Contents

Using the ClearQuest API ............................................. 13
Using this reference manual ........................................... 14
Understanding the ClearQuest API .................................... 16
Understanding ClearQuest API objects ......................... 20
Working with sessions .................................................. 25
Working with queries .................................................... 29
Working with records ................................................... 33
Understanding user database objects ............................. 37
Accessing the schema repository .................................. 48
Understanding schema repository objects ..................... 53
Understanding the schema repository collection objects .... 61
Understanding additional database objects .................... 64
Glossary ................................................................. 79
Examples of hooks and scripts ................................. 95
Enumerated Constants ................................................ 123
Session object ........................................................ 131
BuildEntity method .................................................... 136
BuildQuery method ................................................. 138
BuildResultSet method ............................................. 140
BuildSQLQuery method ............................................. 142
DeleteEntity method .................................................. 144
EditEntity method ...................................................... 146
FireRecordScriptAlias method .................................. 148
GetAccessibleDatabases method ................................ 149
GetAuxEntityDefNames method ................................ 151
GetDefaultEntityDef method ................................... 153
GetEnabledEntityDefs method .................................. 154
GetEnabledPackageRevs method ........................................ 155
GetEntity method ......................................................... 156
GetEntityByDbId method ............................................... 158
GetEntityDef method ..................................................... 160
GetEntityDefFamily method .......................................... 162
GetEntityDefFamilyNames method ................................. 163
GetEntityDefNames method ........................................... 164
GetInstalledMasters method .......................................... 166
GetQueryEntityDefNames method ................................ 168
GetReqEntityDefNames method ...................................... 170
GetServerInfo method .................................................. 172
GetSessionDatabase method ........................................... 173
GetSubmitEntityDefNames method ................................. 174
GetUserEmail method .................................................... 176
GetUserFullName method ............................................... 178
GetUserGroups method ................................................ 180
GetUserLoginName method ............................................. 182
GetUserMiscInfo method ............................................... 184
GetUserPhone method .................................................. 186
GetWorkSpace method ................................................... 188
HasValue method .......................................................... 189
IsMetadataReadonly method ........................................ 190
MarkEntityAsDuplicate method .................................... 191
OpenQueryDef method .................................................. 193
OutputDebugString method ............................................ 194
UnmarkEntityAsDuplicate method ................................. 195
UserLogon method ........................................................ 197
Entity object ............................................................... 199
AddFieldValue method .................................................. 206
BeginNewFieldUpdateGroup method .............................. 208
Commit method .......................................................... 210
DeleteFieldValue method ............................................... 212
FireNamedHook method ...........................................214
GetActionName .....................................................216
GetActionType .....................................................217
GetDuplicates method ...........................................218
GetFieldValues method ..........................................220
GetDbId method ....................................................221
GetDefaultActionName ..........................................222
GetDisplayName method ..........................................223
GetDuplicates method ...........................................225
GetEntityDefName method ......................................227
GetFieldChoiceList method ....................................229
GetFieldChoiceType method ....................................231
GetFieldMaxLength method ......................................233
GetFieldNames method ..........................................234
GetFieldOriginalValue method ..................................236
GetFieldRequiredness method ..................................238
GetFieldsUpdatedThisAction method .........................240
GetFieldsUpdatedThisGroup method .........................242
GetFieldsUpdatedThisSetValue method .......................244
GetFieldType method .............................................246
GetFieldValue method .............................................248
GetInvalidFieldValues method .................................250
GetLegalActionDefNames method ..............................251
GetOriginal method ..............................................253
GetOriginalID method ...........................................255
GetSession method ..............................................257
GetType method ...................................................259
HasDuplicates method ............................................261
InvalidateFieldChoiceList function ..........................263
IsDuplicate method ..............................................264
IsEditable method .................................................266
IsOriginal method ...............................................268
LookupStateName method ........................................ 270
Revert method ..................................................... 271
SetFieldChoiceList function ...................................... 273
SetFieldRequirednessForCurrentAction method ............ 275
SetFieldValue method ............................................. 277
Validate method .................................................. 279
EntityDef object .................................................. 281
DoesTransitionExist method .................................... 283
GetActionDefNames method ...................................... 285
GetActionDefType method ......................................... 287
GetActionDestStateName method ................................. 289
GetFieldDefNames method ....................................... 290
GetFieldDefType method .......................................... 292
GetFieldReferenceEntityDef method ......................... 294
GetHookDefNames method ....................................... 296
GetLocalFieldPathNames method ............................... 298
GetName method .................................................. 299
GetStateDefNames method ...................................... 300
GetType method ................................................... 302
IsActionDefName method ........................................ 304
IsFamily method ................................................ 305
IsFieldDefName method .......................................... 306
IsStateDefName method ......................................... 307
IsSystemOwnedFieldDefName method ....................... 308
QueryDef object .................................................. 309
BuildField method ............................................... 316
BuildFilterOperator method ..................................... 318
Save method .................................................... 320
ResultSet Object ................................................. 321
AddParamValue method ......................................... 322
ClearParamValues method ....................................... 323
Execute method .................................................. 324
GetColumnLabel method ..................................326
GetColumnType method ..................................327
GetColumnValue method ..................................328
GetNumberOfColumns method .........................330
GetNumberOfParams method ..............................331
GetParamChoiceList method ............................332
GetParamComparisonOperator method ................333
GetParamFieldType method ..............................334
GetParamLabel method ..................................335
GetParamPrompt method ................................336
GetRowEntityDefName method ..........................337
GetSQL method ............................................338
LookupPrimaryEntityDefName method ...............339
MoveNext method ........................................340
QueryFilterNode object .................................341
BuildFilter method ......................................342
BuildFilterOperator method ...........................344
AdminSession object ..................................345
CreateDatabase method ................................355
CreateGroup method ....................................357
CreateUser method .......................................358
DeleteDatabase method ................................359
GetDatabase method ....................................361
GetGroup method ........................................363
GetUser method ..........................................365
Logon method .............................................367
Database object .........................................369
SetInitialSchemaRev method ...........................390
Upgrade method .........................................391
UpgradeMasterUserInfo method .......................392
Schema object ..........................................393
SchemaRev object .......................................397
<table>
<thead>
<tr>
<th>Method/Collection</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetEnabledEntityDefs</td>
<td>401</td>
</tr>
<tr>
<td>GetEnabledPackageRevs</td>
<td>402</td>
</tr>
<tr>
<td>User object</td>
<td>403</td>
</tr>
<tr>
<td>SubscribeDatabase method</td>
<td>416</td>
</tr>
<tr>
<td>UnsubscribeAllDatabases method</td>
<td>417</td>
</tr>
<tr>
<td>UnsubscribeDatabase method</td>
<td>418</td>
</tr>
<tr>
<td>Group object</td>
<td>419</td>
</tr>
<tr>
<td>SubscribeDatabase method</td>
<td>425</td>
</tr>
<tr>
<td>UnsubscribeAllDatabases method</td>
<td>426</td>
</tr>
<tr>
<td>UnsubscribeDatabase method</td>
<td>427</td>
</tr>
<tr>
<td>Databases collection object</td>
<td>429</td>
</tr>
<tr>
<td>Item method</td>
<td>431</td>
</tr>
<tr>
<td>EntityDefs collection object</td>
<td>433</td>
</tr>
<tr>
<td>Item method</td>
<td>435</td>
</tr>
<tr>
<td>Groups collection object</td>
<td>437</td>
</tr>
<tr>
<td>Item method</td>
<td>439</td>
</tr>
<tr>
<td>Schemas collection object</td>
<td>441</td>
</tr>
<tr>
<td>Item method</td>
<td>443</td>
</tr>
<tr>
<td>SchemaRevs collection object</td>
<td>445</td>
</tr>
<tr>
<td>Item method</td>
<td>447</td>
</tr>
<tr>
<td>Users collection object</td>
<td>449</td>
</tr>
<tr>
<td>Item Method</td>
<td>451</td>
</tr>
<tr>
<td>Attachment-Related Objects</td>
<td>453</td>
</tr>
<tr>
<td>AttachmentFields collection object</td>
<td>455</td>
</tr>
<tr>
<td>Item Method</td>
<td>457</td>
</tr>
<tr>
<td>Attachment object</td>
<td>459</td>
</tr>
<tr>
<td>Load method</td>
<td>468</td>
</tr>
<tr>
<td>AttachmentField object</td>
<td>471</td>
</tr>
<tr>
<td>Attachments collection object</td>
<td>477</td>
</tr>
<tr>
<td>Add method</td>
<td>480</td>
</tr>
<tr>
<td>Delete method</td>
<td>482</td>
</tr>
<tr>
<td>Item method</td>
<td>484</td>
</tr>
</tbody>
</table>
Sort method ............................................. 545
Link object ............................................. 547
GetChildEntity method ................................. 548
GetChildEntityDef method ............................. 549
GetChildEntityDefName method ....................... 550
GetChildEntityID method ............................... 551
GetParentEntity method ................................. 552
GetParentEntityDef method ............................. 553
GetParentEntityDefName method ....................... 554
GetParentEntityID method ............................... 555
OleDbMailMsg object .................................. 557
  AddBcc method ....................................... 558
  AddCc method ....................................... 559
  AddTo method ....................................... 560
  ClearAll method ..................................... 561
  Deliver method ..................................... 562
  MoreBody method ..................................... 563
  SetBody method ..................................... 564
  SetFrom method ..................................... 565
  SetSubject method ................................... 566
CHARTMGR object ...................................... 567
  MakeJPEG method ..................................... 575
  MakePNG method ..................................... 576
  SetResultSet method ................................ 577
ReportMgr object ...................................... 579
  ExecuteReport method ................................ 580
  GetQueryDef method .................................. 581
  SetHTMLFileName method ............................... 582
WORKSPACE object ..................................... 583
  GetAllQueriesList method ............................... 584
  GetChartDef method .................................. 585
  GetChartList method .................................. 586
Using the ClearQuest API

This chapter introduces you, the ClearQuest Administrator, to the ClearQuest application programming interface (API) you can use to customize ClearQuest with code (hook scripts and external applications). This chapter includes the following sections:

- “Using this reference manual” on page 14, which provides a documentation roadmap and tells you where to find examples.
- “Understanding the ClearQuest API” on page 16, which indicates what knowledge this reference assumes you have, explains what the ClearQuest API is, and provides guidelines for how you can use the API.
- “Working with sessions” on page 25, which explains the fundamental object your code always addresses first.
- “Working with queries” on page 29, which explains how to fetch data from the user database.
- “Working with records” on page 33, which explains how to manipulate data in the user database.
- “Understanding user database objects” on page 37, which provides detailed remarks for the objects you use to work with records and queries.
- “Accessing the schema repository” on page 48, which explains how to work with metadata in the schema repository (master database).
- “Understanding schema repository objects” on page 53, which explains how to work with user data (records and queries).
- “Understanding the schema repository collection objects” on page 61, which provides remarks for accessing multiple schema repository objects.
- “Understanding additional database objects” on page 64, which provides remarks for attachments, database descriptions, record history, and more.
Using this reference manual

This introductory section to the introductory chapter of the ClearQuest API Reference provides an overview of ClearQuest documentation, where you can find examples of code that uses the API, and how to use this reference.

You can get more information about administering ClearQuest, and about the ClearQuest API. For an introduction to hook scripts, see the “Using hooks to customize your workflow” chapter in *Administering Rational ClearQuest*.

ClearQuest documentation set

ClearQuest includes the following documentation, and we recommend that the ClearQuest administer be familiar with each document.

Finding examples

For an introduction to hook scripts, see the chapter entitled “Using hooks to customize your workflow” in *Administering Rational ClearQuest*, which includes some sample code.
For additional examples of hooks and external applications, see the following.

- the chapter entitled Examples of hooks and scripts in this reference
- 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.

**Using this reference**

We recommend that, the ClearQuest administrator interested in writing hook scripts and external applications, do the following:

- Read this introductory chapter before you begin working with the API
- Have the ClearQuest documentation set available to you

In addition, we recommend that you read the following documents in the following sequence:

1. *Introducing Rational ClearQuest*, a brief overview of the entire product.
2. ClearQuest Designer Tutorial, available from the Help menu of ClearQuest Designer, which provides explanations and hands-on exercises.
3. “Using hooks to customize your workflow”, the scripting overview chapter in *Administering Rational ClearQuest*.
4. The Working with hooks chapter of ClearQuest Designer Help, which provides overview material, as well as procedures and sample code.
5. “Using the ClearQuest API” (this chapter), the definitive introduction to the API for administrators writing hook scripts or external applications.
Understanding the ClearQuest API

The Rational ClearQuest API Reference provides you, the ClearQuest administrator and schema designer, with a comprehensive overview of the application programming interface (API). The ClearQuest API is a set of objects and methods that you use to write code that directs ClearQuest to meet your organization’s needs.

Knowledge assumed

The ClearQuest API Reference assumes that you are familiar with

- scripting in VBScript or Perl, if you are writing hook scripts
- the programming language you want to use, if you are writing external applications
- relational database concepts, such as queries, tables, and unique keys

Ways to use the ClearQuest API

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

<table>
<thead>
<tr>
<th>Type of code</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hook scripts for your ClearQuest schema</td>
<td>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.)</td>
</tr>
<tr>
<td>External applications that run outside of ClearQuest</td>
<td>View or modify the data ClearQuest stores in the user database and schema repository.</td>
</tr>
</tbody>
</table>
Organization of the API Reference

The *Rational ClearQuest API Reference* consists of these sections:

- Using the ClearQuest API (this chapter), which explains how to use this reference and provides a conceptual overview to help you take advantage of the ClearQuest API.
- Glossary, which provides a list of keys terms and their definitions.
- Examples of hook scripts and external applications, which provide sample code.
- Reference Tables (the main section of this reference), which provide details of syntax and usage for each API object. This section also includes the Enumerated Constants.
- Index, which provides an alphabetized listing of every class, method, and property.

Choosing a scripting language

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

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

**Using Perl**

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

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

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

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

**Note:** Rational Software has no relation to this site.

Using Perl for external applications

If you are planning to write an external application in Perl, make sure that it does not invoke an action that triggers a Perl hook. Otherwise, the version of Perl that ClearQuest uses causes the external application to fail. For this release, if you want to 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.

We recommend you execute your external applications using the ClearQuest Perl engine, CQPerl.

Notation Conventions for Perl

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

<table>
<thead>
<tr>
<th>Prefix</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CQ</td>
<td>Prefix for objects that the ClearQuest API can access through its CQPerlExt package. For example: CQEntity</td>
</tr>
<tr>
<td>$CQPerlExt::CQ</td>
<td>Prefix for Perl Enumerated Constants. For example, $CQPerlExt::CQ_ORACLE</td>
</tr>
</tbody>
</table>

This document shows the syntax of Perl using the “get” and/or “set” prefix for calls to a property. All the Perl “get” calls to a property return a value. The “Variant” datatype is unique to VBScript/Visual Basic.
Using VBScript

VBScript, a subset of Microsoft Visual Basic, offers a convenient solution for ClearQuest scripting within the Windows environment. For example, you might find it easy to move certain code sections between a Visual Basic external application and a VBScript hook script.

Notation Conventions for VBScript/Visual Basic

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

<table>
<thead>
<tr>
<th>Prefix</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OAd</td>
<td>Prefix for objects that the ClearQuest API can access through its COM library. For example: OAdEntity</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> The Session and AdminSession objects do not use the OAd prefix. (See Syntax for manually creating the Session object (or the AdminSession object) in an external application)</td>
</tr>
<tr>
<td>AD</td>
<td>Prefix for VBScript Enumerated Constants. For example: AD_ORACLE</td>
</tr>
</tbody>
</table>
Understanding ClearQuest API objects

In ClearQuest, the object you work with the most is the Entity object. The Entity object represents a single data record. However, you must provide ClearQuest with verification that you are authorized to access the database before you can work with any data records.

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

The Session object acts as the primary root object to the remaining database objects. (To learn about the other root object, the AdminSession object, see Using the AdminSession object.) 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)

You must acquire an Entity object before you can view or change the data in a record.

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

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

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

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

The ClearQuest schema repository is the “master” database that contains your schemas. In addition to the objects in the preceding diagram, ClearQuest also defines a set of objects for accessing the schema repository (see Accessing the schema repository).
collection objects (see *Understanding the schema repository collection objects*), and additional objects (see *Understanding additional database objects*).

**Overview Table of the API objects**

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

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

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

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

<table>
<thead>
<tr>
<th>Schema Repository Objects</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AdminSession object</td>
<td>Access the schema repository</td>
</tr>
<tr>
<td>Database object</td>
<td>Represents the user database; use to get explicitly subscribed users</td>
</tr>
<tr>
<td>Schema object</td>
<td>Lists the schema revisions in the schema repository</td>
</tr>
<tr>
<td>SchemaRev object</td>
<td>Represents the schema revisions in the schema repository</td>
</tr>
<tr>
<td>User object</td>
<td>Represents a user of a user database</td>
</tr>
</tbody>
</table>
Schema repository collection objects are convenient for accessing with a single API multiple schema repository objects. For more information, see Understanding the schema repository collection objects.

<table>
<thead>
<tr>
<th>Schema Repository Collection Objects</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Databases collection object</td>
<td>Collection of user databases</td>
</tr>
<tr>
<td>EntityDefs collection object</td>
<td>Collection of EntityDef (record type) objects</td>
</tr>
<tr>
<td>Groups collection object</td>
<td>Collection of user database groups</td>
</tr>
<tr>
<td>Schemas collection object</td>
<td>Collection of schemas in the schema repository</td>
</tr>
<tr>
<td>SchemaRevs collection object</td>
<td>Collection of schema revisions objects in the schema repository</td>
</tr>
<tr>
<td>Users collection object</td>
<td>Collection of user database users</td>
</tr>
</tbody>
</table>
The additional objects include API for working with attachments, database descriptions, hooks, record history, charts, and reports. The additional objects include two collection objects.

<table>
<thead>
<tr>
<th>Additional Objects</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AttachmentField object</td>
<td>Represents a single attachment field in a record</td>
</tr>
<tr>
<td>AttachmentsFields collection object</td>
<td>Represents the attachment fields in a record</td>
</tr>
<tr>
<td>Attachment object</td>
<td>Stores an attachment file and information about it</td>
</tr>
<tr>
<td>Attachments collection object</td>
<td>Represents a set of attachments in one attachment field of a record</td>
</tr>
<tr>
<td>DatabaseDescription object</td>
<td>Provides information about a given database, including whether it is a schema repository or a user database</td>
</tr>
<tr>
<td>EventObject object</td>
<td>Provides read-only information about a record’s named hook</td>
</tr>
<tr>
<td>FieldInfo object</td>
<td>Provides read-only information about a field in a user database record (for example, what value it currently stores), but you typically use the Entity object instead</td>
</tr>
<tr>
<td>HistoryField object</td>
<td>Represents a single history field in a record</td>
</tr>
<tr>
<td>HistoryFields collection object</td>
<td>Contains all history-related objects</td>
</tr>
<tr>
<td>History object</td>
<td>Provides a string that describes the modifications a record has undergone</td>
</tr>
<tr>
<td>Histories collection object</td>
<td>Contains History objects</td>
</tr>
<tr>
<td>HookChoices object</td>
<td>Lists choices in a CHOICE-LIST hook</td>
</tr>
<tr>
<td>Link object</td>
<td>Connects an original record (parent) with the duplicate (child) record.</td>
</tr>
<tr>
<td>OleMailMsg object</td>
<td>Supports an email notification hook</td>
</tr>
<tr>
<td>CHARTMGR object</td>
<td>Provides an interface for creating charts</td>
</tr>
<tr>
<td>ReportMgr object</td>
<td>Provides an interface for generating reports</td>
</tr>
<tr>
<td>WORKSPACE object</td>
<td>Provides an interface for manipulating saved queries, reports, and charts</td>
</tr>
</tbody>
</table>
Working with sessions

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

Getting a Session Object

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

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

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

<table>
<thead>
<tr>
<th>Scripting Language</th>
<th>Syntax for making a call to an Entity object in a hook</th>
</tr>
</thead>
<tbody>
<tr>
<td>VBScript</td>
<td>set currentSession = GetSession</td>
</tr>
<tr>
<td></td>
<td>VBScript hooks implicitly associate the Entity object with the current record.</td>
</tr>
<tr>
<td>Perl</td>
<td>ClearQuest provides two variables that you can use in the context of a Perl hook: $entity and $session. In this context, you have two options: either (1) make an explicit call to $entity-&gt;GetSession() such as $currentSession=$entity-&gt;GetSession(); or (2) take advantage of the convenient $session variable that implicitly associates itself with the current session.</td>
</tr>
</tbody>
</table>
For external applications, you must create a Session object manually. If you want to use the adminSession object, the same rule applies.

<table>
<thead>
<tr>
<th>Language Example</th>
<th>Syntax for manually creating the Session object (or the AdminSession object) in an external application</th>
</tr>
</thead>
<tbody>
<tr>
<td>Visual Basic</td>
<td><code>set currentSession = CreateObject(&quot;CLEARQUEST.SESSION&quot;)</code></td>
</tr>
</tbody>
</table>
|                  | `set adminSession = CreateObject _  
|                  | ("ClearQuest.AdminSession")` |
| Perl             | `$currentSession = CQSession::Build();` |
|                  | `$currentSession = CQAdminSession::Build();` |
|                  | When you are done with the object, destroy it: |
|                  | `CQAdminSession::Unbuild($currentSession);` |
|                  | `CQAdminSession::Unbuild($currentAdminSession);` |

### Logging on to a database

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

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

1. Get a list of the databases associated with a schema repository by calling the GetAccessibleDatabases method of the Session object.
   
   This method returns a collection of DatabaseDescription objects, each of which contains information about a single user database.

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

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

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

### Using session-wide variables

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

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

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

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

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

**VB**

```vbnet
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")
```
Perl

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

myValue = "Hello World";

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

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

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

Ending a session (for external applications)

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

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

Your external application should end a session properly:

- Delete any objects that you explicitly created and do not need any more.

**Note:** The session ends when the user logs out or the final reference to the Session object ceases to exist.

Working with multiple sessions

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

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

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

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

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

Creating queries

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

To create a query, follow these steps:

1. Create a QueryDef object and fill it with the search parameters.
   To create this object, you can use either the BuildQuery method or the BuildSQLQuery method of the Session object.
   
   **Note:** We recommend that use the BuildQuery method. The BuildSQLQuery method generates a ResultSet object directly from an SQL query string.

2. Use the methods of QueryDef to add search criteria and to specify the fields of each record you want the query to return.

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

4 Run the query by calling the ResultSet object’s Execute method.

5 Access the data using other methods of this object. (See Moving through the result set.)

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

Defining your search criteria

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

For examples of building a query with the API, see Building queries for defects and users in the chapter entitled Examples of hooks and scripts.

