   # Display the list of users in the group...
      my($UCol) = $GObj->GetUsers();
      if ($UCol->Count() == 0) {
         print "(none)\n";
         }
      else {
         for (my($u) = 0; $u < $UCol->Count(); $u++) {
            my($UObj) = $UCol->Item($u);
            print ($UObj->GetName() . " ");
            }
         print ("\n");
```

```
        }
    }
print "\n";
}
}

# Initiate an admin session and log on as "admin"...
$adminSession= CQAdminSession::Build();
or die "Error creating AdminSession object.\n";
print "Admin session create OK\n" if ($DEBUG);
$adminSession->Logon("admin", "", "LOCALTEST");
print "Back from logging in to admin session\n" if ($DEBUG);
# Display the users who are members of groups before we do anything...
&ShowUsersInGroups("At the beginning of this program's execution");
# Get handles for the 'QE' user and the 'MyGroup' group...
$GObj = $adminSession->GetGroup("MyGroup");
$UObj = $adminSession->GetUser("QE");
# Add user "QE" to the "MyGroup" group...
$GObj->AddUser($UObj);
# Display the users who are members of groups...
&ShowUsersInGroups("After adding user 'QE' to group 'MyGroup'");
# Remove the user from the group...
$GObj->RemoveUser($UObj);
# Display the users in groups now...
&ShowUsersInGroups("After removing user 'QE' from group 'MyGroup'");
CQAdminSession::Unbuild($adminSession);
```

Upgrading User Information

The following examples use the UpgradeInfo method (of the User object) to optimize peformance when setting User properties and updating the databases to which a user is subscribed.

These examples update a user profile and upgrade all the databases that the User is subscribed to. They also verify the update by attempting to log into each accessible database.

For more information, see **`UpgradeInfo`**.

Examples

VBScript

```
Dim  dbset
Dim  adminUser
Dim  adminPasswd
Dim  userName
Dim  passwd
Dim  adminSession
Dim  userList
Dim  userObj
Dim  email
Dim  phone
Dim  fullname
Dim  failed_dbs
Dim  session
Dim  dbdescs
Dim  db
Dim  dbname

dbset = "2003.06.00"
adminUser = "admin"
adminPasswd = ""
' Create a ClearQuest admin session
set adminSession = CreateObject("CLEARQUEST.ADMINSESSION")
If (adminSession.Logon (adminUser, adminPasswd, dbset)) Then
    MsgBox "Could not get Admin session login"
End If

' the user who we want to change user profile
userName = "QE1"
passwd = "secret"
'Get the User
set userObj = adminSession.GetUser(userName)
If userObj is Nothing Then
```

```
   'Do NOT use MsgBox if you are using a ClearQuest Web server
    MsgBox "Error: Failed lookup for user" & userName
End If

 email = "qe@rational.com"
 phone = "123456789"
 fullname = "Best Quality Engineer"
 userObj.Password(passwd)
 userObj.Email(email)
 userObj.Phone(phone)
 userObj.FullName(fullname)

' Upgrade all the databases that the user is subscribed to
MsgBox "Upgrade user profile for userName"
failed_dbs = userObj.UpgradeInfo
if (failed_dbs > 0) then
    MsgBox "Set password failed in database(s)" & failed_dbs
end if

' verify the new password by logging into databases
set session = CreateObject("ClearQuest.Session")
dbdescs = session.GetAccessibleDatabases("MASTR", userName, dbset)
for each db in dbdescs
    dbname = db.GetDatabaseName
    session.UserLogon userName, passwd, dbname, AD_PRIVATE_SESSION, dbset
    If session is Nothing then
   MsgBox "Could not get session login to db" & dbname
    End If
    MsgBox "Checking user info in db" & dbname & ":"
    ' verify the new phone, fullname, and email
    if (email <> session.GetUserEmail) then
   MsgBox "Email is not upgraded in" & dbname
    end if
    if (fullname <> session.GetUserFullName) then
   MsgBox "Full name is not upgraded in" & dbname
    end if
    if (phone <> session.GetUserPhone) then
   MsgBox "Phone is not upgraded in" & dbname
    end if
next
```

Perl

```
use CQPerlExt;
use Time::Local;
$dbset = "2003.06.00";
$adminUser = "admin";
$adminPasswd = "";

# Create a ClearQuest admin session
my $adminSession= CQAdminSession::Build();
if ($adminSession->Logon( $adminUser, $adminPasswd, $dbset )) {
    print "Could not get Admin session login\n";
    exit 1;
   }
# the user who we want to change user profile
my ($userName, $passwd) = ("testuser", "password");

# Get the collection of users associated with the schema repository.
my $userObj = $adminSession->GetUser($userName);
unless ($userObj) {
    print "$0: Error: Failed lookup for user \"$userName\"\n";
    exit 1;
   }
$email="qe\@rational.com";
$phone="123456789";
$fullname="Best Engineer";
$userObj->SetPassword($passwd);
$userObj->SetEmail($email);
$userObj->SetPhone($phone);
$userObj->SetFullName($fullname);

# Upgrade all the databases that the user is subscribed to
print "Upgrade user profile for $userName\n";
$failed_dbs = $userObj->UpgradeInfo();
@temp = @$failed_dbs;
if ($#temp > -1)
   {
     printf "fail dbs: %s.\n", join(',', @temp);
   }
# verify the new password by logging into databases
$session = CQSession::Build();
```

```
$dbdescs = $session->GetAccessibleDatabases("MASTR", $userName, $dbset);
foreach $i (0 .. $dbdescs->Count()-1)
{
    $dbname = $dbdescs->Item($i)->GetDatabaseName();
    $session->UserLogon($userName, $passwd, $dbname, $dbset);
    unless (defined $session)
    {
     print LOG "Could not get session login to db $dbname\n";
     exit 1;
    }
    print "Checking user info in db $dbname:\n";
    # verify the new phone, fullname, and email
    if ($email ne $session->GetUserEmail()) {
     print "Email is not upgraded in $dbname\n";
      }
    if ($fullname ne $session->GetUserFullName()) {
     print "Full name is not upgraded in $dbname\n";
      }
    if ($phone ne $session->GetUserPhone()) {
     print "Phone is not upgraded in $dbname\n";
      }
    else {
     print "User information verified.\n";
     }
}
```

Index

A

B

C

D

E

F

G

H

I

L

M

N

O

P

Q

R

S

T

U

V

W