Using query filters

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

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

Rather than returning the entire record, ClearQuest returns only those fields of the record that you specified by calling the BuildField method of the QueryDef object (see Creating queries). 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.
Moving through the result set

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

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

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

Retrieving the values from the fields of the record

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

<table>
<thead>
<tr>
<th>If you created your query using ...</th>
<th>The order of the columns corresponds to ...</th>
</tr>
</thead>
<tbody>
<tr>
<td>the QueryDef object</td>
<td>the order in which you added fields using the BuildField method.</td>
</tr>
<tr>
<td>a SQL statement</td>
<td>the SQL statement</td>
</tr>
<tr>
<td></td>
<td>(To discover which column has the data you want, use the ResultSet object: GetNumberOfColumns method, the GetColumnType method, and the GetColumnLabel method.)</td>
</tr>
</tbody>
</table>

Using the ClearQuest API
Working with records

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

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

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

Getting entity objects

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

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

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

<table>
<thead>
<tr>
<th>Task</th>
<th>Entity object method to call</th>
</tr>
</thead>
<tbody>
<tr>
<td>Examine or modify the values of a field</td>
<td>GetFieldValue method, SetFieldValue method</td>
</tr>
<tr>
<td>Validate and commit the record</td>
<td>Validate method, Commit method</td>
</tr>
<tr>
<td>Determine which fields must be filled in by the user</td>
<td>GetFieldRequiredness method</td>
</tr>
</tbody>
</table>
Creating a new record

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

Editing an existing record

To edit an existing record, follow these steps:

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

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

2. Call the EditEntity method of the Session object.

   Only one user at a time can edit a record. If you are creating a new record, you have permission to modify the contents of the record. However, if you are using the EditEntity method to modify an existing record while someone else is modifying it, the record is locked. If another user has a prior lock on the record, you can modify the record, but you cannot commit the record to the database with your changes.

<table>
<thead>
<tr>
<th>Task</th>
<th>Entity object method to call</th>
</tr>
</thead>
<tbody>
<tr>
<td>Determine the acceptable values for each field, and which fields have invalid values</td>
<td>GetFieldType method, GetInvalidFieldValues method</td>
</tr>
<tr>
<td>Determine which fields have been updated</td>
<td>GetFieldsUpdatedThisAction method, GetFieldsUpdatedThisGroup method, GetFieldsUpdatedThisSetValue method</td>
</tr>
<tr>
<td>Find other data records that are considered duplicates of this one</td>
<td>GetDuplicates method</td>
</tr>
<tr>
<td>Find the original data record, if this one is a duplicate</td>
<td>GetFieldOriginalValue method</td>
</tr>
</tbody>
</table>
Using the methods of the Entity object, you can perform these tasks:

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

**Saving your changes**

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

1. Validate that data in the record by calling the Validate method of the Entity object. This method returns any validation errors so that you can fix them before you attempt to save your changes.

2. Call the Commit method of the Entity object. This method writes the changes to the database, ends the current action, and checks in the record so that it cannot be edited.

**Reverting your changes**

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

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

**Viewing the contents of a record**

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

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

1. Use the GetEntity method to acquire the record.

2. Use methods of the returned Entity object to access the record’s fields.
To get a list of record types by name, use the following methods of the Session object.

<table>
<thead>
<tr>
<th>To list the names of ...</th>
<th>call this Session object method</th>
</tr>
</thead>
<tbody>
<tr>
<td>All record types</td>
<td>GetEntityDefNames method</td>
</tr>
<tr>
<td>Record types that have states</td>
<td>GetReqEntityDefNames method</td>
</tr>
<tr>
<td>Record types that are stateless</td>
<td>GetAuxEntityDefNames method</td>
</tr>
<tr>
<td>Record types that belong to a record type family</td>
<td>GetQueryEntityDefNames method</td>
</tr>
<tr>
<td>Record types you can use to create a new record</td>
<td>GetSubmitEntityDefNames method</td>
</tr>
</tbody>
</table>

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

**Ensuring that record data is current**

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

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

**Viewing the metadata of a record**

To learn how to access metadata (information about a record and its fields), see Accessing the schema repository.
Understanding user database objects

The user database objects are the following:

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

Session object

See Working with sessions.
Entity object

An Entity object represents a record in the database.

Remarks:

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

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

Accessing the fields of a record

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

To acquire a FieldInfo object, call the GetFieldValue method.

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

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

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

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

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

Although you would normally use a FieldInfo object to access a field, there are situations where you must use methods of Entity.
To set the value of a field, call the SetFieldValue method.

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

**Note:** Although you can get the behavior of a field using either an Entity object or FieldInfo object, you can only use the SetFieldRequirednessForCurrentAction method of Entity to set the field’s behavior.

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

<table>
<thead>
<tr>
<th>Task</th>
<th>Entity object method to call</th>
</tr>
</thead>
<tbody>
<tr>
<td>To retrieve the list of permissible values in the field</td>
<td>GetFieldChoiceList method</td>
</tr>
<tr>
<td>To get a constant indicating whether or not you can add additional items to the choice list.</td>
<td>GetFieldChoiceType method</td>
</tr>
<tr>
<td>To add items to a choice list that can be modified</td>
<td>AddFieldValue method</td>
</tr>
<tr>
<td>To delete items from a choice list that can be modified</td>
<td>DeleteFieldValue method</td>
</tr>
</tbody>
</table>

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

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

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

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

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

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

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

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

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

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

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

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

Finding duplicate records and the original record

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

Finding duplicate objects and the original object

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

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

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

Inside a VBScript hook, ClearQuest supplies an implicit Entity object representing the current data record. If your VBScript hook calls a method of Entity without supplying a leading identifier, ClearQuest automatically uses this implicit Entity object. In addition, ClearQuest hooks define an explicit "entity" variable to use if you want to specify the object to which you are referring. The entity variable name is identical to the record type name. If you are accessing the API from outside of a hook, or if you are accessing an Entity object other than the implicit one, you must specify the other Entity object explicitly. (Also, if you are using Perl, you must always supply an explicit variable, and its name is "entity": see Getting a Session Object.)

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.

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

or

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

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

You identify Entity objects using the display name of the corresponding record type. For stateless record types, you identify individual records using the contents of the unique key field of the record type. For state-based record types, you identify records using the record’s visible ID. ClearQuest assigns each new record a visible ID string composed of the logical database name and a unique, sequential number. For example, the tenth record in the database “BUGID” can have the visible ID “BUGID00000010”.
The following VBScript example is from a hook that accesses two Entity objects: the implicit object, and a duplicate object. The duplicate object corresponds to the record whose ID is "BUGID00000031".

set sessionObj = GetSession
' Call a method of the implicit Entity object.
set fieldValue = GetFieldValue("fieldname")
' VBScript assumes the current entity implicitly.
' The fieldname must be valid or ClearQuest returns an error.
value = fieldValue.GetValue()
' Call the same method for the duplicate object, by explicitly acquiring
' the other entity, which is of the defect record type.
set otherEntity = sessionObj.GetEntity("defect", "BUGID00000031")
set fieldValue2 = otherEntity.GetFieldValue("fieldname")
value = fieldValue2.GetValue()

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

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

An EntityDef object represents one of the record type in a schema.

**Remarks:**

In a schema, a record type specifies the metadata for one kind of record. The record type metadata defines the generic structure of that record. Metadata does not include the user data itself. Record type metadata includes the number of fields, the names of the fields, which data type each field must contain, the names of permitted actions, the names of permitted states, and so on.

An EntityDef object is the runtime representation of a record type. An EntityDef object contains information ClearQuest uses to create corresponding Entity objects at runtime. EntityDef objects can be either state-based or stateless. A state-based EntityDef object contains information about the states in which a corresponding Entity object can be placed. A stateless EntityDef object does not have any state information, but does specify which field of the Entity object is used as the unique key.

You cannot create or modify EntityDef objects at runtime. To create a new EntityDef object, you must define a corresponding record type using ClearQuest Designer. You can use an EntityDef object to obtain information about the corresponding record type. For example, you can use the GetFieldDefNames method, GetActionDefNames method, and GetStateDefNames method to obtain the names of the record type’s fields, actions, and states, respectively. You can also use the GetFieldDefType method or GetActionDefType method to obtain the type of a particular field or action.

You can use methods of the current Session object to discover the available EntityDef objects.

**Note:** If you need to create a new data record, see the Session object’s BuildEntity method.
QueryDef object

A QueryDef object defines the parameters for a query, which is used to retrieve specific records from a database.

Remarks:

A QueryDef object contains a query expression and a list of display fields. The query expression defines the search parameters for the query and can contain a complex set of conditional statements. To run the query, you must create a ResultSet Object and call its Execute method. (You can use the Session object’s BuildResultSet method to create the ResultSet object.) The ResultSet object uses the list of display fields in the QueryDef object to summarize the search results.

To create a QueryDef object,

1 Call the Session object’s BuildQuery method. The BuildQuery methods returns an QueryDef object with display fields and filters undefined.

2 Add the filters and fields for your query to the QueryDef object.

To create a query that returns all of the records in the database, you create the simplest QueryDef object by to the query one field that calls the QueryDef object’s BuildField method.

You can add filters and nodes to a QueryDef object to create more complex queries. The nodes of a QueryDef object consist of one or more QueryFilterNode objects, each containing one or more filters. Nodes group together each of their filters under a single boolean operator. You use the QueryDef object’s BuildFilterOperator method to create the root node in this tree. After that, you use the methods of QueryFilterNode to define the remaining nodes and filters. The filters themselves can use other comparison operators to test the relationship of a field to the specified data.

Note: You can also construct a query from a raw SQL query string using the Session object’s BuildSQLQuery method. However, this technique does not create a QueryDef object.
**ResultSet Object**

You can use a ResultSet object to execute a query and browse the query results.

**Remarks:**

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.
**QueryFilterNode object**

A QueryFilterNode object represents one node in the query-expression tree.

**Remarks:**

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 enumerated 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 enumerated 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.
Accessing the schema repository

Normally, you modify the schema repository (master database) using the ClearQuest Designer. However, it is possible to get information from, and make limited changes to, the schema repository using the ClearQuest API. Performing user administration, for example, is among such tasks.

Objects in the schema repository

ClearQuest defines a set of objects for the schema repository:

<table>
<thead>
<tr>
<th>Schema Repository Object</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AdminSession object</td>
<td>You use the AdminSession object to access the schema repository. (This is analogous to using the Session object to access a user database.)</td>
</tr>
<tr>
<td>Database object</td>
<td>The database for user data, such as defects.</td>
</tr>
<tr>
<td>Schema object</td>
<td>Each schema in the schema repository is represented by a Schema object. You cannot modify schemas programmatically. Use the ClearQuest Designer to make changes to a schema. The Schema object provides you with a list of schema revisions that you can use to upgrade a database.</td>
</tr>
<tr>
<td>SchemaRev object</td>
<td>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.</td>
</tr>
<tr>
<td>Group object</td>
<td>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.</td>
</tr>
<tr>
<td>User object</td>
<td>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.</td>
</tr>
<tr>
<td>collection objects</td>
<td>See Understanding the schema repository collection objects</td>
</tr>
</tbody>
</table>
Using the AdminSession object

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.

The schema repository also defines a set of collection objects for containing database, group, user, schema, schema revisions, and record types:

- Databases collection object
- Groups collection object
- Users collection object
- Schemas collection object
- SchemaRevs collection object
- EntityDefs collection object

Logging on to the schema repository

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

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

Most of the schema repository information can be found in the properties of various objects. For example, the AdminSession object has properties that return a complete list of the databases, schemas, users, and groups associated with the schema repository. The AdminSession object also has methods that retrieve database, user, and group objects whose name you already know. You can also use methods of the AdminSession object to create new databases, user accounts, and groups.

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

**Updating user database information**

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

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

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

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

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

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

To get a User object for an existing user, call the GetUser method of the AdminSession object, or iterate through the objects in the Users property.

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

To get an existing group, call the GetGroup method, or iterate through the Groups property.

To add a user to a group, call the AddUser method of the Group object.

3 Use the group to add or remove user accounts.

Note: You cannot remove User or Group objects from the schema repository. Once you create these objects, they remain permanently.
**Common API calls to get user information**

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

<table>
<thead>
<tr>
<th>User Administration Task</th>
<th>Session object method to call</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get the name of the current user</td>
<td>GetUserLoginName method</td>
</tr>
<tr>
<td>Get a list of groups to which the user belongs.</td>
<td>GetUserGroups method</td>
</tr>
<tr>
<td>Get a user’s email address</td>
<td>GetUserEmail method</td>
</tr>
<tr>
<td>Get a user’s full name</td>
<td>GetUserFullName method</td>
</tr>
<tr>
<td>Get a user’s phone number</td>
<td>GetUserPhone method</td>
</tr>
<tr>
<td>Get any additional information about the user</td>
<td>GetUserMiscInfo method</td>
</tr>
</tbody>
</table>
Understanding schema repository objects

The SchemaRepository objects allow you to get (and, in some cases, set) data in the schema repository. The Schema Repository objects are:

- AdminSession object
- Database object
- Schema object
- SchemaRev object
- User object
- Group object
AdminSession object

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

Remarks:

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:

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

After you create the AdminSession object, you must log on to the schema repository using the Logon method of the AdminSession object. To log on to the database, you must know the administrator's login name and password, as well as the name of the record containing the schema repository. When you are logged on, you can use the methods of the AdminSession object to get information from the schema repository.
**Database object**

A Database object stores information about a user database.

**Remarks:**

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

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

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

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

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

1. Create the database by calling the CreateDatabase method of the current AdminSession object.

2. Set the initial schema revision by using the SetInitialSchemaRev method.

**Note:** As new schema revisions become available, update the database by using the Upgrade method.
The following example shows you how to create a database and set its initial schema revision.

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

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

db.ApplyPropertyChanges
```
**Schema object**

A Schema object contains information about a particular schema.

**Remarks:**

A Schema object represents a single schema in a master database. Use Schema objects to refer to schemas and to get a list of the revisions of the schema that are available.

**Note:** The API does not allow you to create new schemas or modify existing schemas. Schemas must be created or modified by using ClearQuest Designer. You can get a list of schemas defined in the schema repository (master database) by accessing the Schemas property of the AdminSession object.
**SchemaRev object**

A SchemaRev object contains information about a single schema revision, including information about its packages.

**Remarks:**

Schema revisions identify a particular version of a schema. You use schema revisions when creating and updating databases.

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

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

To discover which packages and package revisions apply to the current user database, use the GetEnabledPackageRevs method and the GetEnabledEntityDefs method.
**User object**

A User object contains information about a single user account.

**Remarks:**

The information in a User object corresponds to the information on a user properties page in ClearQuest Designer. To view a user properties page, use the user administration tools in ClearQuest Designer to select the user and edit that user's information.

Using the User object, you can get or set a user's personal information, including the user's name, email address, phone number, and access privileges. You can also use the methods of User to change the databases to which the user is subscribed.

Changes you make to user accounts are reflected in the schema repository (master database) as soon as you call UpgradeMasterUserInfo, but not in the associated user databases.

To update the user databases, do one of the following:

- **Iterate through the databases in the AdminSession object’s Databases property and upgrade each database individually by calling its UpgradeMasterUserInfo method.**

- **Use the user administration tools in ClearQuest Designer.**
Group object

A Group object contains information about a single group of users.

Remarks:

Groups allow you to administer users as one or more groups, which is more convenient than administering each user separately. Use the Group object to get or modify the properties of a group, including the group’s name and the databases to which it is subscribed. You can also add users to the group.

Changes you make to groups are immediately reflected in the schema repository (master database) but not the associated user databases. To update the user databases, use the user administration tools in ClearQuest Designer.
Understanding the schema repository collection objects

A collection is a container for 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:

- Databases collection object
- EntityDefs collection object
- Schemas collection object
- SchemaRevs collection object
Databases collection object

A Databases object is a collection object for Database objects.

Remarks:

You can get the number of items in the collection by accessing the value in the “Count property” on page 430. Use the “Item method” on page 431 to retrieve items from the collection.

EntityDefs collection object

The EntityDefs object (EntityDefs) is a collection object that contains a collection of EntityDef objects.

Groups collection object

A Groups object is a collection object for Group objects.

Remarks:

You can get the number of items in the collection by accessing the value in the “Count property” on page 438. Use the “Item method” on page 439 to retrieve items from the collection.

Schemas collection object

A Schemas object is a collection object for Schema objects.

Remarks:

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

SchemaRevs collection object

A SchemaRevs object is a collection object for SchemaRev objects.

Remarks:

You can get the number of items in the collection by accessing the value in the “Count property” on page 446. Use the “Item method” on page 447 to retrieve items from the collection.
**Users collection object**

A Users object is a collection object for User objects.

*Remarks:*

You can get the number of items in the collection by accessing the value in the Count property. Use the Item Method to retrieve items from the collection.
Understanding additional database objects

The additional Database objects are:

- AttachmentField object
- AttachmentsFields collection object
- Attachment object
- Attachments collection object
- DatabaseDescription object
- EventObject object
- FieldInfo object
- HistoryField object
- HistoryFields collection object
- History object
- Histories collection object
- HookChoices object
- Link object
- OleMailMsg object
- CHARTMGR object
- ReportMgr object
- WORKSPACE object
**AttachmentField object**

An AttachmentField object represents one attachment field in a record.

**Remarks:**

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

The AttachmentField object has three properties:

- “FieldName property” on page 476, which returns the field name
- “DisplayNameHeader property” on page 474, which returns the unique keys of the attachments
- “Attachments property” on page 472, which returns an Attachments collection object

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

**AttachmentsFields collection object**

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

**Remarks:**

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

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

An Attachment object represents a single attached file.

Remarks:
The attachment is physically stored in the user database.

An Attachment object

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

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

To attach files to a database, use the Add method of the Attachments collection object. (You never create instances of Attachment directly.)

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

To delete an Attachment object, use the Delete method of the Attachments collection object.

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

Attachments collection object

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

Remarks:

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

Every AttachmentField object has exactly one Attachments object. You retrieve it by retrieving the AttachmentField object’s Attachments property.
**DatabaseDescription object**

The DatabaseDescription object provides information about a particular database.

**Remarks:**

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

When you have a DatabaseDescription object, you can

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

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

**EventObject object**

An EventObject contains information that is passed to the named hook of an Entity object.

**Remarks:**

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.
**FieldInfo object**

A FieldInfo object contains static information about one field of a user data record.

**Remarks:**

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 more details, see the methods of the Entity object.

As a convenience, Entity contains a few methods that act as wrappers for FieldInfo methods. For example, the GetFieldType method of Entity is equivalent to the GetType method of FieldInfo. However, Entity also has some methods that have no FieldInfo counterparts, such as the GetFieldOriginalValue method and the GetFieldChoiceList method.
HistoryField object

A HistoryField object represents one history field in a record.

Remarks:
The HistoryField object represents the history field in the record. (In the current version of ClearQuest, there can be only one history field per record.) A HistoryFields collection object represents the set of all the record’s history fields.

The HistoryField object has one property: the Histories property. This property contains the set of History objects that describe the changes to the record.

HistoryFields collection object

The HistoryFields object represents all of the history fields in a record. (In the current version of ClearQuest, there can be only one history field per record, so the HistoryFields collection object always contains only one item.)

Remarks:
HistoryFields is a collection object similar to the standard Visual Basic collection objects. It is a container for a set of HistoryField objects. The HistoryFields object’s property and methods tell you how many items are in the collection and let you retrieve individual items. You cannot add, remove, or modify the items.

Every Entity object has exactly one HistoryFields object. You cannot create a new HistoryFields object. However, you can retrieve the pre-existing HistoryFields object from a given Entity object by invoking Entity’s HistoryFields property.

History object

A History object contains information about a modification to a record.

Remarks:
The History object encapsulates the String that is displayed for one entry in a history field of a data record. The History object has only one property: the Value property.
Histories collection object

The Histories object represents the set of history entries in one history field of a record.

Remarks:

Histories is a collection object that is a container for a set of History objects. The Histories object’s property and methods tell you how many items are in the collection and let you retrieve those items.

Every HistoryField object has exactly one Histories object. You can retrieve it by invoking HistoryField’s Histories property.
HookChoices object

A HookChoices object represents the list of choices presented by a CHOICE_LIST hook.

Remarks:

The HookChoices object is a special object that is invisible except inside a CHOICE_LIST hook. This object has only one method, the AddItem method, which you can use to add new items to the list.

The HookChoices object is stored in a variable called choices and you can only access it by that name.

Note: For Perl, use a Perl array to return a choice list. See the Hook Choices Code Example.
**Link object**

A Link object connects two Entity objects.

**Remarks:**

Links are the edges in the tree of duplicates. Links point both to the original record (the "parent") and to the duplicate record (the "child"). Both records must be state-based (as opposed to stateless). However, the parent and child do not need not be based on the same record type.

The methods of link allow you to retrieve:

- the parent and child record objects that are linked together.
- the ID strings for the parent and child.
- the EntityDef that is the template for the parent or child.
- the names of these EntityDefs

To create a Link object, use the MarkEntityAsDuplicate method of the Entity object that is to become the duplicate. To delete the object, use the UnmarkEntityAsDuplicate method.
OleMailMsg object

An OleMailMsg object represents an e-mail message that you can send to your users.

Remarks:

The main purpose for the OleMailMsg object is to send e-mail messages from an action notification hook. You can use the methods of this object to specify the contents of the e-mail message including the recipients, sender, subject, and body text. You can then use the Deliver method of this object to send the e-mail message.

This object does not support Perl. To create a new OleMailMsg object, you must use the VBScript CreateObject method as follows:

```vbnet
Dim mailmsg Set mailmsg = CreateObject("PAINET.MAILMSG")
```

When you have an OleMailMsg object, you can
- add recipients using the AddTo, AddCc, and AddBcc methods
- set the return address using the SetFrom method
- add a subject line using the SetSubject method
- set the body text of the e-mail message using the SetBody and MoreBody methods
**CHARTMGR object**

The CHARTMGR object provides an interface for creating charts.

**Remarks:**

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

1. Verify that the WORKSPACE object is associated with a Session object.

2. Call the GetChartMgr method of the WORKSPACE object.

3. Execute a query by calling the ResultSet object’s Execute method.

4. Specify the data to use for the chart by calling the SetResultSet method and specifying a ResultSet object containing the data your query generated.

5. Specify the chart to use in creating the image and generate the image.

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

The ReportMgr object provides an interface for generating reports.

Remarks:

You can use this object to write external applications to execute reports defined in the ClearQuest workspace. You can also use the methods of this object to check the status and parameters of a report.

1 Associate the WORKSPACE object with a Session object.
   This association makes it possible to access reports in the ClearQuest workspace.

2 Get a ReportMgr object by calling the GetReportMgr method of the WORKSPACE object.
   When you call GetReportMgr, you must specify the name of the report you want to execute. ClearQuest associates that report with the returned ReportMgr object. To execute a different report, you must create a new ReportMgr object.

3 Set the name of the file in which to put the report data by calling the SetHTMLFileName method.

4 Execute the report by calling the ExecuteReport method.
**WORKSPACE object**

The WORKSPACE object provides an interface for manipulating saved queries, reports, and charts in the ClearQuest workspace.

**Remarks:**

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:

```vbnet
set wkspcObj = CreateObject("CLEARQUEST.WORKSPACE")
```

Your Perl code uses this syntax:

```
$wkspcObj = new CQWorkspaceMgr
```

Before you can use a WORKSPACE object created using CreateObject, you must assign a Session object to it. To assign a Session object, you must call the SetSession method of the WORKSPACE object.

You use the methods of the WORKSPACE object to get information about the contents of the ClearQuest workspace. You can get a list of the queries, charts, or reports in the workspace. You can also separate items based on whether they are in the Public Queries folder or in a user’s Personal Queries folder. You can also use this object to save queries back to the workspace.

**Pathnames in the Workspace**

The workspace organizes items into a hierarchical structure that you navigate as a series of nested folders. This hierarchy resembles the Windows Explorer in that you can expand or collapse folders to reveal the layered contents.
You identify individual queries, charts, and reports using the pathname information for that item. The pathname for an item is composed of the folder names enclosing it. Folder names are separated using a forward slash (/) character. For example, the pathname of a query called All Defects and located in the Public Queries folder would have the pathname `Public Queries/All Defects`.

ClearQuest does not provide an explicit way to create new folders. However, you can create nested folders implicitly when you save a query. The `SaveQueryDef` method lets you specify pathname information for a query. If the folders in the pathname do not exist, ClearQuest creates them (unless they are top-level folder). ClearQuest does not allow you to create top-level folders; all elements must be nested inside either the Public Queries or Personal Queries folders.
**Glossary**

**access control**
Access control limits the use or modification of actions to designated users. Access for actions is set through the action’s Properties dialog and can be open to all users, limited to a specific group, or controlled by a hook. Access to fields is determined by the Behaviors table.

**action**
Whenever you modify a record, you invoke an action from the Action menu to register the changes. Actions may result in a state transition or they may simply modify information in the record’s fields. In ClearQuest, the Action menu displays only the appropriate legal actions.

ClearQuest allows you to modify a record or to transition a record from one state to another state. For example, you can transition a record from the open state to the closed state.

In ClearQuest Designer: the ClearQuest administrator modifies the Actions table and state transition matrix of each record type to define the legal actions for records of that type. The definition of each action is stored in the ClearQuest schema repository.

**administrator**
The person responsible for setting up schemas and databases at your company.

The ClearQuest administrator uses ClearQuest Designer to create and modify schemas, databases, and forms. In addition, the administrator performs other tasks, such as creating user groups and establishing permissions, maintaining the database and setting up e-mail notification.

**aging chart**
Aging charts show how many records have been in the selected states for how long. Use aging charts to answer the questions: “How many defects have been open for one week? For two weeks? For three weeks?”

**API**
Application Programming Interface.

ClearQuest contains a robust interface that administrators can use to customize the behavior of their databases. The API consists of a set of objects, methods, and functions...
that can be called from hook code to perform tasks such as getting or setting the value of a field.

**attachment**

ClearQuest allows you to associate a file with a particular record. Attachments are stored in the ClearQuest user database, along with other data contained in the record.

**attachment field**

An attachment field is a field whose type is ATTACHMENT_LIST. An attachment field stores attached files.

**Attachment object**

In the ClearQuest Designer API: An Attachment object stores information about a single attached file.

**Attachments object**

In the ClearQuest Designer API: An Attachments object is a collection for Attachment objects. The overall collection contains the attached files for a single field (represented by an AttachmentField object).

**AttachmentField object**

In the ClearQuest Designer API: An AttachmentField object represents the attached files for a single field. This object stores a reference to an Attachments object.

**AttachmentFields object**

In the ClearQuest Designer API: An AttachmentFields object is a collection object that represents all of the attached files for a given record. This object stores references to one AttachmentField object for every attachment field in the record.

**bar chart**

A bar chart illustrates comparisons among individual items. Categories are organized horizontally, values vertically, to focus on comparing values. For example, the bar chart “Defects by Engineer” displays the names of the engineers along the horizontal or x axis and the type of defect along the vertical or y axis.
**behavior**

The behavior of a field defines the access restrictions for the field. The behavior for a given field can be READONLY, OPTIONAL, MANDATORY, or USE_HOOK. To set the behavior for a field, modify the field’s entry in the Behaviors table.

**change-state action**

A change-state action moves a record from one state to another state.

**chart**

A chart is a graphical representation of a selected set of records, created usually for the purpose of comparing attributes of those records. There are several different kinds of charts including distribution charts, trend charts, and aging charts. Results can be displayed using several different kinds of graphics, including bar chart and pie chart graphics.

**checkout/checkin**

The two-part process that allows you to edit a schema and add a new version of the schema to the ClearQuest schema repository. You can modify a schema only after you check it out of the schema repository. A version of the schema can be associated with a database only after you check it in to the schema repository.

A checkout allows you to add fields, record types, forms, states and actions to a schema.

A checkin adds a new version of the schema to the ClearQuest schema repository. Once you check a schema into the schema repository, you can make changes to it only by creating a new version of the schema. You can save intermediate changes to a schema, without checking it into the schema repository and without updating the version of the schema.

**control**

A graphic element such as a text box, list box, button or picture that you place on a form to display data, enter data, perform an action, or make the form easier to read.

**cursor**

The cursor is a placeholder used while navigating through a result set. The cursor indicates which row is currently being reviewed.
**database**

In ClearQuest, the term database refers to the client database that contains all user data and a copy of the schema associated with the database. The ClearQuest database contains all forms, fields, the state transition matrix, and all data entered by users. Compare with production database and test database.

**DatabaseDescription object**

In the ClearQuest Designer API: A DatabaseDescription object contains information about a particular database. You can use this object to get information about the database.

**database set**

A database set consists of a schema repository and all of the databases associated with that repository. The databases in the set can be either production databases or test databases.

**dependent field**

A dependent field is one whose value is affected by the values in other fields. To set up a dependent field, you must create hooks that set the value of the field when the original field (or fields) changes. Typically, you would modify one of the following hooks: the field default value hook, the field value-changed hook, or the field choice-list hook.

**Designer toolbar**

In ClearQuest Designer: The Designer toolbar provides easy access to some of the more commonly-used menu items.

**destination state**

When you perform an action that causes a state transition, the destination state is the state to which the record is sent. The record originates from the source state.

**distribution chart**

Distribution charts are used to measure how many records fall into defined categories or match the values you indicate.
For example, use a distribution chart to see the current status of a group of records, or see who has been assigned the most/least change requests. Another example is a chart that details which records have the highest priority.

**duplicate**
In ClearQuest, the term duplicate refers to the nature of a record, action or field in the state transition matrix.

A record that contains data already recorded in a previous entry is a duplicate. In the case of defect tracking, a duplicate identifies a defect that has previously been reported.

**duplicate action**
A duplicate action marks a given record as a duplicate of another record.

**entity object**
In ClearQuest Designer: An entity object is a runtime object that represents a record in the database.

**entitydef object**
In ClearQuest Designer: An entitydef object is a runtime object that represents the metadata for a record. This metadata describes the structure of the record, including the number of fields, their names, what data types they must contain, the names of the permitted actions and states for this record type, and so on.

**external application**
You can write an external application in VBScript or Perl to perform tasks against a ClearQuest database. An external application must begin by creating a session object and logging in to a ClearQuest database. Among the tasks you can then perform are: create a query, execute a saved query, create new records, perform actions on existing records.

**expression**
Any combination of an operator, value, and field name that evaluate a single value. Filters use expressions to define query criteria.
field
A field represents a singular piece of data in a record. Fields can contain simple data types such as numbers and strings or they can contain more complex information such as references to other fields, dates, or the current state of a record.

FieldInfo object
In the ClearQuest Designer API: A FieldInfo object contains information about a particular field. You can use this object to obtain the field’s value and other attributes.

filter
In ClearQuest, filters are restrictions you place on a query to limit the number of records returned. Typically, a filter specifies which field values are needed to identify the specific records you want to work with. For example, you can set up a filter that limits the query to records submitted after a certain date. Records that were submitted before the given date are not returned in the query results.

Filter dialog
In ClearQuest, use the Filter dialog to edit the criteria of a given filter.

form
A form provides a visual interface for entering data into a new record, for modifying the data in an existing record, or for specifying query information. You can create record forms or submit forms for your schema.

Form Layout toolbar
In ClearQuest Designer: The Form Layout toolbar allows you to adjust the alignment and size of controls in a form.

hook
Hooks are entry points, like triggers, for pieces of code that ClearQuest executes at specified times to more fully customize the product. Actions can have hooks for access control, initialization, notification, committal, and validation. Fields can have hooks for specifying default values, choice lists, and permissions and for handling tasks associated with the field when it is validated or its value changes. Records can use record scripts that allow you to trigger actions that are specific to a record type. Global
scripts allow you write a subroutine, such as an e-mail notification, that can be called from any hook in any record type.

**history**

ClearQuest allows you to track all modifications of each record. The history of a record includes the creation date and each modification made to the record, such as assigning a defect to an engineer, adding details to the description field and resolving a defect.

**history field**

In ClearQuest Designer: A History field stores information about the actions that have taken place on a record. Every record has an implicit history field associated with it. You cannot create new history fields. You can place a history control on a form to allow the user to view the history of a record.

**History object**

In the ClearQuest Designer API: A History object stores a text string describing an action that was initiated on a record.

**Histories object**

In the ClearQuest Designer API: A Histories object is a collection that stores the History objects associated with a single history field.

**HistoryField object**

In the ClearQuest Designer API: A HistoryField object represents the history entries displayed in a single history field.

**HistoryFields object**

In the ClearQuest Designer API: A HistoryFields object is a collection that stores all of the history information for a given record. This object stores references to one HistoryField object for every history field used on the record type’s form.

**HookChoices object**

In the ClearQuest Designer API: A HookChoices object represents the choices stored for a given field.
**import action**
An import action is used when records are imported from another database. During an import action, the new records are added to the database with only a limited amount of validation. In particular, records are not validated to determine whether or not they could have legally reached their current state.

**initialization hook**
In ClearQuest Designer: An initialization hook can be associated with an action to initialize the fields of a record to some default values. Because this hook provides access to all the fields of the record, you should use it primarily for complex initialization. Compare with the default value hook for fields.

**line chart**
A line chart shows trends in data at equal intervals. Generally time elements are displayed along one axis and values displayed along the alternate axis.

**Link object**
In the ClearQuest Designer API: A Link object represents a link between a duplicate and its original record. You cannot create Link objects directly.

**metadata**
Metadata is information that describes other information. In ClearQuest, metadata is used to specify the structure of records. Databases use metadata to perform searches.

**modify action**
A modify action allows users to modify the fields of a record without changing the record’s state.

**notification hook**
In ClearQuest Designer: A notification hook can be associated with an action to send notifications or to trigger other actions. For example, a notification hook can send an e-mail message to a group of people to alert them to changes in a particular record.

**Operator**
Operators act on field values to create a filter expression. Valid operators are:
An original record is a record that has one or more duplicate records associated with it. ClearQuest updates duplicate records using information in the original record.

**Note:** An original object can itself be a duplicate of another object.

**parent**

A parent record is the original record among two or more duplicate records. All other records are children (or duplicates) of the parent and should draw their state information from the parent.
**permission**

Users must be granted permission to access a database or to access the fields of a record. The ClearQuest administrator defines the permissions for each user using ClearQuest Designer.

**pie chart**

A pie chart shows the relationship of items to the sum of the items, or as a percentage of the whole. It always displays only one data series and is useful when you want to emphasize a significant element.

**poll interval**

The poll interval for a database is the amount of time a database waits before checking to see if a user's connection is still valid.

**production database**

The production database is the database used by users to submit defects, run queries, and modify records. The data in this database is used by the company to track defects from the time they are found to the time they are fixed.

**property**

In VBScript, a property is a data member of an object. Properties contain readable (and occasionally writable) values associated with the object.

**query**

A query is a request to the system to return a set of records that match the specified search criteria. Queries use filters to set up the search criteria.

**QueryDef object**

In the ClearQuest Designer API: A QueryDef object contains the information for a query, including the fields to display and the search criteria. You must use this object in conjunction with a ResultSet object to initiate a query.

**QueryFilterNode object**

In the ClearQuest Designer API: A QueryFilterNode object contains information about the search criteria in a query. This object represents a single condition in the search
criteria. Multiple QueryFilterNode objects can be created and grouped to perform complex searches.

**record form**

A record form is a form that can be used to display the contents of a record or to submit new records. Every record type must have at least one record form, which ClearQuest displays by default when you query the ClearQuest database. If a record type also has a submit form, ClearQuest uses that form when submitting new records instead of the record form.

**record type**

A record type is a template that defines the actions, fields, forms, behaviors, and state information associated with a record. The state information associated with record types defines the rules for how a record moves from state to state in the database. Schemas can also contain stateless record types which do not move from state to state.

**record type family**

ClearQuest enables you to define a "family" of record types that have related characteristics so that one query can be defined which will return the results from one of these "families" of types. This will, among other things, enable you to run "todo" lists across multiple record types, such as defects, enhancement requests, and tasks with a single query.

**result set**

A result set contains the data returned from a database search (query). This data is organized into rows and columns where each row represents a single record and each column represents a designated field of the record.

**ResultSet object**

In the ClearQuest Designer API: A ResultSet object initiates a query and provides methods to allow you to navigate through the search results.

**schema**

In ClearQuest, the term schema refers to all the attributes associated with a database. This includes field definitions, field behaviors, the state transition table, actions, report formats, and forms.
The ClearQuest administrator creates and modifies schemas in ClearQuest Designer. ClearQuest supports multiple schemas and multiple versions of each schema. Each version of a schema can be associated with multiple databases.

ClearQuest allows you to update and delete a schema. There must always be at least one schema in the ClearQuest schema repository.

**schema repository**

The schema repository is a master database that contains all the data associated with existing schemas. No user data is stored in the schema repository.

**Session object**

In the ClearQuest Designer API: A Session object represents the context in which users access a database. The Session object provides methods to allow the creation and modification of records and queries.

**source state**

When you perform an action that causes a state transition, the source state is the state from which the record originated. The record is sent to the destination state.

**SQL**

SQL stands for Standard Query Language and is a language supported by most databases for specifying queries.

**SQL editor**

In ClearQuest, use the SQL editor to edit SQL expressions.

**state**

The state of a record refers to the record’s location in the record lifecycle. The ClearQuest administrator defines the possible states in which a record can exist. For example, a record is usually given the Submit state when it is first entered into the system. From there, it might proceed to the Open state while the defect is being examined, and then to the Fixed state when the defect has been corrected.
**state transition**
A state transition occurs when a record moves from one state to a different state. Actions trigger state transitions based on the rules set up in your system’s state transition matrix.

**state transition matrix**
The state transition matrix defines the rules for moving from one state to another state. For each state, the administrator decides the appropriate set of state transitions for that state and enters them into the matrix.

**submit action**
A submit action allows users to create new records in the database.

**submit form**
A submit form is a specialized type of form that is used only for adding new records to the ClearQuest database. If a submit form is available, it is used instead of the default record form when adding new records. ClearQuest still uses the record form to display existing records in the database.

**test database**
A test database is a database used by the administrator to verify the correctness of the schema associated with the database. Typically, a test database contains a set of artificial records whose contents are created solely for the purpose of testing.

**trend chart**
Trend charts show how many records were transitioned into the selected states by day, week or month. In other words, they show you the rate at which new records are being submitted, resolved or moved into other states.

**UNC Pathname**
A Uniform Naming Convention pathname allows you to fully specify the location of a file. A UNC Pathname includes the host machine and directory information and is of the format:

`\machine_name\directory\file.ext`
**undo checkout**

Cancels a schema checkout. ClearQuest cancels all edits to a schema and reverts to the previously saved version of the schema when you undo a checkout. You can save intermediate changes to a schema, without checking it into the schema repository or updating the version of the schema. Once you check a schema into the schema repository, you can only make changes to it by creating a new version of the schema.

**unduplicate action**

An unduplicate action removes the mark from a record that identifies it as a duplicate of another record.

**unique key**

The database needs to know which column or combination of columns always have a unique value. For record types that contain states, the unique key is the ID. For stateless record types, the administrator must assign a unique key. For example, in a project table, the Project Name could be the unique key. In the case that there are multiple versions of the project, the Project Name and the Version can be the unique key.

**upgrade database**

The process of applying recent changes to a user database. The changes are created using ClearQuest Designer and stored in the master database.

**user**

A ClearQuest user is someone who submits records to a database using ClearQuest or who modifies existing records in a database. Users can also create their own custom forms to use when creating queries but cannot modify the public forms provided with the database. Compare with administrator.

**user group**

A user group is a list of users with similar privileges and access permissions. ClearQuest uses user groups to limit access to certain actions. When access to an action is limited to a user group, only members of that group may perform the action.
**validate**

ClearQuest stores all schemas in the schema repository. Before checking in changes to a schema ClearQuest validates all changes, verifying that field types and behavior are valid. Some of the tests ClearQuest performs during the validation process are:

- Validates that you have not used SQL reserved words incorrectly.
- Validates that you have entered unique labels and names for fields and actions.
- Validates that you have assigned a type to each field and a behavior for each state of each field.
- Validates that you have supplied a reference_to record type for each reference field.
- Validates that you have defined a source state and a destination state for all state transitions.
- Validates that you have defined a unique key for all stateless record types.

**validation hook**

A validation hook verifies that the fields in a record do not contain illegal values. Validation hooks can be associated with fields to verify the contents of the field immediately or with actions to verify the fields in an entire record.

**version**

ClearQuest allows you to modify or update a schema. Each time that you checkout a schema, ClearQuest creates a new version, or revision, of the schema. ClearQuest stores each version of the schema in the schema repository. You can associate any version of a schema to a database.

ClearQuest allows you to delete either the last version of the schema or the entire schema.

**Workspace**

The Workspace displays the currently available elements in the left pane of the ClearQuest component. Elements in the Workspace are displayed as a series of navigable folders that can be expanded and collapsed as needed.

In ClearQuest, the Workspace displays your personal and system queries, charts, and reports.
In ClearQuest Designer, the Workspace displays the elements of the currently selected schema. Schema elements include field and behavior tables, states and the state transition matrix, forms and stateless record types, such as user and project tables.
Examples of hooks and scripts

The chapter contains the following hooks and scripts to help use the ClearQuest API to meet your organization’s business needs:

- “Getting and setting attachment information” on page 96
- “Building queries for defects and users” on page 98
- “Updating duplicate records to match the parent record” on page 103
- “Managing records (entities) that are stateless and stateful” on page 105
- “Extracting data about an EntityDef (record type)” on page 109
- “Extracting data about a field in a record” on page 111
- “Notifying users of changes to an entity (record)” on page 113
- “Running a query and reporting on its result set” on page 115
- “Getting session and database information” on page 117
- “Running a query against more than one record type (multitype query)” on page 119
- “Triggering a task with the destination state” on page 121

For more sample code, see Finding examples in the chapter entitled “Using the ClearQuest API.”
Getting and setting attachment information

ClearQuest supports attachments, which enable ClearQuest users to add to a change request record one or more files (text, spreadsheets, screen shots, diagrams, and more). You can both get and set certain kinds of attachment information, such as the attachment description.

The following code fragment iterates over all the attachment fields of a record. For each of the attachment fields, this code

- prints the field names of the attachment_list type, which is a list of attached files (for more information, see GetValueAsList method).
- iterates over that attachment field’s attachments to print the file name, file size, description, and content of each attachment.

To illustrate that the attachment’s description is a read/write property, the code also

- alters the description of the attachment
- prints the new description

**Note:** The following code fragment is a hook (for example, an action initialization hook), and therefore “gets” the session object. However, you can also include this code in an external application if you manually create the session object and log on to the database.

**VBScript**

```vbnet
Dim attachFields 'This is an AttachmentFields collection object.
Dim attachField ' This is an AttachmentField object.
Dim attaches 'This is an Attachments collection object.
Dim myAttach 'This is an Attachment object.
Dim session

' for an external application in Visual Basic,
' manually create the session object as follows:
' set session = CreateObject('CLEARQUEST.SESSION')
set session = GetSession()

Set attachFields = AttachmentFields
' Iterate over the attachment fields on an Entity.
For Each attachField In attachFields
' Print to the DBWin32.exe window.
    session.OutputDebugString attachField.FieldName
```

Examples of hooks and scripts
Examples of hooks and scripts

Set attaches = attachField.Attachments
' iterate over the attachment’s field attachments
For Each myAttach In attaches
    session.OutputDebugString myAttach.FileName
    session.OutputDebugString myAttach.FileSize
    session.OutputDebugString myAttach.Description
' Alter the description and print again
    myAttach.Description = "This is a new description."
    session.OutputDebugString myAttach.Description
' Use the Load method to write the object’s contents to a file.
    myAttach.Load("c:\temp\foo")
    ' Here, put some code to print out c:\temp\foo (for example).
    Next myAttach
Next attachField

Perl

    # You can use the $session variable that ClearQuest provides.
    my $attachFields = $entity->GetAttachmentFields();
    foreach $attachField (@$attachFields) {
        $session->OutputDebugString($attachField->GetFieldName());
        my $attaches = $attachField->GetAttachments();
        foreach $attach (@$attaches) {
            $session->OutputDebugString($attach->GetFileName());
            $session->OutputDebugString($attach->GetFileSize());
            $session->OutputDebugString($attach->GetDescription());
            # Alter the description and print again
            $attach->SetDescription("This is a new description");
            $session->OutputDebugString($attach->Description());
            # Use the Load method to write the object’s contents to a file.
            $attach->Load("c:\temp\foo");
            # Here, put some code to print out c:\temp\foo (for example).
        }
    }
}
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).

Select all defects that belong to the “defect” record type:

**VBScript**

```vbscript
set session = GetSession
Set querydef = session.BuildQuery("defect")
querydef.BuildField("id")
querydef.BuildField("headline")

Set resultset = session.BuildResultSet(querydef)
```

**Perl**

```perl
# You can use the $session variable that ClearQuest provides.
my $myQuerydef = $session->BuildQuery("defect");
$myQuerydef->BuildField("id");
$myQuerydef->BuildField("headline");

my $myResultset = $session->BuildResultSet($myQuerydef);
```
Select defects that match these criteria:

- “beta2” planned release
- “assigned to” user “johndoe”

**VBScript**

```vbscript
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)
```

**Perl**

```perl
# You can use the $session variable that ClearQuest provides.
my $querydef = $session->BuildQuery("defect");
$querydef->BuildField("id");
$querydef->BuildField("headline");

my $operator =
    $querydef->BuildFilterOperator($CQPerlExt::CQ_BOOL_OP_AND);
$operator->BuildFilter("assigned-to", $CQPerlExt::CQ_OOMP_OP_EQ, "johndoe");
$operator->BuildFilter("planned_release", $CQPerlExt::CQ_OOMP_EQ, "beta2");

my $resultset = $session->BuildResultSet($querydef);
```
Select defects that match these criteria:

- assigned to a certain set of users
- planned for resolution during this release
- not yet resolved

**VBScript**

```vbscript
set session = GetSession
Set querydef = session.BuildQuery("defect")
querydef.BuildField("id")
querydef.BuildField("component")
querydef.BuildField("priority")
querydef.BuildField("assigned_to.login_name")
querydef.BuildField("headline")

Set operator = querydef.BuildFilterOperator(AD_BOOL_OP_AND)
operator.BuildFilter("planned_release", AD_COMP_OP_EQ, "beta")
operator.BuildFilter("state", AD_COMP_OP_NEQ, "resolved","verified")
operator.BuildFilter("priority", AD_COMP_OP_IN, 
"(1,2)")

Set suboperator = operator.BuildFilterOperator(AD_BOOL_OP_OR)
suboperator.BuildFilter("assigned_to",AD_COMP_OP_IN, _
"'lihong','gonzales','nougareau','makamoto'")

Set resultset = session.BuildResultSet(querydef)
```
Perl

my $querydef = $session->BuildQuery("defect");
$querydef->BuildField("id");
$querydef->BuildField("component");
$querydef->BuildField("priority");
$querydef->BuildField("assigned_to.login_name");
$querydef->BuildField("headline");

my $operator = $querydef->BuildFilterOperator(CQPerlExt::CQ_BOOL_OP_AND);
$operator->BuildFilter("planned_release", CQPerlExt::CQ_COMP_OP_EQ, "beta");
$operator->BuildFilter("state", CQPerlExt::CQ_COMP_OP_NEQ, "'resolved','verified'");
$operator->BuildFilter("priority", CQPerlExt::CQ_COMP_OP_IN, "(1,2)");
my $suboperator =
$operator->BuildFilterOperator(CQPerlExt::CQ_BOOL_OP_OR);
$suboperator->BuildFilter("assigned_to", CQPerlExt::CQ_COMP_OP_IN, "'lihong','gonzales','nougareau','makamoto'";

my $resultset = $session->BuildResultSet($querydef);
Find the users in a certain group (software engineering, sw_eng):

**VBScript**

```vbnet
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)
```

**Perl**

```perl
my $querydef = $session->BuildQuery("users");
$querydef->BuildField("login_name");

my $operator = $querydef->BuildFilterOperator(CQPerlExt::CQ_BOOL_OP_AND)
$operator->BuildFilter("group.name", CQPerlExt::CQ_COMP_OP_EQ, "sw_eng");

my $resultset = $session->BuildResultSet(querydef);
```

Find 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:

**VBScript**

```vbnet
set session = GetSession
Set resultset = session.BuildSQLQuery("select project,component,
   severity from defect where user='johndoe'")
resultset.Execute  ' Launch the query
```

**Perl**

```perl
# Perl hook scripts have the current session available in $session.
# Therefore, in this context, it is not necessary to call
# $entity->GetSession()
my $resultset=$session->BuildSQLQuery("select project,component,
   severity from defect where user='johndoe'");
$resultset->Execute;  ' Launch the query
```
Updating duplicate records to match the parent record

The following VBScript code fragment 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.

```vbnet
Dim session ' The current Session object
Dim parent_id ' The current Entity’s display name (ID string)
Dim dups ' Array of all direct duplicates of this Entity
Dim dupvar ' Variant containing a Link to a duplicate
Dim dupobj ' The same Link, but as an Object rather than a Variant
Dim entity ' The Entity extracted from the Link

If (HasDuplicates()) Then
    Set session = GetSession
    dups = GetDuplicates
    parent_id = GetDisplayName
    For Each dupvar In dups
        ' You could check these various functions for failures and then
        ' report any failures to the user (for example, using MsgBox).
        ' Failures are unlikely, but possible— for example, someone
        ' could concurrently "unmark" an entity as a duplicate.
        Set dupobj = dupvar
        Set entity = dupobj.GetChildEntity
        session.EditEntity entity, "dupdone"
        SetFieldValue "action_info", "Original " & parent_id & " is tested"
        ' commit the record to the database if validation returns no errors
        status = Validate
        if status = "" then
            Commit
            Else
            Revert
        End If
    Next
End If
```
Examples of hooks and scripts
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. This external application example contains two Visual Basic subroutines: No_state for stateless records, and Has_state for records that have states. The example

1. Uses the Session’s BuildEntity method to create an Entity object.
2. Set the values in one or more fields.
3. Validates and commits the entity.
4. Retrieves and modifies the entity.
5. Reverts the entity.

The code invokes some external routines that are not shown here:

- StdOut, which prints its arguments to a file
- DumpFields, which prints out an entity’s fields to the standard output
- ValidateAndCommit, which calls the Entity object’s Validate method and Commit method

'subroutine for stateless records
Sub No_state(session As Object)
    Dim entity As Object
    Dim failure As String

    StdOut "Test for stateless entities is starting"
    StdOut "submit a stateless entity"
    Set entity = session.BuildEntity("project")
    ' ignore failure
    failure = entity.SetFieldValue("name", "initial project name")

    DumpFields entity
    ValidateAndCommit entity
    Set entity = Nothing
Examples of hooks and scripts

StdOut "Reload, show values before modification"
Set entity = session.GetEntity("project", "initial project name")
DumpFields entity

StdOut "Modify, then show new values"
session.EditEntity entity, "modify"

' ignore the failure
failure = entity.SetFieldValue("name", "modified project name")
DumpFields entity

StdOut "revert, then show restored values"
entity.Revert
DumpFields entity

StdOut "Modify again, and commit"
session.EditEntity entity, "modify"

' ignore failure
failure = entity.SetFieldValue("name", "final project name")
ValidateAndCommit entity
Set entity = Nothing

StdOut "Reload, and show final result"
Set entity = session.GetEntity("project", "final project name")
DumpFields entity
Set entity = Nothing

StdOut "Test for stateless entities is done"
End Sub

' subroutine for stateful records
Sub Has_states(session As Object)
Dim entity As Object ' the entity that is stateful
' failure message from functions that return strings
Dim failure As String
Dim failures As Object ' iterator containing list of failure reasons
Dim id As Long ' ClearQuest defect database ID

StdOut "Test for stateful entities is starting"
StdOut "submit a stateful entity"
Set entity = session.BuildEntity("defect")

' ignore failures
failure = entity.SetFieldValue("headline", "man bites dog!")
failure = entity.SetFieldValue("project", "final project name")
failure = entity.SetFieldValue("submit_date", "03/18/2000 10:09:08")
id = entity.GetDbId

Open "XXStdout" For Append As #1
Print #1, "Entity id is": id; Chr(10);
Close #1

DumpFields entity
ValidateAndCommit entity
Set entity = Nothing

StdOut "Reload, show values before modification"
Set entity = session.GetEntityByDbId("defect", id)
DumpFields entity

StdOut "Modify then show new values"
session.EditEntity entity, "modify"
’ ignore failure
failure = entity.SetFieldValue("headline", "man bites tree!")
DumpFields entity

StdOut "revert, then show restored values"
entity.Revert
DumpFields entity

StdOut "Modify again and commit"
session.EditEntity entity, "modify"
’ ignore failure
failure = entity.SetFieldValue("headline", "tree bites man!")
ValidateAndCommit entity
Set entity = Nothing

StdOut "Reload and show before changing state"
Set entity = session.GetEntityByDbId("defect", id)
DumpFields entity

StdOut "Change to new state, then show new values"
session.EditEntity entity, "close"
failure = entity.SetFieldValue("description", “looked like an oak tree”) ’ ignore failure
DumpFields entity

StdOut "revert then show restored values"
entity.Revert
DumpFields entity

StdOut "Change to new state again then commit"
session.EditEntity entity, "close"
failure = entity.SetFieldValue("description", _
    "man of steel, tree of maple") ' ignore failure
ValidateAndCommit entity
Set entity = Nothing

StdOut "Reload, show final values"
Set entity = session.GetEntityByDbId("defect", id)
DumpFields entity
Set entity = Nothing

StdOut "Test of stateful entities is done"
End Sub
Extracting data about an EntityDef (record type)

To illustrate that you can manipulate metadata, this Visual Basic example of an external application prints the following:

- the name of the EntityDef
- the names and types of each field and action it contains
- the names of each state it contains

This subroutine makes use of another routine (not included here) called StdOut, which prints its arguments to the standard output.

```vbnet
Sub DumpOneEntityDef(edef As Object)
    ' The parameter is an EntityDef object.
    Dim names As Variant
    Dim name As String
    Dim limit As Long
    Dim index As Long

    StdOut "Dumping EntityDef " & edef.GetName

    StdOut " FieldDefs:
    names = edef.GetFieldDefNames
    If IsArray(names) Then
        index = LBound(names)
        limit = UBound(names) + 1
        Do While index < limit
            name = names(index)
            StdOut " " & name & " type=" & edef.GetFieldDefType(name)
            index = index + 1
        Loop
    End If

    StdOut " ActionDefs:
    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

End Sub
```
If edef.GetType() = AD_REQ_ENTITY Then
    StdOut " EntityDef is a REQ entity def"  ' stated record type
    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
    StdOut " EntityDef is an AUX entity def"  ' stateless record type
End If

StdOut ""
End Sub
Extracting data about a field in a record

One of the most common API calls is to the FieldInfo object. For example, the FieldInfo object has the GetValue method that enables you to get the value of a field in a record.

The following Visual Basic external application subroutine prints out the information stored in a FieldInfo object. The code invokes an external routine that is not shown here: StdOut, which prints its arguments to a file.

```vbnet
Sub DumpFieldInfo(info As Object) ' The parameter is a FieldInfo object.
    Dim temp As Long
    Dim status As String
    Dim validity As String
    Dim valuechange As String
    Dim validchange As String
    Dim value As String
    temp = info.GetValueStatus()
    If temp = AD_VALUE_NOT_AVAILABLE Then
        status = "VALUE_NOT_AVAILABLE"
    ElseIf temp = AD_HAS_VALUE Then
        status = "HAS_VALUE" value = "'" & info.GetValue() & "'
    ElseIf temp = AD_HAS_NO_VALUE Then
        status = "NO_VALUE"
    Else
        status = "<invalid value status: " & temp & ">"
    End If
    temp = info.GetValidationStatus()
    If temp = AD_KNOWN_INVALID Then
        validity = "INVALID"
    ElseIf temp = AD_KNOWN_VALID Then
        validity = "VALID"
    ElseIf temp = AD_NEEDS_VALIDATION Then
        validity = "NEEDS_VALIDATION"
    Else
        validity = "<invalid validation status: " & temp & ">"
    End If
    valuechange = ""
    If info.ValueChangedThisSetValue() Then
        valuechange = valuechange & " setval=Y"
    Else
        valuechange = valuechange & " setval=N"
```
End If

If infoValueChangedThisGroup() Then
    valuechange = valuechange & " group=Y"
Else
    valuechange = valuechange & " group=N"
End If

If infoValueChangedThisAction() Then
    valuechange = valuechange & " action=Y"
Else
    valuechange = valuechange & " action=N"
End If

validchange = ""
If infoValidityChangedThisSetValue() Then
    validchange = validchange & " setval=Y"
Else
    validchange = validchange & " setval=N"
End If

If infoValidityChangedThisGroup() Then
    validchange = validchange & " group=Y"
Else
    validchange = validchange & " group=N"
End If

If infoValidityChangedThisAction() 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() & "/"
Notifying users of changes to an entity (record)

When you want a hook to fire after a record has been committed to the database, use an action notification hook. For example, ClearQuest uses an action notification hook for the e-mail notification feature that informs multiple users after one user has committed a record to the database.

The following external application code fragment is a complete Visual Basic action notification hook. This hook

- fires when a record (entity) is modified
- uses the Entity’s GetFieldOriginalValue method and GetFieldValue method to find the values of each field before and after the modification
- uses the Session method OutputDebugString method to display messages to a debug window. (To see the debug window, go to the ClearQuest installation directory and run the dbwin32.exe utility.)

```vbs
Set session = GetSession
session.OutputDebugString "Modify action completed," & _
    " action notification hook started"

fieldnames = GetFieldNames
If IsArray(fieldnames) Then
    i = LBound(fieldnames)
    limit = UBound(fieldnames) + 1
    Do While i < limit
        onename = fieldnames(i)
        Set oldinfo = GetFieldOriginalValue(onename)
        Set newinfo = GetFieldValue(onename)

        oldstat = oldinfo.GetValueStatus
        If oldstat = AD_HAS_NO_VALUE Then
            oldempty = True
        Else
            oldempty = False
            oldval = oldinfo.GetValue
        End If

        newstat = newinfo.GetValueStatus
        If newstat = AD_HAS_NO_VALUE Then
            newempty = True
        Else
            newempty = False
```

Examples of hooks and scripts
newval = newinfo.GetValue
End If
If oldstat = AD_VALUE_UNAVAILABLE Then
    session.OutputDebugString "Field " & onename & ": original value unknown"
Else
    If newempty And Not oldempty Then
        session.OutputDebugString "Field " & onename & " has its value deleted"
    ElseIf oldempty And Not newempty Then
        session.OutputDebugString "Field " & onename & " now = " & newval
    ElseIf oldval <> newval Then
        session.OutputDebugString "Field " & onename & " was = " & oldval
    Else
        session.OutputDebugString "Field " & onename & " is unchanged"
    End If
End If

i = i + 1
Loop
End If

session.OutputDebugString "Modify action and action " & 
    "notification hook completed"
Running a query and reporting on its result set

ClearQuest client provides powerful reporting capability in a graphical user interface (GUI) environment. The ClearQuest API also supports programmatic reporting.

Sometimes all you need is the raw results rather than a highly formatted report. The following Visual Basic subroutine in an external application:

- uses an existing query object to run the query
- prints out the name of the entitydef (record type) that the query runs against
- iterates through all the records in the result set to print the label and value of each field in each record. This subroutine makes use of two other routines, not included here: StdOut, which prints its arguments to a file, and ToStr, which converts its argument to a string.

```vbnet
Sub RunBasicQuery(session As Object, querydef As Object)
    ' The parameters to this subroutine are a Session object and a QueryDef object. It is assumed that the QueryDef is valid (for example, BuildField has been used to select one or more fields to retrieve).
    Dim rsltset As Object ' This is a ResultSet object
    Dim status As Long
    Dim column As Long
    Dim num_columns As Long
    Dim num_records As Long
    Set rsltset = session.BuildResultSet(querydef)
rsltset.Execute

    StdOut "primary entity def for query == " & _
    rsltset.LookupPrimaryEntityDefName

    num_columns = rsltset.GetNumberOfColumns
    num_records = 0
    status = rsltset.MoveNext
    Do While status = AD_SUCCESS
        num_records = num_records + 1
        StdOut "Record #" & num_records
        column = 1
        Do While column <= num_columns
            StdOut " " & rsltset.GetColumnLabel(column) & "=" & _
```

Note: result set indices are based 1..N, not the usual 0..N-1 column = 1
Do While column <= num_columns
    StdOut " " & rsltset.GetColumnLabel(column) & "=" & _
ToStr(rsltset.GetColumnValue(column))
column = column + 1
Loop

StdOut ""
status = rsltset.MoveNext
Loop
End Sub
Getting session and database information

The following Visual Basic **external application** illustrates some of the Session and DatabaseDescription 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 DatabaseDescription object.

The following code prints out the information stored in the Session's DatabaseDescription object, as well as all the user-related information. This subroutine makes use of another routine (not included here) called StdOut, which prints its arguments to a file.

```vbscript
' Connect via OLE to ClearQuest
Set session = CreateObject("CLEARQUEST.SESSION")

' login_name, password, and dbname are Strings that have been set elsewhere
session.UserLogon login_name, password, dbname, AD_PRIVATE_SESSION, ""

Set dbDesc = session.GetSessionDatabase
StdOut "DB name = " & dbDesc.GetDatabaseName
StdOut "DB set name = " & dbDesc.GetDatabaseSetName
StdOut "DB connect string = " & dbDesc.GetDatabaseConnectString

StdOut "user login name = " & session.GetUserLoginName
StdOut "user full name = " & session.GetUserFullname
StdOut "user email = " & session.GetUserEmail
StdOut "user phone = " & session.GetUserPhone
StdOut "misc user info = " & session.GetUserMiscInfo

StdOut "user groups:" Set userGroups = session.GetUserGroups
```
If IsArray(userGroups) Then
  i = 0
  limit = UBound(userGroups) + 1
  Do While i < limit
    onename = userGroups(i)
    StdOut " group " & onename
    i = i + 1
  Loop
End If
Running a query against more than one record type (multitype query)

ClearQuest enables you to create a query that retrieves data from more than one record type. A Multitype query fetches data from all the records types that belong to a given "record type family". Here are some possible examples of record type families:

- "change requests" include "defects", "enhancement requests", and "documentation requests"
- "work orders" include "software fixes" and "hardware fixes"
- "issues" includes "porting," "features", and "problem incidents"

To learn about record type families, look up "record type families" in the index of *Administrating ClearQuest*.

This code fragment from an **external application** assumes that

- the schema has one record type family, "TestFamily"
- "TestFamily" contains two record types (for example, "Defect" and "Enhancement Request"

**Note:** The output of this code should be: TestFamily, True, True

**Visual Basic:**

```vbnet
dim qryDef as OAdQuerydef
dim resultSet as OAdResultset
dim familyEntDef as OAdEntityDef
' Insert code here to get the session object and log in to the database
families = session.GetEntityDefFamilyNames
if isArray(families) then
    Debug.Print UBound(families)
    for i = 0 to UBound(families)
        ' Do something with families(i)
    next i
end if
set qryDef = session.BuildQuery(families(0))
qryDef.BuildField ("Description")
set resultSet = session.BuildResultSet(qryDef)
if qryDef.IsMultiType then
    ' Do something.
end if
set familyEntDef = session.GetEntityDefFamily(families(0))
```
If familyEntDef.IsFamily Then
    ' Do something.
End If
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;
}
**Triggering a task with the destination state**

To apply some conditional logic, you can determine the destination state of the record currently undergoing an action. Here are some examples:

- Send an email to the Project Manager if a user moves a priority 1 defect into the “postponed” state.
- Allow the user to modify (reapply the “opened” state) to a defect that is currently in the “resolved” state if, and only if, that user belongs to the Manager group.

The following **action notification hook** gets the destination state and sends an email if the current record is being closed.

**Note:** This action notification hook uses a base action. A base action is an action that occurs with every action. A base action is convenient if you want a hook to fire with more than one action, such as an e-mail notification hook that fires with every action.

**VBScript**

```vbnet
Sub Defect_Notification(actionname, actiontype)
    ' actionname As String
    ' actiontype As Long
    ' action = test_base
    set cqSes = GetSession
    actionName = GetActionName
    ' 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 = $session->GetEntityDef($entity->GetEntityDefName());
    if ($entDef->GetActionCode($actionName) eq "Closed") {
        # put send notification message code here
    }
}
Enumerated Constants

This topic lists all the constants used as arguments or return values by the methods and properties in the ClearQuest API, except as otherwise noted.

**Note:** For the difference between VBScript and Perl constants, see Notation Conventions for VBScript/Visual Basic and Notation Conventions for Perl.

**ActionType**

The ActionType constants define the legal action types in VBScript.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_SUBMIT</td>
<td>1</td>
<td>Create a new record.</td>
</tr>
<tr>
<td>_MODIFY</td>
<td>2</td>
<td>Change the contents of a record.</td>
</tr>
<tr>
<td>_CHANGE_STATE</td>
<td>3</td>
<td>Change the state of a record.</td>
</tr>
<tr>
<td>_DUPLICATE</td>
<td>4</td>
<td>Mark the record as a duplicate of another record.</td>
</tr>
<tr>
<td>_GETACTIONNAME</td>
<td></td>
<td>Get the name that belongs to the current Entity object action.</td>
</tr>
<tr>
<td>_GETACTIONTYPE</td>
<td></td>
<td>Get the action type that belongs to the current Entity object method.</td>
</tr>
<tr>
<td>_UNDUPLICATE</td>
<td>5</td>
<td>Undo the DUPLICATE action.</td>
</tr>
<tr>
<td>_IMPORT</td>
<td>6</td>
<td>Import a new record.</td>
</tr>
<tr>
<td>_DELETE</td>
<td>7</td>
<td>Delete an entity.</td>
</tr>
<tr>
<td>_BASE</td>
<td>8</td>
<td>Base actions fire with all other actions. [See the Schemas and Packages appendix of Administrating ClearQuest.]</td>
</tr>
<tr>
<td>_RECORD_SCRIPT_ALIAS</td>
<td>9</td>
<td>Allows you to call one single method. GetActionName, instead of having to call three: EditEntity method, Validate method, and Commit method.</td>
</tr>
</tbody>
</table>
**Behavior**

The Behavior constants identify the behavior of the designated field.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_MANDATORY</td>
<td>1</td>
<td>A value must be provided. Corresponds to the MANDATORY field behavior in the user interface.</td>
</tr>
<tr>
<td>_OPTIONAL</td>
<td>2</td>
<td>A value may be provided but is not required. Corresponds to the OPTIONAL field behavior in the user interface.</td>
</tr>
<tr>
<td>_READONLY</td>
<td>3</td>
<td>The designated field cannot be changed. Corresponds to the READONLY field behavior in the user interface.</td>
</tr>
<tr>
<td>_USE_HOOK</td>
<td>4</td>
<td>The behavior of the field is determined by calling the associated hook. Corresponds to the USE_HOOK field behavior in the user interface.</td>
</tr>
</tbody>
</table>

**BoolOp**

The BoolOp constants identify the valid boolean operations.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_BOOL_OP_AND</td>
<td>1</td>
<td>Boolean AND operator</td>
</tr>
<tr>
<td>_BOOL_OP_OR</td>
<td>2</td>
<td>Boolean OR operator</td>
</tr>
</tbody>
</table>

**CompOp**

The CompOp constants identify the valid comparison operators.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_COMP_OP_EQ</td>
<td>1</td>
<td>Equality operator (=)</td>
</tr>
<tr>
<td>_COMP_OP_NEQ</td>
<td>2</td>
<td>Inequality operator (&lt;&gt;)</td>
</tr>
<tr>
<td>_COMP_OP_LT</td>
<td>3</td>
<td>Less-than operator (&lt;)</td>
</tr>
<tr>
<td>_COMP_OP_LTE</td>
<td>4</td>
<td>Less-than or Equal operator (&lt;=)</td>
</tr>
</tbody>
</table>
Enumerated Constants

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_COMP_OP_GT</td>
<td>5</td>
<td>Greater-than operator (&gt;)</td>
</tr>
<tr>
<td>_COMP_OP_GTE</td>
<td>6</td>
<td>Greater-than or Equal operator (&gt;=)</td>
</tr>
<tr>
<td>_COMP_OP_LIKE</td>
<td>7</td>
<td>Like operator (value is a substring of the string in the given field)</td>
</tr>
<tr>
<td>_COMP_OP_NOT_LIKE</td>
<td>8</td>
<td>Not-like operator (value is not a substring of the string in the given field)</td>
</tr>
<tr>
<td>_COMP_OP_BETWEEN</td>
<td>9</td>
<td>Between operator (value is between the specified delimiter values)</td>
</tr>
<tr>
<td>_COMP_OP_NOT_BETWEEN</td>
<td>10</td>
<td>Not-between operator (value is not between specified delimiter values)</td>
</tr>
<tr>
<td>_COMP_OP_IS_NULL</td>
<td>11</td>
<td>Is-NULL operator (field does not contain a value)</td>
</tr>
<tr>
<td>_COMP_OP_IS_NOT_NULL</td>
<td>12</td>
<td>Is-not-NULL operator (field contains a value)</td>
</tr>
<tr>
<td>_COMP_OP_IN</td>
<td>13</td>
<td>In operator (value is in the specified set)</td>
</tr>
<tr>
<td>_COMP_OP_NOT_IN</td>
<td>14</td>
<td>Not-in operator (value is not in the specified set)</td>
</tr>
</tbody>
</table>

DatabaseVendor

The DatabaseVendor constants identify the supported database types.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_SQL_SERVER</td>
<td>1</td>
<td>An SQL Server database.</td>
</tr>
<tr>
<td>_MS_ACCESS</td>
<td>2</td>
<td>An MS Access database.</td>
</tr>
<tr>
<td>_SQL_ANYWHERE</td>
<td>3</td>
<td>An SQL Anywhere database.</td>
</tr>
<tr>
<td>_ORACLE7</td>
<td>4</td>
<td>An Oracle database using Oracle7 client networking software</td>
</tr>
</tbody>
</table>
Enumerated Constants

**EntityType**

The EntityType constants identify state-based or stateless records.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_REQ_ENTITY</td>
<td>1</td>
<td>State-based records</td>
</tr>
<tr>
<td>_AUX_ENTITY</td>
<td>2</td>
<td>Stateless records</td>
</tr>
<tr>
<td>_ANY_ENTITY</td>
<td>3</td>
<td>Either state-based or stateless records</td>
</tr>
</tbody>
</table>

**EventType**

The Type constants identify the cause of hook invocations.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_BUTTON_CLICK</td>
<td>1</td>
<td>The hook invocation is triggered by a push button click.</td>
</tr>
<tr>
<td>_SUBDIALOG_BUTTON_CLICK</td>
<td>2</td>
<td>The hook invocation is triggered by a click on the subdialog button.</td>
</tr>
<tr>
<td>_SELECTION</td>
<td>3</td>
<td>The hook invocation is triggered by an item selection.</td>
</tr>
<tr>
<td>_DBLCLICK</td>
<td>4</td>
<td>The hook invocation is triggered by a point device double-click.</td>
</tr>
<tr>
<td>_CONTEXTMENU_ITEM_SELECTION</td>
<td>5</td>
<td>The hook invocation is triggered by a contextual menu selection.</td>
</tr>
<tr>
<td>_CONTEXTMENU_ITEM_CONDITION</td>
<td>6</td>
<td>Indicates whether the hook should enable or disable a contextual menu item. A string value of &quot;1&quot; indicates the item should be enabled. A String value of &quot;0&quot; indicates the item should be disabled.</td>
</tr>
</tbody>
</table>
FetchStatus

The FetchStatus constants identify the status of moving the cursor in a request set.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_SUCCESS</td>
<td>1</td>
<td>The next record in the request set was successfully obtained.</td>
</tr>
<tr>
<td>_NO_DATA_FOUND</td>
<td>2</td>
<td>No more records were found in the request set.</td>
</tr>
<tr>
<td>_MAX_ROWS_EXCEEDED</td>
<td>3</td>
<td>Not used.</td>
</tr>
</tbody>
</table>

FieldType

The FieldType constants identify the information contained in a field.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_SHORT_STRING</td>
<td>1</td>
<td>Simple text field (255 character limit)</td>
</tr>
<tr>
<td>_MULTILINE_STRING</td>
<td>2</td>
<td>Arbitrarily long text</td>
</tr>
<tr>
<td>_INT</td>
<td>3</td>
<td>Integer</td>
</tr>
<tr>
<td>_DATE_TIME</td>
<td>4</td>
<td>Timestamp information</td>
</tr>
<tr>
<td>_REFERENCE</td>
<td>5</td>
<td>A pointer to a stateless record</td>
</tr>
<tr>
<td>_REFERENCE_LIST</td>
<td>6</td>
<td>A list of references</td>
</tr>
<tr>
<td>_ATTACHMENT_LIST</td>
<td>7</td>
<td>A list of attached files</td>
</tr>
<tr>
<td>_ID</td>
<td>8</td>
<td>A special string ID for records</td>
</tr>
<tr>
<td>_STATE</td>
<td>9</td>
<td>The current state of a state-based record</td>
</tr>
<tr>
<td>_JOURNAL</td>
<td>10</td>
<td>A special list of rows in a subtable that belongs exclusively to this record</td>
</tr>
<tr>
<td>_DBID</td>
<td>11</td>
<td>A special internal numeric ID</td>
</tr>
</tbody>
</table>
FieldValidationStatus

The FieldValidationStatus constants identify the status of the designated field.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_KNOWN_VALID</td>
<td>1</td>
<td>The field's value is known to be valid.</td>
</tr>
<tr>
<td>_KNOWN_INVALID</td>
<td>2</td>
<td>The field's value is known to be invalid.</td>
</tr>
<tr>
<td>_NEEDS_VALIDATION</td>
<td>3</td>
<td>The field's value may be valid but has not been checked.</td>
</tr>
</tbody>
</table>

QueryType

The QueryType constants indentify the type of stored query.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_LIST_QUERY</td>
<td>1</td>
<td>A list that corresponds to the result set grid in ClearQuest Designer.</td>
</tr>
<tr>
<td>_REPORT_QUERY</td>
<td>2</td>
<td>A report that corresponds to a report in the ClearQuest Designer workspace.</td>
</tr>
<tr>
<td>_CHART_QUERY</td>
<td>3</td>
<td>A chart that corresponds to a chart in the ClearQuest Designer workspace.</td>
</tr>
</tbody>
</table>

SessionType

The SessionType constants identify the type of session desired.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_SHARED_SESSION</td>
<td>1</td>
<td>More than one client can access this session's data</td>
</tr>
<tr>
<td>_PRIVATE_SESSION</td>
<td>2</td>
<td>Only one client can access this session's data</td>
</tr>
<tr>
<td>_ADMIN_SESSION</td>
<td>3</td>
<td>The system administrator is logged into the session.</td>
</tr>
</tbody>
</table>
### ValueStatus

The ValueStatus constants identify the status of a field. OLEWKSPCQUERYTYPE

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_HAS_NO_VALUE</td>
<td>1</td>
<td>The field has no value set.</td>
</tr>
<tr>
<td>_HAS_VALUE</td>
<td>2</td>
<td>The field has a value.</td>
</tr>
<tr>
<td>_VALUE_NOT_AVAILABLE</td>
<td>3</td>
<td>The current state of the field prevents it from returning a value.</td>
</tr>
</tbody>
</table>

**Note:** The following constants do not use the notational convention.

The OLEWKSPCQUERYTYPE constants identify the desired source of a query.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OLEWKSPCQUERIESNONE</td>
<td>0</td>
<td>Do not return queries.</td>
</tr>
<tr>
<td>OLEWKSPCSYSTEMQUERIES</td>
<td>1</td>
<td>Return system queries only.</td>
</tr>
<tr>
<td>OLEWKSPCUSERQUERIES</td>
<td>2</td>
<td>Return user queries only.</td>
</tr>
<tr>
<td>OLEWKSPCBOTHQUERIES</td>
<td>3</td>
<td>Return either system or user queries.</td>
</tr>
</tbody>
</table>

### OLEWKSPCREPORTTYPE

**Note:** The following constants do not use the notational convention.

The OLEWKSPCREPORTTYPE constants identify the desired source of a report.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OLEWKSPCREPORTSNONE</td>
<td>0</td>
<td>Do not return reports</td>
</tr>
<tr>
<td>OLEWKSPCSYSTEMREPORTS</td>
<td>1</td>
<td>Return system reports only.</td>
</tr>
<tr>
<td>OLEWKSPCUSERREPORTS</td>
<td>2</td>
<td>Return user reports only.</td>
</tr>
<tr>
<td>OLEWKSPCBOTHREPORTS</td>
<td>3</td>
<td>Return either system or user reports.</td>
</tr>
</tbody>
</table>
### Session object properties:

<table>
<thead>
<tr>
<th>Property name</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NameValue property</td>
<td>Read/Write</td>
<td>Gets or sets the value of one of this property's named variables.</td>
</tr>
</tbody>
</table>

### Session object methods:

<table>
<thead>
<tr>
<th>Method name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BuildEntity method</td>
<td>Creates a new record of the specified type and begins a Submit action.</td>
</tr>
<tr>
<td>BuildQuery method</td>
<td>Creates and returns a new QueryDef object for the specified record type.</td>
</tr>
<tr>
<td>BuildResultSet method</td>
<td>Creates and returns a result set that can be used to run a query.</td>
</tr>
<tr>
<td>BuildSQLQuery method</td>
<td>Creates and returns a ResultSet object using a raw SQL string.</td>
</tr>
<tr>
<td>DeleteEntity method</td>
<td>Deletes the specified record from the current database.</td>
</tr>
<tr>
<td>EditEntity method</td>
<td>Performs the specified action on a record and makes the record available for editing.</td>
</tr>
<tr>
<td>FireRecordScriptAlias method</td>
<td>Calls the action that calls the hook script; use to simulate a user choosing an action that launches a hook.</td>
</tr>
<tr>
<td>GetAccessibleDatabases</td>
<td>Returns a list of databases that are available for the specified user to log in to.</td>
</tr>
<tr>
<td>GetAuxEntityDefNames method</td>
<td>Returns an array of Strings, each of which corresponds to the name of one of the schema stateless record types.</td>
</tr>
<tr>
<td>GetDefaultEntityDef method</td>
<td>Returns the schema's default EntityDef object.</td>
</tr>
<tr>
<td>GetEnabledEntityDefs method</td>
<td>Returns the EntityDefs collection object enabled in the current schema for a given package revision.</td>
</tr>
<tr>
<td>GetEnabledPackageRevs method</td>
<td>Returns a collection object representing the packageRev set that is enabled for the current revision of the schema.</td>
</tr>
<tr>
<td>GetEntity method</td>
<td>Returns the specified record.</td>
</tr>
<tr>
<td>GetEntityByDbId method</td>
<td>Returns the record with the specified database ID.</td>
</tr>
<tr>
<td>Method name</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>GetEntityDef method</td>
<td>Returns the specified EntityDef object if it is a family.</td>
</tr>
<tr>
<td>GetEntityDefFamily method</td>
<td>Returns the requested EntityDef object if it is a family.</td>
</tr>
<tr>
<td>GetEntityDefNames method</td>
<td>Returns an array containing the requested EntityDef family names.</td>
</tr>
<tr>
<td>GetEntityDefNames method</td>
<td>Returns an array containing the names of the record types in the current database’s schema.</td>
</tr>
<tr>
<td>GetInstalledMasters method</td>
<td>Returns the list of registered database sets and master databases.</td>
</tr>
<tr>
<td>GetQueryEntityDefNames method</td>
<td>Returns an array containing the names of the record types that are suitable for use in queries.</td>
</tr>
<tr>
<td>GetReqEntityDefNames method</td>
<td>Returns an array containing the names of the state-based record types in the current database’s schema.</td>
</tr>
<tr>
<td>GetServerInfo method</td>
<td>Returns the name of the session’s OLE server.</td>
</tr>
<tr>
<td>GetSessionDatabase method</td>
<td>Returns general information about the database that is being accessed in the current session.</td>
</tr>
<tr>
<td>GetSubmitEntityDefNames method</td>
<td>Returns an array containing the names of the record types that are suitable for use in creating a new record.</td>
</tr>
<tr>
<td>GetUserEmail method</td>
<td>Returns the electronic mail address of the user who is logged in for this session.</td>
</tr>
<tr>
<td>GetUserFullName method</td>
<td>Returns the full name of the user who is logged in for this session.</td>
</tr>
<tr>
<td>GetUserGroups method</td>
<td>Returns a list of the groups to which the current user belongs.</td>
</tr>
<tr>
<td>GetUserLoginName method</td>
<td>Returns the name that was used to log in for this session.</td>
</tr>
<tr>
<td>GetUserMiscInfo method</td>
<td>Returns miscellaneous information about the user who is logged in for this session.</td>
</tr>
<tr>
<td>GetUserPhone method</td>
<td>Returns the telephone number of the user who is logged in for this session.</td>
</tr>
<tr>
<td>GetWorkSpace method</td>
<td>Returns the session’s WORKSPACE object.</td>
</tr>
<tr>
<td>HasValue method</td>
<td>Returns a Bool indicating whether the specified session variable exists.</td>
</tr>
<tr>
<td>IsMetadataReadonly method</td>
<td>Returns a boolean indicating whether the session’s metadata is read-only.</td>
</tr>
</tbody>
</table>
### Method name

<table>
<thead>
<tr>
<th>Method name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MarkEntityAsDuplicate method</td>
<td>Modifies the specified record to indicate that it is a duplicate of another record.</td>
</tr>
<tr>
<td>OpenQueryDef method</td>
<td>Loads a query from a file.</td>
</tr>
<tr>
<td>OutputDebugString method</td>
<td>Specifies a message that can be displayed by a debugger or a similar tool.</td>
</tr>
<tr>
<td>UnmarkEntityAsDuplicate method</td>
<td>Removes the indication that the specified record is a duplicate of another record.</td>
</tr>
<tr>
<td>UserLogon method</td>
<td>Log in as the specified user for a database session.</td>
</tr>
</tbody>
</table>

**See Also:**

Session and DatabaseDescription Code Example
**NameValue property**

Gets or sets the value of one of this property’s named variables.

**VB Syntax:**

```
session.NameValue name
session.NameValue name, newValue
```

**Perl Syntax:**

```
$database->GetNameValue(name);
$database->SetNameValue(name, newValue);
```

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>session</code></td>
<td>The Session object that represents the current database-access session.</td>
</tr>
<tr>
<td><code>name</code></td>
<td>A String containing the name of the variable to get or set.</td>
</tr>
<tr>
<td><code>newValue</code></td>
<td>A reference to a Variant containing the new value for the variable.</td>
</tr>
</tbody>
</table>

**Return value**
A Variant when getting a value. Nothing when setting a value.

**Member of:** Session object

**Remarks:**

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.

**Examples:**

```vbnet
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"

See Also:

HasValue method
BuildEntity method

Creates a new record of the specified type and begins a "submit" action.

**VB Syntax:**

```
session.BuildEntity entitydef_name
```

**Perl Syntax:**

```
$session->BuildEntity(entitydef_name);
```

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>session</code></td>
<td>The Session object that represents the current database-access session.</td>
</tr>
<tr>
<td><code>entitydef_name</code></td>
<td>A String containing the name of the EntityDef object to use as a template when creating the record.</td>
</tr>
</tbody>
</table>

**Return value**

A new Entity object that was built using the named EntityDef object as a template.

**Member of:** Session object

**Remarks:**

This method creates a new record and initiates a "submit" action, thus enabling you to begin editing the record’s contents. (You do not need to call EditEntity method to make the record editable.) You can assign values to the new record’s fields using the SetFieldValue method of the returned Entity object. When you are done updating the record, use the Validate method and Commit method of the Entity object to validate and commit any changes you made to the record, respectively.

The name you specify in the `entitydef_name` parameter must also correspond to an appropriate record type in the schema. To obtain a list of legal names for `entitydef_name`, use the GetSubmitEntityDefNames method.
**Examples:**

```vba
set sessionObj = GetSession

' Create a new "defect" record
set entityObj = sessionObj.BuildEntity("defect")
```

**See Also:**

- `EditEntity method`
- `GetEntity method`
- `GetSubmitEntityDefNames method`
- `Commit method of the Entity object`
- `SetFieldValue method of the Entity object`
- `Validate method of the Entity object`
- `Entity object`
- `Entity Code Example`
**BuildQuery method**

Creates and returns a new QueryDef object for the specified record type.

**VB Syntax:**

```
session.BuildQuery entitydef_name
```

**Perl Syntax:**

```
$session->BuildQuery(entitydef_name);
```

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>session</td>
<td>The Session object that represents the current database-access session.</td>
</tr>
<tr>
<td>entitydef_name</td>
<td>A String containing the name of the EntityDef object to use as a template when creating the record.</td>
</tr>
</tbody>
</table>

**Return value**

A new QueryDef object. This object contains no filters or build fields.

**Member of:** Session object

**Remarks:**

You can use the returned QueryDef object to build a query for searching records whose record type matches the specified EntityDef. Before you can perform the search, you must add at least one field to query’s display list by calling the BuildField method of the QueryDef object. You can also add filters to the QueryDef object to specify the search criteria. For more information on specifying this information, see the description and methods of the QueryDef object.

The name you specify in the entitydef_name parameter must correspond to an appropriate record type in the schema. To obtain a list of legal names for entitydef_name, use the GetQueryEntityDefNames method.

Before you can run the query, you must associate the QueryDef object with a ResultSet Object. See the BuildResultSet method for information on how to do this.
Examples:

set sessionObj = GetSession

' Create a query for "defect" records
set queryDefObj = sessionObj.BuildQuery("defect")

See Also:

BuildResultSet method
GetEntityDefNames method
GetQueryEntityDefNames method
BuildField method of the QueryDef object
QueryDef object
ResultSet Object
BuildQuery Code Example
BuildResultSet method

Creates and returns a result set that can be used to run a query.

**VB Syntax:**

`session.BuildResultSet querydef`

**Perl Syntax:**

`$session->BuildResultSet(querydef);`

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>session</td>
<td>The Session object that represents the current database-access session.</td>
</tr>
<tr>
<td>querydef</td>
<td>A QueryDef object that defines the desired query.</td>
</tr>
<tr>
<td><strong>Return value</strong></td>
<td>A ResultSet Object suitable for eventual execution of the query.</td>
</tr>
</tbody>
</table>

**Member of:** Session object

**Remarks:**

This method creates a ResultSet object for the specified QueryDef object. You can then use the returned ResultSet object to run the query and store the resulting data.

Do not call this method until you have added all of the desired fields and filters to the QueryDef object. This method uses the information in the QueryDef object to build the set of data structures needed to store the query data. If you add new fields or filters to the QueryDef object after calling this method, the ResultSet object will not reflect the new additions. To run the query and fetch the resulting data, you must subsequently call the ResultSet object’s Execute method.

**Note:**

To obtain the QueryDef object that you pass to this method, you must call the BuildQuery method. To construct a ResultSet object directly from a raw SQL query string, use the BuildSQLQuery method.
Examples:

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)

See Also:

BuildQuery method
BuildSQLQuery method
Execute method of the ResultSet object
QueryDef object
ResultSet Object
BuildQuery Code Example
ResultSet Code Example
BuildSQLQuery method

Creates and returns a ResultSet object using a raw SQL string.

VB Syntax:

```
session.BuildSQLQuery SQL_string
```

Perl Syntax:

```
$session->BuildSQLQuery($SQL_string);
```

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>session</td>
<td>The Session object that represents the current database-access session.</td>
</tr>
<tr>
<td>SQL_string</td>
<td>A String containing the raw SQL commands for the query.</td>
</tr>
<tr>
<td>Return value</td>
<td>A ResultSet Object suitable for running the query.</td>
</tr>
</tbody>
</table>

Member of: Session object

Remarks:

We recommend you use the ClearQuest API to define a query and filter(s), as opposed to writing raw SQL.

Like BuildResultSet method, this method creates a ResultSet object that you can use to run a query. Unlike BuildResultSet, this method uses a raw SQL string instead of a QueryDef object to build the data structures of the ResultSet object. Do not call this method until you have completely constructed the SQL query string.

Like BuildResultSet method, this method generates the data structures needed to store the query data but does not fetch the data. To run the query and fetch the resulting data, you must call the ResultSet object’s Execute method.

Unlike BuildResultSet, BuildSQLQuery makes no use of a QueryDef object, so the query defined by the SQL string cannot be manipulated before constructing the ResultSet.
Examples:

```javascript
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)
```

See Also:

- BuildQuery method
- ResultSet Object
- BuildQuery Code Example
DeleteEntity method

Deletes the specified record from the current database.

**VB Syntax:**

```
session.DeleteEntity entity, deleteActionName
```

**Perl Syntax:**

```
$search->DeleteEntity(entity, deleteActionName);
```

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>session</td>
<td>The Session object that represents the current database-access session.</td>
</tr>
<tr>
<td>entity</td>
<td>The Entity object corresponding to the record to be deleted.</td>
</tr>
<tr>
<td>deleteActionName</td>
<td>A String containing the name of the action to use when deleting the entity.</td>
</tr>
</tbody>
</table>

**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 (""").

**Member of:** Session object

**Remarks:**

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.

**Examples:**

```
set sessionObj = GetSession

' Delete the record whose ID is "BUGDB00000042" using the "delete" action
set objToDelete = sessionObj.GetEntity("defect", "BUGDB00000042")
sessionObj.DeleteEntity objToDelete, "delete"
```
See Also:

BuildEntity method
EditEntity method
GetEntity method
Entity object
**EditEntity method**

Performs the specified action on a record and makes the record available for editing.

**VB Syntax:**

```
session.EditEntity entity, edit_action_name
```

**Perl Syntax:**

```
$session->EditEntity(entity, edit_action_name);
```

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>session</td>
<td>The Session object that represents the current database-access session.</td>
</tr>
<tr>
<td>entity</td>
<td>The Entity object corresponding to the record that is to be edited.</td>
</tr>
<tr>
<td>edit_action_name</td>
<td>A String containing the name of the action to initiate for editing. (For example: &quot;modify&quot; or &quot;resolve&quot;)</td>
</tr>
</tbody>
</table>

**Return value**: None.

**Member of**: Session object

**Remarks:**

The Entity object you specify in the entity parameter must have been previously obtained by calling the GetEntityByDbId method or GetEntity method method, or by running a query. If you created the Entity object using the BuildEntity method and have not yet committed it to the database, the object is already available for editing.

To obtain a list of legal values for the edit_action_name parameter, call the GetActionDefNames method of the appropriate EntityDef object.

After calling this method, you can call the methods of the Entity object to modify the fields of the corresponding record. When you are done editing the record, validate it and commit your changes to the database by calling the Entity object’s Validate and Commit methods, respectively.
Examples:

```
set sessionObj = GetSession

' Edit the record whose ID is "BUGDB00000010" using the "modify" action
set objToEdit = sessionObj.GetEntity("defect", "BUGDB00000010")
sessionObj.EditEntity objToEdit, "modify"
```

See Also:

- BuildEntity method
- GetEntity method
- GetEntityByDbId method
- Commit method of the Entity object
- Validate method of the Entity object
- GetActionDefNames method of the EntityDef object
- Entity object
- ActionType
- Duplicates Code Example
- Entity Code Example
FireRecordScriptAlias method

Calls the action that calls the hook script.

**VB Syntax:**

`session.FireRecordScriptAlias entity, editActionName`

**Perl Syntax:**

`$session->FireRecordScriptAlias(entity, editActionName);`

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entity</td>
<td>The entity must be an entity object previously returned by BuildEntity, GetEntityById, or GetEntityByDisplayName.</td>
</tr>
<tr>
<td>editActionName</td>
<td>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.</td>
</tr>
<tr>
<td>Return value</td>
<td>A String containing the script return value determined by the hook.</td>
</tr>
</tbody>
</table>

**Member of:** Session object

**Remarks:**

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.

**See Also:**

_RECORD_SCRIPT_ALIAS constant in ActionType
Commit method
EditEntity method
Validate method
FireNamedHook method
Notation Conventions
GetAccessibleDatabases method

Returns a list of databases that are available for the specified user to log in to.

**VB Syntax:**

`session.GetAccessibleDatabases master_db_name, user_login_name, database_set`

**Perl Syntax:**

`$session->GetAccessibleDatabases(master_db_name, user_login_name, database_set);`

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>session</code></td>
<td>The Session object that represents the current database-access session.</td>
</tr>
<tr>
<td><code>master_db_name</code></td>
<td>A String that specifies the schema repository.</td>
</tr>
<tr>
<td><code>user_login_name</code></td>
<td>A String that specifies the user’s login.</td>
</tr>
<tr>
<td><code>database_set</code></td>
<td>A String that specifies the database set in which to look for accessible databases. By default, this argument should contain the empty String.</td>
</tr>
</tbody>
</table>

**Return value**

A Variant containing an Array whose elements are Variants of type DatabaseDescription object.

**Member of:** Session object

**Remarks:**

This method returns only the databases that the specified user is allowed to log in to. If the `user_login_name` parameter contains an empty String, this method returns a list of all of the databases associated with the specified master database.

You can examine each DatabaseDescription object to get the corresponding database’s name and other information needed to log in to it.
Examples:

```
set sessionObj = GetSession

' Get the list of databases in the
' "ClearQuest 1.1" master database set.
databases = sessionObj.GetAccessibleDatabases("ClearQuest 1.1", ",", ")
for each db in databases
    ' Get the name of the database
    dbName = db.GetDatabaseName
Next
```

See Also:

UserLogon method
GetDatabaseName method of the DatabaseDescription object
DatabaseDescription object
GetAuxEntityDefNames method

Returns an array of Strings, each of which corresponds to the name of one of the schema stateless record types.

**VB Syntax:**

```
session.GetAuxEntityDefNames
```

**Perl Syntax:**

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

**Identifier** | **Description**
--- | ---
```
session
```
| The Session object that represents the current database-access session.

**Return value** | A Variant containing an Array of Strings. Each String contains the name of a stateless record type.

**Member of:** Session object

**Remarks:**

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.

**Examples:**

```
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

See Also:

GetEntityDef method
GetEntityDefNames method
GetQueryEntityDefNames method
GetReqEntityDefNames method
GetSubmitEntityDefNames method
GetDefaultEntityDef method

Returns the schema’s default EntityDef object.

**VB Syntax:**

`session.GetDefaultEntityDef`  

**Perl Syntax:**

`$session->GetDefaultEntityDef();`

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>session</code></td>
<td>The Session object that represents the current database-access session.</td>
</tr>
</tbody>
</table>

**Return value**: The default EntityDef object.

**Member of**: Session object

**Remarks:**

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 (""").

**Examples:**

```vbnet
set sessionObj = GetSession  
set defEntityDef = sessionObj.GetDefaultEntityDef
```

**See Also:**

GetEntityDef method  
EntityDef object
GetEnabledEntityDefs method

Returns the EntityDefs collection object enabled in the current schema for a given package revision.

**VB Syntax:**

```vbnet
schemaRev.GetEnabledEntityDefs packName, rev
```

**Perl Syntax:**

```perl
$schemaRev->GetEnabledEntityDefs(packName, rev);
```

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>packName</td>
<td>A String that specifies the package name.</td>
</tr>
<tr>
<td>rev</td>
<td>A String that specifies the package revision.</td>
</tr>
</tbody>
</table>

**Return value** The EntityDefs object for the current package revision.

**Member of:** SchemaRev object, Session object

**Remarks:**

Use with GetEnabledPackageRevs method to discover which packages and package revisions apply to the current user database.

**See Also:**

GetEnabledPackageRevs method
GetEnabledEntityDefs method of the SchemaRev object
GetEnabledPackageRevs method

Returns a collection object representing the packageRev set that is enabled in the current revision of the schema.

**VB Syntax:**

```vbnet
session.GetEnabledPackageRevs PackageName, RevString
```

**Perl Syntax:**

```perl
$session->GetEnabledPackageRevs(PackageName, RevString);
```

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PackageName</td>
<td>Name of the package.</td>
</tr>
<tr>
<td>RevString</td>
<td>Represents the revision of the package.</td>
</tr>
</tbody>
</table>

**Return values**
The collection object of the packageRev set.

**Member of:** SchemaRev object, Session object

**Remarks:**

You can call this method from the Session object, in which case the schema revision is already known when you log onto the user database.

See this method, GetEnabledPackageRevs method, in its other object, SchemaRev object, for an alternative use.

**See Also:**

GetEntity method
GetEnabledPackageRevs method in SchemaRev object
GetEntity method

Returns the specified record.

**VB Syntax:**

```
session.GetEntity entity def_name, display_name
```

**Perl Syntax:**

```
$session->GetEntity(entity def_name, display_name);
```

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>session</code></td>
<td>The Session object that represents the current database-access session.</td>
</tr>
<tr>
<td><code>entity def_name</code></td>
<td>A String that identifies the name of the record type to which the record belongs.</td>
</tr>
<tr>
<td><code>display_name</code></td>
<td>A String that identifies the record.</td>
</tr>
</tbody>
</table>

**Return value**

An Entity object corresponding to the requested record.

**Member of:** Session object

**Remarks:**

When requesting a state-based record type, the `display_name` parameter must contain the visible ID of the record (for example, “DEF00013323”). For stateless record type, this parameter must contain the value of the record’s unique key field.

To request a record using its database ID instead of its visible ID, use the `GetEntityById` method.

**Examples:**

```
set sessionObj = GetSession

set record1 = sessionObj.GetEntity("defect", "DEF00013323")
```
See Also:

BuildEntity method
EditEntity method
GetEntityByDbId method
Entity Code Example
**GetEntityByDbId method**

Returns the record with the specified database ID.

**VB Syntax:**

`session.GetEntityByDbId entitydef_name, db_id`

**Perl Syntax:**

`$session->GetEntityByDbId(entitydef_name, db_id);`

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>session</code></td>
<td>The Session object that represents the current database-access session.</td>
</tr>
<tr>
<td><code>entitydef_name</code></td>
<td>A String that identifies the name of the record type to which the desired record belongs.</td>
</tr>
<tr>
<td><code>db_id</code></td>
<td>A Long that is the number used by the database to identify the record.</td>
</tr>
</tbody>
</table>

**Return value**

An Entity object corresponding to the requested record.

**Member of:** Session object

**Remarks:**

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.

**Examples:**

```vbscript
' Save this record’s ID for later use.
set sessionObj = GetSession

id = entity.GetDbId

...```
' Get the record again
set record = sessionObj.GetEntityByDbId("defect", id)

See Also:

BuildEntity method
EditEntity method
GetEntity method
GetDbId method of the Entity object
Entity Code Example
GetEntityDef method

Returns the specified EntityDef object.

VB Syntax:

`session.GetEntityDef(entitydef_name)`

Perl Syntax:

`$session->GetEntityDef(entitydef_name);`

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>session</td>
<td>The Session object that represents the current database-access session.</td>
</tr>
<tr>
<td>entitydef_name</td>
<td>A String containing the name of an EntityDef object.</td>
</tr>
<tr>
<td>Return value</td>
<td>The requested EntityDef object.</td>
</tr>
</tbody>
</table>

Member of: Session object

Remarks:

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).

Examples:

```vbnet
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

See Also:

GetAuxEntityDefNames method
GetEntityDefNames method
GetQueryEntityDefNames method
GetReqEntityDefNames method
GetSubmitEntityDefNames method
EntityDef object
GetEntityDefFamily method
GetEntityDefFamilyNames method
GetIsMaster method of the DatabaseDescription object
**GetEntityDefFamily method**

Returns the named family EntityDef object.

**VB Syntax:**

```vbnet
session.GetEntityDefFamily entitydefName
```

**Perl Syntax:**

```perl
$session->GetEntityDefFamily(entitydefName);
```

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>session</td>
<td>The Session object that represents the current database-access session.</td>
</tr>
<tr>
<td>entitydefName</td>
<td>A String containing the name of an EntityDef object.</td>
</tr>
<tr>
<td>Return value</td>
<td>The requested EntityDef object.</td>
</tr>
</tbody>
</table>

**Member of:** Session object

**Remarks:**

Returns a valid object if entitydefName corresponds to a family. This method is convenient if you expect the record type to belong to an family. Otherwise, see the IsFamily method.

**Example:**

```vbnet
gentitydefFamily
    Returns the name of a family EntityDef.
```

**See Also:**

IsFamily method
GetEntityDefFamilyNames method
GetIsMaster method of the DatabaseDescription object DatabaseDescription object
GetEntityDefFamilyNames method

Returns an array that contains the names of all family EntityDefs in the schema repository.

**VB Syntax:**

`session.GetEntityDefFamilyNames`

**Perl Syntax:**

`$session->GetEntityDefFamilyNames();`

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>session</code></td>
<td>The Session object that represents the current database-access session.</td>
</tr>
<tr>
<td><code>Return value</code></td>
<td>The requested EntityDef names in a String array.</td>
</tr>
</tbody>
</table>

**Member of:** Session object

**Remarks:**

Provides support for multitype queries.

**See Also:**

IsFamily method
GetEntityDefFamily method
GetEntityDefNames method
GetEntityDefNames method

Returns an array containing the names of the record types in the current database’s schema.

**VB Syntax:**

```vb
session.GetEntityDefNames
```

**Perl Syntax:**

```perl
$session->GetEntityDefNames();
```

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>session</code></td>
<td>The Session object that represents the current database-access session.</td>
</tr>
<tr>
<td><strong>Return value</strong></td>
<td>A Variant containing an array of Strings. Each string in the array contains the name of a single EntityDef in the schema.</td>
</tr>
</tbody>
</table>

**Member of:** Session object

**Remarks:**

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.

**Examples:**

```vbnet
set sessionObj = GetSession

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

' Iterate over all the record types
for each name in entityDefNames
    set entityDefObj = sessionObj.GetEntityDef(name)
    ' Do something with the EntityDef object
    Next
```
See Also:

GetAuxEntityDefNames method
GetEntityDef method
GetQueryEntityDefNames method
GetReqEntityDefNames method
GetSubmitEntityDefNames method
EntityDef object
GetInstalledMasters method

Returns the list of registered database sets and master databases.

**VB Syntax:**

```vb
session.GetInstalledMasters dbSets, masterDBs
```

**Perl Syntax:**

```perl
$session->GetInstalledMasters(dbSets, masterDBs);
```

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>session</code></td>
<td>The Session object that represents the current database-access session.</td>
</tr>
<tr>
<td><code>dbSets</code></td>
<td>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.</td>
</tr>
<tr>
<td><code>masterDBs</code></td>
<td>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.</td>
</tr>
</tbody>
</table>

**Return value**

None.

**Member of:** Session object

**Remarks:**

The returned Variants always contain the same number of strings. The contents of both Variants are ordered so that each schema repository (master database) listed in `masterDBs` belongs to the database set at the same index in `dbSets`.

**Examples:**

```vb
set sessionObj = GetSession
Dim dbSets, masterDBs
sessionObj.GetInstalledMasters dbSets, masterDBs
For Each db in dbSets
```

```perl
```
See Also:

GetIsMaster method of the DatabaseDescription object
DatabaseDescription object
GetQueryEntityDefNames method

Returns an array containing the names of the record types that are suitable for use in queries.

**VB Syntax:**

```vbnet
session.GetQueryEntityDefNames
```

**Perl Syntax:**

```perl
$session->GetQueryEntityDefNames();
```

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>session</code></td>
<td>The Session object that represents the current database-access session.</td>
</tr>
</tbody>
</table>

**Return value**  A Variant containing an array of Strings. Each String contains the name of an EntityDef that can be used in a query.

**Member of:** Session object

**Remarks:**

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.

**Examples:**

```vbnet
set sessionObj = GetSession
'
Get the list of names of the record types that support queries.
entityDefNames = sessionObj.GetQueryEntityDefNames
'
Iterate over the state-based record types
for each name in entityDefNames
    set queryDefObj = sessionObj.BuildQuery(name)
```
Fill in the query parameters and run it.

See Also:

BuildQuery method
GetAuxEntityDefNames method
GetEntityDef method
GetEntityDefNames method
GetReqEntityDefNames method
GetSubmitEntityDefNames method
EntityDef object
GetReqEntityDefNames method

Returns an array containing the names of the state-based record types in the current database’s schema.

**VB Syntax:**

```vbnet
session.GetReqEntityDefNames
```

**Perl Syntax:**

```perl
$session->GetReqEntityDefNames();
```

### Identifier Description

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>session</td>
<td>The Session object that represents the current database-access session.</td>
</tr>
</tbody>
</table>

**Return value**

A Variant containing an array of Strings. Each string in the array contains the name of one of the desired record types.

**Member of:** Session object

**Remarks:**

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.

**Examples:**

```vbnet
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

See Also:

BuildQuery method
GetAuxEntityDefNames method
GetEntityDef method
GetEntityDefNames method
GetQueryEntityDefNames method
GetSubmitEntityDefNames method
EntityDef object
GetServerInfo method

Returns a string identifying the session’s OLE server.

**VB Syntax:**

`session.GetServerInfo`

**Perl Syntax:**

`$session->GetServerInfo();`

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>session</code></td>
<td>The Session object that represents the current database-access session.</td>
</tr>
</tbody>
</table>

**Return value**

A String identifying the OLE server.

**Member of:** Session object

**Remarks:**

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.

**Examples:**

```vbnet
set sessionObj = GetSession

serverName = sessionObj.GetServerInfo
```

**See Also:**

GetSessionDatabase method
GetSessionDatabase method

Returns information about the database that is being accessed in the current session.

**VB Syntax:**

`session.GetSessionDatabase`

**Perl Syntax:**

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

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>session</code></td>
<td>The Session object that represents the current database-access session.</td>
</tr>
<tr>
<td><strong>Return value</strong></td>
<td>A DatabaseDescription object that contains information about the current database.</td>
</tr>
</tbody>
</table>

**Member of:** Session object

**Remarks:**

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.

**Examples:**

```
set sessionObj = GetSession
set dbDescObj = sessionObj.GetSessionDatabase
```

**See Also:**

GetAccessibleDatabases method  
DatabaseDescription object  
Session and DatabaseDescription Code Example
**GetSubmitEntityDefNames method**

Returns an array containing the names of the record types that are suitable for use in creating a new record.

**VB Syntax:**

```vbnet
session.GetSubmitEntityDefNames
```

**Perl Syntax:**

```perl
$session->GetSubmitEntityDefNames();
```

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>session</code></td>
<td>The Session object that represents the current database-access session.</td>
</tr>
<tr>
<td><strong>Return value</strong></td>
<td>A Variant containing an array of Strings. Each string contains the name of one of the desired record types.</td>
</tr>
</tbody>
</table>

**Member of:** Session object

**Remarks:**

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.
Examples:

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

See Also:

GetAuxEntityDefNames method
GetEntityDef method
GetEntityDefNames method
GetQueryEntityDefNames method
GetReqEntityDefNames method
EntityDef object
**GetUserEmail method**

Returns the electronic mail address of the user who is logged in for this session.

**VB Syntax:**

`session.GetUserEmail`

**Perl Syntax:**

`
$session->GetUserEmail();
`

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>session</code></td>
<td>The Session object that represents the current database-access session.</td>
</tr>
<tr>
<td><strong>Return value</strong></td>
<td>A String containing the email address of the user who is logged in for this session.</td>
</tr>
</tbody>
</table>

**Member of:** Session object

**Remarks:**

If you have access to the schema repository, you can change the text of the user’s email address using the schema repository object User. Simply assign a new value to the Email property of User.

**Examples:**

```
set sessionObj = GetSession

' Get the user’s personal information
userName = sessionObj.GetUserFullName
userLogin = sessionObj.GetUserLoginName
userEmail = sessionObj.GetUserEmail
userPhone = sessionObj.GetUserPhone
userMisc = sessionObj.GetUserMiscInfo
```
See Also:

- GetUserFullName method
- GetUserGroups method
- GetUserLoginName method
- GetUserMiscInfo method
- GetUserPhone method
- Email property of the User object
- User object
- Session and DatabaseDescription Code Example
GetUser FullName method

Returns the full name of the user who is logged in for this session.

**VB Syntax:**

`session.GetUserFullName`

**Perl Syntax:**

`$session->GetUserFullName();`

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>session</code></td>
<td>The Session object that represents the current database-access session.</td>
</tr>
</tbody>
</table>

**Return value**

A String containing the full name (such as "Jenny Jones") of the user who is logged in for this session.

**Member of:** Session object

**Remarks:**

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.

**Examples:**

```vbnet
set sessionObj = GetSession

' Get the user’s personal information
userName = sessionObj.GetUserFullName
userLogin = sessionObj.GetUserLoginName
userEmail = sessionObj.GetUserEmail
userPhone = sessionObj.GetUserPhone
userMisc = sessionObj.GetUserMiscInfo
```
See Also:

- GetUserEmail method
- GetUserGroups method
- GetUserLoginName method
- GetUserMiscInfo method
- GetUserPhone method
- Fullname property of the User object
- User object
- Session and DatabaseDescription Code Example
GetUserGroups method

Returns a list of the groups to which the current user belongs.

VB Syntax:

`session.GetUserGroups`

Perl Syntax:

`

Member of: Session object

Remarks:

The returned variant can be empty.

If you have access to the schema repository, you can change the groups to which the user belongs using the Group object. To add a user to a group, call the AddUser method of Group.

Examples:

```
set sessionObj = GetSession

' Iterate over the user’s groups
userGroups = sessionObj.GetUserGroups
If IsEmpty(userGroups) = 0 Then
' Code to handle if no user groups exist
Else
For Each group in userGroups
    ...
Next
```
See Also:

GetUserEmail method
GetUserFullName method
GetUserLoginName method
GetUserMiscInfo method
GetUserPhone method
AddUser method of the Group object
Group object
Session and DatabaseDescription Code Example
**GetUserLoginName method**

Returns the name that was used to log in for this session.

**VB Syntax:**

```
session.GetUserLoginName
```

**Perl Syntax:**

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

**Identifier** | **Description**
--- | ---
`session` | The Session object that represents the current database-access session.

**Return value** | A String containing the login name (such as "jones") of the user who is logged in for this session.

**Member of:** Session object

**Remarks:**

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.

**Examples:**

```
set sessionObj = GetSession

' Get the user’s personal information
userName = sessionObj.GetUserFullName
userLogin = sessionObj.GetUserLoginName
userEmail = sessionObj.GetUserEmail
userPhone = sessionObj.GetUserPhone
userMisc = sessionObj.GetUserMiscInfo
```
See Also:

GetUserEmail method
GetUserFullName method
GetUserGroups method
GetUserMiscInfo method
GetUserPhone method
AdminSession object
User object
Session and DatabaseDescription Code Example
GetUserMiscInfo method

Returns miscellaneous information about the user who is logged in for this session.

**VB Syntax:**

```vbnet
session.GetUserMiscInfo
```

**Perl Syntax:**

```perl
$session->GetUserMiscInfo();
```

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>session</code></td>
<td>The Session object that represents the current database-access session.</td>
</tr>
</tbody>
</table>

**Return value** A String containing any miscellaneous information about the user.

**Member of:** Session object

**Remarks:**

Miscellaneous information is any information that has been entered by the administrator into that user’s profile. Information about the user’s login name, full name, email address, phone number, and groups is stored separately and can be retrieved by the corresponding Session methods.

If you have access to the schema repository, you can change the text of the miscellaneous information using the schema repository object User. Simply assign a new value to the MiscInfo property of User.

**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
```
See Also:
GetUserEmail method
GetUserFullName method
GetUserGroups method
GetUserLoginName method
GetUserPhone method
MiscInfo property of the User object
User object
Session and DatabaseDescription Code Example
GetUserPhone method

Returns the telephone number of the user who is logged in for this session.

**VB Syntax:**

```vbnet
session.GetUserPhone
```

**Perl Syntax:**

```perl
$session->GetUserPhone();
```

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>session</td>
<td>The Session object that represents the current database-access session.</td>
</tr>
<tr>
<td>Return value</td>
<td>A String containing the telephone number (if known) of the user who is logged in for this session.</td>
</tr>
</tbody>
</table>

**Member of:** Session object

**Remarks:**

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.

**Examples:**

```vbnet
set sessionObj = GetSession

' Get the user’s personal information
userName = sessionObj.GetUserFullName
userLogin = sessionObj.GetUserLoginName
userEmail = sessionObj.GetUserEmail
userPhone = sessionObj.GetUserPhone
userMisc = sessionObj.GetUserMiscInfo
```
See Also:

- GetUserEmail method
- GetUserFullName method
- GetUserGroups method
- GetUserLoginName method
- GetUserMiscInfo method
- Phone property of the User object
- User object
- Session and DatabaseDescription Code Example
GetWorkSpace method

Returns the session’s WORKSPACE object.

**VB Syntax:**

```
session.GetWorkSpace
```

**Perl Syntax:**

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

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>session</code></td>
<td>The Session object that represents the current database-access session.</td>
</tr>
</tbody>
</table>

**Return value**

The WORKSPACE object belonging to the current session.

**Member of:** Session object

**Remarks:**

You can use the WORKSPACE object to manipulate saved queries, charts, and reports in the ClearQuest workspace.

**Examples:**

```vbscript
set sessionObj = GetSession

' Get the workspace for manipulating query, chart, and report info.
wkSpc = sessionObj.GetWorkspace
```

**See Also:**

WORKSPACE object
HasValue method

Returns a Bool indicating whether the specified session variable exists.

VB Syntax:

```
session.HasValue name
```

Perl Syntax:

```
/session->HasValue(name);
```

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>session</td>
<td>The Session object that represents the current database-access session.</td>
</tr>
<tr>
<td>name</td>
<td>A String containing the name of the session variable.</td>
</tr>
<tr>
<td>Return value</td>
<td>True if the variable exists in this session, otherwise False.</td>
</tr>
</tbody>
</table>

Member of: Session object

Remarks:

Session variables persist until the Session object is deleted. To get or set variables, use the NameValue method.

Examples:

```
set sessionObj = GetSession
If HasValue("foo") Then
    fooValue = sessionObj.NameValue("foo")
End If
```

See Also:

NameValue property
IsMetadataReadonly method

Returns a Bool indicating whether the session’s metadata is read-only.

VB Syntax:

`session.IsMetadataReadonly`

Perl Syntax:

`
$session->IsMetadataReadonly();
``

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>session</code></td>
<td>The Session object that represents the current database-access session.</td>
</tr>
</tbody>
</table>

**Return value**

True if the metadata is read-only, otherwise False.

**Member of:** Session object

**Examples:**

```vbnet
set sessionObj = GetSession

If sessionObj.IsMetadataReadonly Then
    ...
End If
```

**See Also:**

EntityDef object
MarkEntityAsDuplicate method

Modifies the specified record to indicate that it is a duplicate of another record.

**VB Syntax:**

```vbnet
session.MarkEntityAsDuplicate duplicate, original, duplicate_action_name
```

**Perl Syntax:**

```perl
$session->MarkEntityAsDuplicate(duplicate, original, duplicate_action_name);
```

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>session</code></td>
<td>The Session object that represents the current database-access session.</td>
</tr>
<tr>
<td><code>duplicate</code></td>
<td>The Entity object that is to be marked as a duplicate of original.</td>
</tr>
<tr>
<td><code>original</code></td>
<td>The Entity object that is the original data record.</td>
</tr>
<tr>
<td><code>duplicate_action_name</code></td>
<td>A String that specifies an action whose ActionType is DUPLICATE. This parameter must identify a valid action for the duplicate record.</td>
</tr>
</tbody>
</table>

**Return value**

None.

**Member of:** Session object

**Remarks:**

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.
Examples:

```
set sessionObj = GetSession

' Mark the entity with ID="BUGID00010345" as a duplicate of this entity.
' Use the action named "duplicate".
set dupEntityObj = sessionObj.GetEntity("defect", "BUGID00010345")
sessionObj.MarkEntityAsDuplicate dupEntityObj, entity, "duplicate"

' Validate and commit the duplicate entity since it
' is currently modifiable.
error = dupEntityObj.Validate
if error = "" then
   dupEntityObj.Commit
End If
```

See Also:

UnmarkEntityAsDuplicate method
Notation Conventions
OpenQueryDef method

 Loads a query from a file.

 **VB Syntax:**

 `session.OpenQueryDef filename`

 **Perl Syntax:**

 `$session->OpenQueryDef(filename);`

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>session</code></td>
<td>The Session object that represents the current database-access session.</td>
</tr>
<tr>
<td><code>filename</code></td>
<td>The name of the file from which to load the query information.</td>
</tr>
<tr>
<td><strong>Return value</strong></td>
<td>A QueryDef object containing the query information.</td>
</tr>
</tbody>
</table>

**Member of:** Session object

**Remarks:**

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.

**Examples:**

```vb
set sessionObj = GetSession

' Get the query from file "C:\queries\myQuery.txt"
set queryDefObj = sessionObj.OpenQueryDef("C:\queries\myQuery.txt")
```

**See Also:**

QueryDef object
OutputDebugString method

Specifies a message that can be displayed by a debugger or a similar tool.

**VB Syntax:**

```
session.OutputDebugString debugString
```

**Perl Syntax:**

```
$session->OutputDebugString($debugString);
```

### Identifier Description

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>session</code></td>
<td>The Session object that represents the current database-access session.</td>
</tr>
<tr>
<td><code>debugString</code></td>
<td>A String containing the text to be displayed.</td>
</tr>
<tr>
<td><code>Return value</code></td>
<td>None.</td>
</tr>
</tbody>
</table>

**Member of:** Session object

**Remarks:**

The value of `debugString` 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.

**Examples:**

```vbnet
set sessionObj = GetSession
sessionObj.OutputDebugString "This is a test message."
```

**See Also:**

UnmarkEntityAsDuplicate method
UnmarkEntityAsDuplicate method

Removes the indication that the specified record is a duplicate of another record.

**VB Syntax:**

```vbnet
session.UnmarkEntityAsDuplicate duplicate, action_name
```

**Perl Syntax:**

```perl
/session->UnmarkEntityAsDuplicate(duplicate, action_name);
```

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>session</td>
<td>The Session object that represents the current database-access session.</td>
</tr>
<tr>
<td>duplicate</td>
<td>The Entity object (currently marked as a duplicate) that is to be modified.</td>
</tr>
<tr>
<td>action_name</td>
<td>A String that specifies the action to be performed on the duplicate.</td>
</tr>
<tr>
<td></td>
<td>This parameter must contain the name of a valid action as defined in the</td>
</tr>
<tr>
<td></td>
<td>schema. Such an action must have the ActionType UNDUPLICATE.</td>
</tr>
</tbody>
</table>

**Return value**

None.

**Member of:** Session object

**Remarks:**

This method breaks the linkage between a duplicate and original Entity object. You can call this method to break a link that was established by the user or by calling the MarkEntityAsDuplicate method. If the DUPLICATE action being undone caused a state transition, that transition is undone unless a subsequent state transition occurred after the DUPLICATE action. After this method returns, the record is editable and must be validated and committed using the Entity object’s Validate method and Commit method, respectively.
**Examples:**

```vbnet
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
```

**See Also:**

- MarkEntityAsDuplicate method
- Validate method of the Entity object
- Entity object
- Notation Conventions
UserLogon method

Log in as the specified user for a database session.

VB Syntax:

```vbnet
session.UserLogon login_name, password, database_name, session_type, database_set
```

Perl Syntax:

```perl
$sessioon->UserLogon(login_name, password, database_name, session_type, database_set);
```

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>login_name</td>
<td>A String that specifies the login name of the user.</td>
</tr>
<tr>
<td>password</td>
<td>A String that specifies the user’s password.</td>
</tr>
<tr>
<td>database_name</td>
<td>A String that specifies the name of the desired user database. (You must not login to the master database using this method.)</td>
</tr>
<tr>
<td>session_type</td>
<td>A Long whose value is a SessionType constant specifying whether the session is shared (SHARED_SESSION) or private (PRIVATE_SESSION). Data from a shared session can be accessed by more than one client at a time. (ADMIN_SESSION is not permitted.)</td>
</tr>
<tr>
<td>database_set</td>
<td>A String that specifies the name of the master database. You should set this string to the empty string (&quot;&quot;).</td>
</tr>
</tbody>
</table>

**Return value**: None.

**Member of**: Session object

**Remarks**:

Before calling this method, you should have already created and initialized a new Session object. No other Session methods should be invoked before UserLogon, with
the exception of the GetAccessibleDatabases method, OutputDebugString method, and UnmarkEntityAsDuplicate method.

If you are writing hook code, you should not need to call this method. ClearQuest creates the Session object for you and logs the user in before calling any hooks.

**Examples:**

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

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

' Login to each database successively.
set databases = sessionObj.GetAccessibleDatabases
For Each db in databases
    dbName = db.GetDatabaseName
    sessionObj.UserLogon "joe", "gh36ak3", dbName, AD_PRIVATE_SESSION, ""
    ' Access the database
    ...
Next
```

**See Also:**

GetDatabaseConnectString method of the DatabaseDescription object
DatabaseDescription object
Session and DatabaseDescription Code Example
Notation Conventions
## Entity object properties

<table>
<thead>
<tr>
<th>Property name</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AttachmentFields property</td>
<td>Read-only</td>
<td>Returns the AttachmentFields collection object containing this Entity object's attachment fields.</td>
</tr>
<tr>
<td>HistoryFields property</td>
<td>Read-only</td>
<td>Returns the HistoryFields collection object containing this Entity object's history fields.</td>
</tr>
</tbody>
</table>

## Entity object methods

<table>
<thead>
<tr>
<th>Method name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddFieldValue method</td>
<td>Adds the specified value to the list of values in the named field.</td>
</tr>
<tr>
<td>BeginNewFieldUpdateGroup method</td>
<td>Marks the beginning of a series of SetFieldValue calls.</td>
</tr>
<tr>
<td>Commit method</td>
<td>Updates the database with the changes made to the Entity object.</td>
</tr>
<tr>
<td>DeleteFieldValue method</td>
<td>Removes the specified value from the field's list of values.</td>
</tr>
<tr>
<td>FireNamedHook method</td>
<td>Executes a named hook of this record's EntityDef object.</td>
</tr>
<tr>
<td>GetActionName</td>
<td>Returns the name of the action associated with the current Entity object.</td>
</tr>
<tr>
<td>GetActionType</td>
<td>Returns the type of the action associated with the current Entity object.</td>
</tr>
<tr>
<td>GetAllDuplicates method</td>
<td>Returns links to all of the duplicates of this Entity, including duplicates of duplicates.</td>
</tr>
<tr>
<td>GetAllFieldValues method</td>
<td>Returns an array of FieldInfo objects corresponding to all of the Entity object’s fields.</td>
</tr>
<tr>
<td>GetDbId method</td>
<td>Returns the Entity object’s database ID number.</td>
</tr>
<tr>
<td>GetDisplayName method</td>
<td>Returns the unique key associated with the Entity.</td>
</tr>
<tr>
<td>GetDuplicates method</td>
<td>Returns links to the immediate duplicates of this object.</td>
</tr>
<tr>
<td>GetEntityDefName method</td>
<td>Returns the name of the EntityDef object that serves as a template for this object.</td>
</tr>
<tr>
<td>Method name</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>GetFieldChoiceList method</td>
<td>Returns the list of permissible values for the specified field.</td>
</tr>
<tr>
<td>GetFieldChoiceType method</td>
<td>Returns the type of the given choice-list field.</td>
</tr>
<tr>
<td>GetFieldMaxLength method</td>
<td>Returns the maximum number of characters allowed for the specified string field.</td>
</tr>
<tr>
<td>GetFieldNames method</td>
<td>Returns the names of the fields in the Entity object.</td>
</tr>
<tr>
<td>GetFieldOriginalValue method</td>
<td>Returns the FieldInfo containing the value that the specified field will revert to, if the action is cancelled.</td>
</tr>
<tr>
<td>GetFieldRequiredness method</td>
<td>Identifies the behavior of the specified field.</td>
</tr>
<tr>
<td>GetFieldsUpdatedThisAction method</td>
<td>Returns a FieldInfo object for each field that was modified by the most recent action.</td>
</tr>
<tr>
<td>GetFieldsUpdatedThisGroup method</td>
<td>Returns a FieldInfo object for each field that was modified since the most recent call to BeginNewFieldUpdateGroup method.</td>
</tr>
<tr>
<td>GetFieldsUpdatedThisSetValue method</td>
<td>Returns a FieldInfo object for each of the Entity’s fields that was modified by the most recent SetFieldValue call.</td>
</tr>
<tr>
<td>GetFieldType method</td>
<td>Identifies the type of data that can be stored in the specified field.</td>
</tr>
<tr>
<td>GetFieldValue method</td>
<td>Returns the FieldInfo object for the specified field.</td>
</tr>
<tr>
<td>GetInvalidFieldValues method</td>
<td>Returns an array of FieldInfo objects corresponding to all the Entity’s invalid fields.</td>
</tr>
<tr>
<td>GetLegalActionDefNames method</td>
<td>Returns the names of the actions that can be used on this Entity object.</td>
</tr>
<tr>
<td>GetOriginal method</td>
<td>Returns the Entity object that is marked as the original of this duplicate object.</td>
</tr>
<tr>
<td>GetOriginalID method</td>
<td>Returns the visible ID of this object’s original Entity object.</td>
</tr>
<tr>
<td>GetSession method</td>
<td>Returns the current Session object.</td>
</tr>
<tr>
<td>GetType method</td>
<td>Returns the type (state-based or stateless) of the Entity.</td>
</tr>
<tr>
<td>HasDuplicates method</td>
<td>Reports whether this object is the original of one or more duplicates.</td>
</tr>
<tr>
<td>InvalidateFieldChoiceList function</td>
<td>Use with SetFieldChoiceList function to refresh values in a choice list.</td>
</tr>
</tbody>
</table>
### Method Name

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

**See Also:**

- `BuildEntity` method of the Session object
- `EditEntity` method of the Session object
- `GetEntity` method of the Session object
- `GetEntityById` method of the Session object
- `EntityDef` object
- `QueryDef` object
- `ResultSet` Object
- `Session` object
AttachmentFields property

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

VB Syntax:

[entity.]AttachmentFields

Perl Syntax:

$entity->GetAttachmentFields();

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entity</td>
<td>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.</td>
</tr>
<tr>
<td>Return value</td>
<td>An AttachmentFields collection object that contains all of the AttachmentField objects currently associated with this Entity object.</td>
</tr>
</tbody>
</table>

Member of: Entity object

Remarks:

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

For an overview of attachments, see Attachment-Related Objects.

Example:

set fields = entity.AttachmentFields
For Each fieldObj in fields
  ’ Do something with each AttachmentField object
See Also:

Attachment object
AttachmentField object
AttachmentFields collection object
Attachments collection object
Attachments Code Example
HistoryFields property

Returns the HistoryFields collection object containing this Entity object’s history fields.

**VB Syntax:**

`[entity.]HistoryFields`

**Perl Syntax:**

`$entity->GetHistoryFields();`

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>entity</code></td>
<td>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.</td>
</tr>
<tr>
<td>Return value</td>
<td>A HistoryFields collection object that contains all the individual HistoryField objects currently associated with this Entity object.</td>
</tr>
</tbody>
</table>

**Member of:** Entity object

**Remarks:**

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

**Example:**

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

**See Also:**

Histories collection object
History object
HistoryField object
HistoryFields collection object
AddFieldValue method

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

**VB Syntax:**

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

**Perl Syntax:**

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

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entity</td>
<td>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.</td>
</tr>
<tr>
<td>field_name</td>
<td>A String containing a valid field name of this Entity object.</td>
</tr>
<tr>
<td>new_value</td>
<td>A Variant containing the new value to add to the field.</td>
</tr>
</tbody>
</table>

**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.

**Member of:** Entity object

**Remarks:**

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

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

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

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

See Also:

DeleteFieldValue method
GetFieldValue method
SetFieldValue method
ValidityChangedThisSetValue method
ValueChangedThisSetValue method
EditEntity method of the Session object
FieldInfo object
BeginNewFieldUpdateGroup method

Marks the beginning of a series of SetFieldValue calls.

**VB Syntax:**

```
[entity.]BeginNewFieldUpdateGroup
```

**Perl Syntax:**

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

**Identifier** | **Description**
---|---
`entity` | An Entity object representing a user data record. Inside a hook, if you omit this part of the syntax, the Entity object corresponding to the current data record is assumed (VBScript only).

**Return value** | None.

**Member of:** Entity object

**Remarks:**

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

**Examples:**

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

' Iterate over all the fields that changed
For Each field In updatedFields
```
See Also:

GetFieldsUpdatedThisAction method
GetFieldsUpdatedThisGroup method
GetFieldsUpdatedThisSetValue method
SetFieldValue method
ValidityChangedThisSetValue method of the FieldInfo object
FieldInfo object
Commit method

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

**VBScript Syntax:**

\[ entity\]Commit

**Perl Syntax:**

$entity->Commit();

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entity</td>
<td>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).</td>
</tr>
</tbody>
</table>

**Return value**

None.

**Member of:** Entity object

**Remarks:**

This method applies 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.

**Examples:**

' Modify the record and then commit the changes.
set sessionObj = GetSession
set entityObj = sessionObj.GetEntity("defect", "BUGID000000042")
sessionObj.EditEntity entityObj, "modify"
... ' modify the Entity object
entityObj.Validate
entityObj.Commit

' The Entity object is no longer editable

**See Also:**
IsEditable method
Revert method
Validate method
BuildEntity method of the Session object
EditEntity method of the Session object
Session object
Duplicates Code Example
DeleteFieldValue method

Removes the specified value from the field’s list of values.

**VBScript Syntax:**

```
[entity.]DeleteFieldValue field_name, old_value
```

**Perl Syntax:**

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

**Identifier Description**

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entity</td>
<td>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).</td>
</tr>
<tr>
<td>field_name</td>
<td>A String containing a valid field name of this Entity object.</td>
</tr>
<tr>
<td>old_value</td>
<td>A Variant containing the value to remove from the field’s list of values.</td>
</tr>
</tbody>
</table>

**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.

**Member of:** Entity object

**Remarks:**

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.

**Examples:**

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

See Also:
AddFieldValue method
GetFieldValue method
SetFieldValue method
ValidityChangedThisSetValue method
ValueChangedThisSetValue method
EditEntity method of the Session object
FieldInfo object
**FireNamedHook method**

Executes a named hook of this record’s EntityDef object.

**VBScript Syntax:**

```vb
[entity]FireNamedHook hookName, parameters
```

**Perl Syntax:**

```perl
$entity->FireNamedHook(hookName, parameters);
```

**Identifier** | **Description**
--- | ---
entiy | An Entity object representing a user data record. Inside a hook, if you omit this part of the syntax, the Entity object corresponding to the current data record is assumed (VBScript only).
hookName | A String containing the name of the hook to execute.
parameter(s) | [A VB Variant or] a Perl string containing the parameters you want to pass to the hook.

**Return value**

A String indicating the status of calling the hook. If the hook executed successfully, this method returns an empty string (""), otherwise the returned string contains a description of the error.

**Member of:** Entity object

**Remarks:**

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

```vb
Function EntityDefName_HookName(parameters)
    ' parameter as Variant
    ' EntityDefName_HookName as Variant

    ' Hook program body
End Function
```
You cannot use this method to execute a field or action hook of a record. You also cannot execute a global hook, except indirectly from the record hook.

You can call this method on an Entity object regardless of whether or not it is editable. However, if your hook attempts to modify the Entity object, either your code or the hook code must first call EditEntity method to make the Entity object editable.

If your hook accepts any parameters, put all of the parameters in a single Variant and specify that Variant in parameters. The hook must be able to interpret the parameters passed into it. Upon return, the hook can similarly return a Variant with any appropriate return values.

**Example:**

' Execute the hook "MyHook" with the specified parameters
Dim params(1)
params(0) = "option 1"
params(1) = "option 2"

returnValue = entity.FireNamedHook("MyHook", params)

**See Also:**

EditEntity method of the Session object
GetHookDefNames method of the EntityDef object
EntityDef object
GetActionName

Returns the name of the current action associated with the current entity.

**VBScript Syntax:**

```vbnet
entity.GetActionName
```

**Perl Syntax:**

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

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>entity</code></td>
<td>An Entity object corresponding to a record in a schema.</td>
</tr>
<tr>
<td><code>Return value</code></td>
<td>A String whose value provides the name of ActionType constant _GETACTIONNAME.</td>
</tr>
</tbody>
</table>

**Member of:** Entity object

**Remarks:**

Used in base action hooks.

**See Also:**

GetActionType
ActionType
GetActionType

Returns the type of the current action associated with the current entity.

**VBScript Syntax:**

```
entitydef.GetActionType
```

**Perl Syntax:**

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

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entity</td>
<td>An Entity object corresponding to a record in a schema.</td>
</tr>
<tr>
<td>Return value</td>
<td>A String whose value is the ActionType constant _GETACTIONNAME.</td>
</tr>
</tbody>
</table>

**Member of:** Entity object

**Remarks:**

Used in base action hooks.

**See Also:**

GetActionName
ActionType
GetAllDuplicates method

Returns links to all of the duplicates of this Entity, including duplicates of duplicates.

**VBScript Syntax:**

```vbnet
[entity]GetAllDuplicates
```

**Perl Syntax:**

```perl
$entity->GetAllDuplicates();
```

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entity</td>
<td>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).</td>
</tr>
</tbody>
</table>

**Return value**

A Variant containing an Array of Link objects. If this object has no duplicates, the return value is an Empty Variant.

**Member of:** Entity object

**Remarks:**

This method returns all duplicates, including duplicates of duplicates. To obtain only the immediate duplicates of an object, call the GetDuplicates method instead.

**Examples:**

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:

```plaintext
linkObjs = entity1.GetAllDuplicates
```

The `linkObjs` variable would be an array of 3 Link objects:

- a link between `entity1` and `entity2`
- a link between `entity1` and `entity3`
- a link between `entity3` and `entity4`
See Also:
GetDuplicates method
GetOriginal method
GetOriginalID method
HasDuplicates method
IsDuplicate method
IsOriginal method
MarkEntityAsDuplicate method of the Session object
UnmarkEntityAsDuplicate method of the Session object
Link object
Session object
Duplicates Code Example
GetAllFieldValues method

Returns an array of FieldInfo objects corresponding to all of the Entity object’s fields.

**VBScript Syntax:**

```
[entity.]GetAllFieldValues
```

**Perl Syntax:**

```
$entity-&gt;GetAllFieldValues();
```

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entity</td>
<td>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).</td>
</tr>
</tbody>
</table>

**Return value**

A Variant containing an Array of FieldInfo objects, one for each field in the Entity object.

**Member of:** Entity object

**Remarks:**

The FieldInfo objects are arranged in no particular order.

**Examples:**

```
' 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
```

**See Also:**

GetFieldValue method
GetInvalidFieldValues method
FieldInfo object
GetDbId method

Returns the Entity object’s database ID number.

**VBScript Syntax:**

```plaintext
[entity:]GetDbId
```

**Perl Syntax:**

```plaintext
$entity->GetDbId();
```

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>entity</code></td>
<td>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).</td>
</tr>
</tbody>
</table>

**Return value**

A Long containing the Entity object's database ID.

**Member of:** Entity object

**Remarks:**

The return value is a database ID. This value is used internally by the database to keep track of records. Do not confuse this value with the defect ID number returned by the GetDisplayName method.

**Examples:**

```plaintext
dbID = entity.GetDbId
```

**See Also:**

GetDisplayName method

Entity Code Example
GetDefaultActionName

Returns the default action name associated with the current state.

VBScript Syntax:

```
[entity.]GetDefaultActionName
```

Perl Syntax:

```
$entity-&gt;GetDefaultActionName();
```

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>entity</code></td>
<td>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).</td>
</tr>
</tbody>
</table>

**Return value**

A String that returns the default action name associated with the current state.

**Member of:** Entity object

**Remarks:**

This method allows you to programmatically move a defect through the default actions (state transition matrix) set in ClearQuest Designer.

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

**Examples:**

```
DefaultActionName = entity.GetDefaultActionName
'COMPLETE EXAMPLE FOR POST BETA.
```

**See Also:**

GetActionDestStateName method
GetDisplayName method

Returns the unique key associated with the Entity.

**VBScript Syntax:**

```vb
[entity]GetDisplayName
```

**Perl Syntax:**

```perl
$entity->GetDisplayName();
```

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entity</td>
<td>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).</td>
</tr>
</tbody>
</table>

**Return value**

A String containing the record type's unique key.

**Member of:** Entity object

**Remarks:**

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.
Examples:

' Get the record ID using 2 different techniques and compare the results
displayName = GetDisplay Name
idName = GetFieldValue("id").GetValue
If idName <> displayName Then
' Error, these id numbers should match
End If

See Also:
GetDbId method
GetFieldValue method
GetValue method of the FieldInfo object
FieldInfo object
Duplicates Code Example
GetDuplicates method

Returns links to the immediate duplicates of this object.

**VBScript Syntax:**

```
[entity.]GetDuplicates
```

**Perl Syntax:**

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

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entity</td>
<td>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).</td>
</tr>
</tbody>
</table>

**Return value**

A Variant containing an Array of Link objects. Each Link object points to a duplicate of this object. If this object has no duplicates, the return value is an Empty Variant.

**Member of:** Entity object

**Remarks:**

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.

**Examples:**

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:

```plaintext
linkObjs = entity1.GetDuplicates
```

The `linkObjs` variable would be an array of 2 Link objects:

- a link between entity1 and entity2
• a link between entity1 and entity3

**See Also:**

GetAllDuplicates method
GetOriginal method
GetOriginalID method
HasDuplicates method
IsDuplicate method
IsOriginal method
MarkEntityAsDuplicate method of the Session object
UnmarkEntityAsDuplicate method of the Session object
Session object
Duplicates Code Example
GetEntityDefName method

Returns the name of the EntityDef object that is the template for this object.

**VBScript Syntax:**

```
[entity] . GetEntityDefName
```

**Perl Syntax:**

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

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entity</td>
<td>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).</td>
</tr>
</tbody>
</table>

**Return value**

A String containing the name of the EntityDef object upon which this object is based.

**Member of:** Entity object

**Remarks:**

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.

**Examples:**

```vbnet
set sessionObj = GetSession

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

GetEntityDef method
EntityDef object
GetFieldChoiceList method

Returns the list of permissible values for the specified field.

**VBScript Syntax:**

```
[entity.]GetFieldChoiceList field_name
```

**Perl Syntax:**

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

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entity</td>
<td>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).</td>
</tr>
<tr>
<td>field_name</td>
<td>A String that identifies a valid field name of entity.</td>
</tr>
<tr>
<td>Return value</td>
<td>A Variant containing an Array. 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.</td>
</tr>
</tbody>
</table>

**Member of:** Entity object

**Remarks:**

The administrator specifies whether the legal values for a given field are restricted to the contents of the choice list. If there is a restriction, specifying a value not in the choice list causes a validation error. If there is no restriction, you may specify values not in the choice list. (Note that any values you specify must still be validated.)

If this method returns an Empty Variant, it does not imply that all values are permitted; it just means that the administrator has not provided any hints about the values permitted in the field.

If the administrator chose to use a hook to determine the values of the choice list, ClearQuest will have already executed the hook and cached the resulting values in a HookChoices object. You can use that object to retrieve the values.
You can use the GetFieldNames method to obtain a list of valid names for the field_name parameter.

**Note:** When calling this method from an external Visual Basic program, this method throws an exception if entity is not editable.

**Examples:**

```vbnet
fieldValue = GetFieldNames("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
```

**See Also:**

GetFieldChoiceType method
GetFieldNames method
HookChoices object
GetFieldChoiceType method

Returns the type of the given choice-list field.

**VBScript Syntax:**

```
[entity.]GetFieldChoiceType field_name
```

**Perl Syntax:**

```
$entity->[GetFieldChoiceType](field_name);
```

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entity</td>
<td>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).</td>
</tr>
<tr>
<td>field_name</td>
<td>A String that identifies a valid field name of entity.</td>
</tr>
<tr>
<td>Return value</td>
<td>A Long indicating the type of the field.</td>
</tr>
</tbody>
</table>

**Member of:** Entity object

**Remarks:**

The return value is either CLOSED_CHOICE or OPEN_CHOICE. If the return value is CLOSED_CHOICE, the valid values for the field are limited to those specified in the choice list. If the return value is OPEN_CHOICE, the user may select an item from the choice list or type in a new value.

**Examples:**

```
' If the field must have a value from a closed choice list, assign the ' first 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
```
See Also:

getFieldChoiceList method
getFieldNames method
HookChoices object
Notation Conventions
GetFieldMaxLength method

Returns the maximum number of characters allowed for the specified string field.

**VBScript Syntax:**

```
[entity.]GetFieldMaxLength field_name
```

**Perl Syntax:**

```
$entity->[GetFieldMaxLength](field_name);
```

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>entity</code></td>
<td>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).</td>
</tr>
<tr>
<td><code>field_name</code></td>
<td>A String that identifies a valid field name of entity. The field must contain a fixed-length string.</td>
</tr>
<tr>
<td><code>Return value</code></td>
<td>A Long indicating the maximum number of characters the field can store.</td>
</tr>
</tbody>
</table>

**Member of:** Entity object

**Remarks:**

This method is relevant only for fields whose type is SHORT_STRING.

**Examples:**

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

**See Also:**

GetFieldType method
Field
Notation Conventions
GetFieldNames method

Returns the names of the fields in the Entity object.

**VBScript Syntax:**

```
[entity.]GetFieldNames field_name
```

**Perl Syntax:**

```
$entity->GetFieldNames(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).

**Return value** | A Variant containing an Array whose elements are Strings. Each String contains the name of one field.

**Member of:** Entity object

**Remarks:**

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.

**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="
```
Next

**See Also:**

GetFieldChoiceList method
GetFieldDefNames method
GetFieldRequiredness method
GetFieldType method
GetFieldValue method
Notification Hook Code Example
GetFieldOriginalValue method

Returns the FieldInfo containing the value that the specified field will revert to, if the action is cancelled.

**VBScript Syntax:**

```
[entity.]GetFieldOriginalValue field_name
```

**Perl Syntax:**

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

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entity</td>
<td>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).</td>
</tr>
<tr>
<td>field_name</td>
<td>A String containing a valid field name of this Entity object.</td>
</tr>
<tr>
<td>Return value</td>
<td>A FieldInfo object that contains the original value for the specified field.</td>
</tr>
</tbody>
</table>

**Member of:** Entity object

**Remarks:**

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.

**Examples:**

````
' Iterate through the fields and report which ones have changed.
fieldNameList = GetFieldNames
For Each fieldName in fieldNameList
    originalValue = GetFieldOriginalValue(fieldName).GetValue
    currentValue = GetFieldValue(fieldName).GetValue
    If currentValue <> originalValue Then
```
' Report a change in the field value
  OutputDebugString "The value in field " & fieldName & " has changed."
End If
Next

See Also:

GetFieldValue method
FieldInfo object
Notification Hook Code Example
GetFieldRequiredness method

Identifies the behavior of the specified field.

**VBScript Syntax:**

`[entity]GetFieldRequiredness field_name`

**Perl Syntax:**

`
$entity->GetFieldRequiredness(field_name);
`

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entity</td>
<td>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).</td>
</tr>
<tr>
<td>field_name</td>
<td>A String that identifies a valid field name of entity.</td>
</tr>
<tr>
<td>Return value</td>
<td>A Long that identifies the behavior of the named field. The value corresponds to one of the Behavior enumeration constants.</td>
</tr>
</tbody>
</table>

**Member of:** Entity object

**Remarks:**

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.
Examples:

' 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

See Also:

GetFieldNames method
GetRequiredness method of the FieldInfo object
FieldInfo object
Notation Conventions
GetFieldsUpdatedThisAction method

Returns a FieldInfo object for each field that was modified by the most recent action.

**VBScript Syntax:**

```vb
[entity]GetFieldsUpdatedThisAction field_name
```

**Perl Syntax:**

```perl
$entity->GetFieldsUpdatedThisAction();
```

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entity</td>
<td>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).</td>
</tr>
</tbody>
</table>

**Return value**

A Variant containing an Array of FieldInfo objects. 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.

**Member of:** Entity object

**Remarks:**

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.
Examples:

```vbnet
set sessionObj = GetSession

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

See Also:

- BeginNewFieldUpdateGroup method
- GetFieldsUpdatedThisAction method
- GetFieldsUpdatedThisSetValue method
- SetFieldValue method
- ValidityChangedThisSetValue method of the FieldInfo object
- FieldInfo object
GetFieldsUpdatedThisGroup method

Returns a FieldInfo object for each field that was modified since the most recent call to BeginNewFieldUpdateGroup method.

**VBScript Syntax:**

```
[entity.]GetFieldsUpdatedThisGroup
```

**Perl Syntax:**

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

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entity</td>
<td>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).</td>
</tr>
</tbody>
</table>

**Return value**

A Variant containing an Array of FieldInfo objects. 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.

**Member of:** Entity object

**Remarks:**

Use this method to mark the end of a group of calls to SetFieldValue method (You must have previously called BeginNewFieldUpdateGroup method to mark the beginning of the group.) This technique is useful for web-based systems where you might need to track any changes to the fields in a form. For example, if the user moves to another web page, you can call this method to save the current state of the form and restore it when the user returns to that page.

**Examples:**

```
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:**

BeginNewFieldUpdateGroup method
GetFieldsUpdatedThisAction method
GetFieldsUpdatedThisSetValue method
SetFieldValue method
ValidityChangedThisSetValue method of the FieldInfo object
FieldInfo object
GetFieldsUpdatedThisSetValue method

Returns a FieldInfo object for each of the Entity’s fields that was modified by the most recent SetFieldValue call.

**VBScript Syntax:**

```
[entity.]GetFieldsUpdatedThisSetValue
```

**Perl Syntax:**

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

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entity</td>
<td>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).</td>
</tr>
</tbody>
</table>

**Return value**

A Variant containing an Array of FieldInfo objects, 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.

**Member of:** Entity object

**Remarks:**

This method usually returns a single FieldInfo object for the field that was modified by SetFieldValue method. However, this method can return multiple FieldInfo objects if other fields are dependent on the field that was changed. In such a case, hook code could automatically modify the value of any dependent fields, causing them to be modified as well and thus reported by this method.

**Examples:**

```vbnet
SetFieldValue "field1" "100"
modifiedFields = GetFieldsUpdatedThisSetValue
numFields = UBound(modifiedFields) + 1
If numFields > 1 Then
  OutputDebugString "Changing field1 resulted in changes to "
```
End If

**See Also:**

BeginNewFieldUpdateGroup method
GetFieldsUpdatedThisAction method
GetFieldsUpdatedThisGroup method
SetFieldValue method
ValidityChangedThisSetValue method of the FieldInfo object
FieldInfo object
GetFieldType method

Identifies the type of data that can be stored in the specified field.

**VBScript Syntax:**

```vbscript
[entity]GetFieldType field_name
```

**Perl Syntax:**

```perl
$entity->GetFieldType(field_name);
```

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>entity</code></td>
<td>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).</td>
</tr>
<tr>
<td><code>field_name</code></td>
<td>A String that identifies a valid field name of entity.</td>
</tr>
<tr>
<td><code>Return value</code></td>
<td>A Long that identifies what type of data can be stored in the named field. The value corresponds to one of the FieldType enumeration constants.</td>
</tr>
</tbody>
</table>

**Member of:** Entity object

**Remarks:**

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 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.
Examples:

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

See Also:

GetFieldNames method
GetType method of the FieldInfo object
FieldInfo object
GetFieldValue method

Returns the FieldInfo object for the specified field.

**VBScript Syntax:**

```
[entity.]GetFieldValue field_name
```

**Perl Syntax:**

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

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entity</td>
<td>An Entity object representing a user data record. Inside a hook, if you omit</td>
</tr>
<tr>
<td></td>
<td>this part of the syntax, the Entity object corresponding to the current data</td>
</tr>
<tr>
<td></td>
<td>record is assumed (VBScript only).</td>
</tr>
<tr>
<td>field_name</td>
<td>A String containing a valid field name of this Entity object.</td>
</tr>
<tr>
<td>Return value</td>
<td>The FieldInfo object corresponding to the specified field.</td>
</tr>
</tbody>
</table>

**Member of:** Entity object

**Remarks:**

This method returns a FieldInfo object from which you can obtain information about the field. This method does not return the actual value stored in the field. To retrieve the actual value (or values), call this method first and then call the FieldInfo object’s GetValue method or GetValueAsList method.

**Examples:**

```vb
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
```
See Also:

AddFieldValue method
DeleteFieldValue method
GetAllFieldValues method
SetFieldValue method
GetValue method of the FieldInfo object
GetValueAsList method of the FieldInfo object
FieldInfo object
Notification Hook Code Example
GetInvalidFieldValues method

Returns an array of FieldInfo objects corresponding to all the Entity’s invalid fields.

**VBScript Syntax:**

```
[entity] GetInvalidFieldValues
```

**Perl Syntax:**

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

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entity</td>
<td>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).</td>
</tr>
</tbody>
</table>

| Return value | A Variant containing an Array of FieldInfo objects. 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. |

**Member of:** Entity object

**Remarks:**

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.

**See Also:**

GetAllFieldValues method
GetFieldValue method
Validate method
ValidityChangedThisSetValue method of the FieldInfo object
FieldInfo object
GetLegalActionDefNames method

Returns the names of the actions that can be used on this Entity object.

**VBScript Syntax:**

```vbnet
[entity.]GetLegalActionDefNames
```

**Perl Syntax:**

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

**Identifier** | **Description**
--- | ---
**entity** | An Entity object representing a user data record. Inside a hook, if you omit this part of the syntax, the Entity object corresponding to the current data record is assumed (VBScript only).

**Return value** | A Variant containing an Array of Strings. 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.

**Member of:** Entity object

**Remarks:**

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.

**Examples:**

```vbnet
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
```

```perl
$entity->GetLegalActionDefNames();
```
actionDefType = entityDefObj.GetActionDefType(actionDef)
if actionDefType = AD_MODIFY Then
    sessionObj>EditEntity(entity, actionDef)
    Exit For
End If
Next

**See Also:**
GetActionDefNames method
EditEntity method of the Session object
Session object
Notation Conventions
GetOriginal method

Returns the Entity object that is marked as the parent of this duplicate object.

**VBScript Syntax:**

```vbscript
[entity.]GetOriginal
```

**Perl Syntax:**

```perl
$entity->GetOriginal();
```

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>entity</code></td>
<td>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).</td>
</tr>
</tbody>
</table>

**Return value**
The Entity object of which `entity` is a duplicate.

**Member of:** Entity object

**Remarks:**

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.

**Examples:**

```vbnet
' 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

See Also:

GetAllDuplicates method
GetDuplicates method
GetOriginalID method
HasDuplicates method
IsDuplicate method
IsOriginal method
MarkEntityAsDuplicate method of the Session object
UnmarkEntityAsDuplicate method of the Session object
Session object
Duplicates Code Example
GetOriginalID method

Returns the visible ID of this object’s original Entity object.

VBScript Syntax:

`[entity.]GetOriginalID`

Perl Syntax:

`$entity->GetOriginalID();`

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entity</td>
<td>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).</td>
</tr>
</tbody>
</table>

Return value: A String containing the ID of this object’s original Entity.

Member of: Entity object

Remarks:

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.

Examples:

`' 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.GetDisplayID

    ' Get the ID of the original record
originalID = entity.GetOriginalID
OutputDebugString "The parent of record " & duplicateID & 
    " is record " & originalID
End If

**See Also:**

GetAllDuplicates method
GetDuplicates method
GetOriginal method
HasDuplicates method
IsDuplicate method
IsOriginal method
MarkEntityAsDuplicate method of the Session object
UnmarkEntityAsDuplicate method of the Session object
Session object
Duplicates Code Example
GetSession method

Returns the current Session object.

**VBScript Syntax:**

```vb
[entity:]GetSession
```

**Perl Syntax:**

```perl
$entity->[GetSession()];
```

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entity</td>
<td>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.</td>
</tr>
</tbody>
</table>

**Return value**

The Session object representing the current database-access session.

**Member of:** Entity object

**Remarks:**

This method instantiate a new Session object using the current session information. This method is intended for use in hook code only and should not be called from any other context.

If you are creating a standalone application, you cannot call this method to obtain a Session object. You must create your own Session object and pass it to any standalone application methods that need it.

You can use this method to obtain the Session object associated with the current user. See the description of the Session object for more information on how to use this object.

**Examples:**

```vb
set sessionObj = GetSession
```
See Also:
UserLogon method of the Session object
Session object
Duplicates Code Example
GetType method

Returns the type (state-based or stateless) of the Entity.

**VBScript Syntax:**

```vb
[entity:]GetType
```

**Perl Syntax:**

```perl
$entity->GetType();
```

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entity</td>
<td>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).</td>
</tr>
</tbody>
</table>

**Return value**

A Long whose value is an EntityType constant: REQ_ENTITY for a state-based Entity object or AUX_ENTITY for a stateless Entity object.

**Member of:** Entity object

**Remarks:**

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.

**Examples:**

```vb
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
```
See Also:

EntityType
Notation Conventions
HasDuplicates method

Reports whether this object is the original of one or more duplicates.

**VBScript Syntax:**

```vbscript
[entity.]HasDuplicates
```

**Perl Syntax:**

```perl
$entity-&gt;HasDuplicates();
```

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entity</td>
<td>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).</td>
</tr>
</tbody>
</table>

**Return value**

A Boolean whose value is True if the Entity has any duplicates, otherwise False.

**Member of:** Entity object

**Remarks:**

An Entity can have more than one duplicate. Furthermore, an Entity can have duplicates and also be a duplicate itself. See the IsDuplicate method and IsOriginal method for details.

**Examples:**

```
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
```
See Also:

GetAllDuplicates method
GetDuplicates method
GetOriginal method
GetOriginalID method
IsDuplicate method
IsOriginal method
MarkEntityAsDuplicate method of the Session object
UnmarkEntityAsDuplicate method of the Session object
Session object
Duplicates Code Example
InvalidateFieldChoiceList function

Erases the values in a (dynamic) choice list, which can then be reset with SetFieldChoiceList function.

**VBScript Syntax:**

\[\text{entity.} \text{InvalidateFieldChoiceList} \text{field_name}\]

**Perl Syntax:**

$\text{entity->InvalidateFieldChoiceList} (\text{field_name});$

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entity</td>
<td>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).</td>
</tr>
<tr>
<td>field_name</td>
<td>A String that identifies a valid field name of an entity.</td>
</tr>
<tr>
<td>return value</td>
<td>None.</td>
</tr>
</tbody>
</table>

**Member of:** Entity object

**Remarks:**

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.

**Example:**

```plaintext```
void InvalidateFieldChoiceList(fieldname)
```

**See Also:**

SetFieldChoiceList function
**IsDuplicate method**

Indicates whether this Entity object has been marked as a duplicate of another Entity object.

SetFieldChoiceList function.

**VBScript Syntax:**

\[entity.IsDuplicate\]

**Perl Syntax:**

\$entity->IsDuplicate();

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entity</td>
<td>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).</td>
</tr>
</tbody>
</table>

**Return value**

A Boolean whose value is True if this Entity object has been marked as a duplicate of another Entity object, otherwise False.

**Member of:** Entity object

**Remarks:**

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.

**Examples:**

`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.GetDisplay
OutputDebugString "The parent of record " & duplicateID & " is record " & originalID

End If

See Also:

GetAllDuplicates method
GetDuplicates method
GetOriginal method
HasDuplicates method
IsOriginal method
MarkEntityAsDuplicate method of the Session object
UnmarkEntityAsDuplicate method of the Session object
Session object
Duplicates Code Example
IsEditable method

Returns True if the Entity object can be modified at this time.

**VBScript Syntax:**

```vbnet
[entity.]IsEditable
```

**Perl Syntax:**

```perl
$entity->IsEditable();
```

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entity</td>
<td>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).</td>
</tr>
</tbody>
</table>

**Return value**

A Boolean whose value is True if the Entity is currently editable, otherwise False.

**Member of:** Entity object

**Remarks:**

To edit an Entity object, you must either create a new object using the BuildEntity method or open an existing object for editing with the EditEntity method. An Entity object remains editable until you either commit your changes with the Commit method or revert the Entity object with the Revert method.

**Examples:**

```vbnet
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
```
See Also:
Commit method
Revert method
BuildEntity method of the Session object
EditEntity method of the Session object
Session object
IsOriginal method

Returns True if this Entity has duplicates but is not itself a duplicate.

VBScript Syntax:

```vbnet
[entity.]IsOriginal
```

Perl Syntax:

```perl
$entity->IsOriginal();
```

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entity</td>
<td>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).</td>
</tr>
</tbody>
</table>

**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.

**Member of:** Entity object

**Remarks:**

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.

**Examples:**

```vbnet
' 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
```

Entity object
duplicateObj = duplicateLink.GetChildEntity
duplicateID = duplicateObj.GetDisplayName
OutputDebugString "Parent ID:" & originalID & 
  " child Id:" & duplicateID
Next
End If

**See Also:**

GetAllDuplicates method
GetDuplicates method
GetOriginal method
GetOriginalID method
HasDuplicates method
IsDuplicate method
MarkEntityAsDuplicate method of the Session object
UnmarkEntityAsDuplicate method of the Session object
Session object
Duplicates Code Example
LookupStateName method

Returns the name of the Entity object’s current state.

**VBScript Syntax:**

```vb
[entity.]LookupStateName
```

**Perl Syntax:**

```perl
$entity->LookupStateName();
```

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>entity</code></td>
<td>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).</td>
</tr>
</tbody>
</table>

**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 (`""`).

**Member of:** Entity object

**Remarks:**

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.

**Examples:**

```vb
currentState = LookupStateName
```

**See Also:**

GetFieldValue method
EditEntity method of the Session object
Session object
Revert method

Discards any changes made to the Entity object.

**VBScript Syntax:**

```
[entity:]Revert
```

**Perl Syntax:**

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

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entity</td>
<td>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).</td>
</tr>
</tbody>
</table>

**Return value**

None.

**Member of:** Entity object

**Remarks:**

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.

**Examples:**

```vb
set sessionObj = GetSession
entityToEdit = sessionObj.GetEntity("defect", "BUGID00000042")
```
sessionObj.EditEntity(entityToEdit, "modify")

' Revert the changes to the record
entityToEdit.Revert

**See Also:**

Commit method
IsEditable method
Validate method
EditEntity method of the Session object
Session object
Entity Code Example
SetFieldChoiceList function

Resets a dynamic choice list. Can be use with InvalidateFieldChoiceList function to empty any values already stored.

**VBScript Syntax:**

```
[entity.]SetFieldChoiceList fieldName, choiceList
```

**Perl Syntax:**

```
$entity->SetFieldChoiceList(fieldName, choiceList);
```

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entity</td>
<td>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).</td>
</tr>
<tr>
<td>fieldName</td>
<td>A String that identifies a valid field name of an entity.</td>
</tr>
<tr>
<td>choiceList</td>
<td>[A VB variant containing a string or] a Perl string array.</td>
</tr>
</tbody>
</table>

**Return value**

None.

**Member of:** Entity object

**Remarks:**

Use this function to force the ClearQuest client to fetch the new choice list values. You can set the values with this function or by other means (for example, a hook script).

You can design your schema so that ClearQuest recalculates a choice list every time a user interacts with it (no cached values), or only the first time (cached values). If you want to refresh cached values, call InvalidateFieldChoiceList function 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 function because no values pre-exist in cache memory.)
Use these two methods in a Value-Changed Field hook. For example, if the end-user selects a new item from the list of projects, the record type changes, and the form needs a refreshed dependent choice list.

**VBScript Example:**

```
SetFieldChoiceList(fieldname, VARIANT choiceList)
```

Sets a list of acceptable values for the field. The parameter choiceList is of the type of Variant and must contain an array of strings.

**NOTE:** In the current implementation, you cannot pass a reference to a variant.

**Perl Example:**

```
SetFieldChoiceList($fieldname, @choiceList)
```

Sets a list of acceptable values for the field. The parameter choiceList is of the type of array and must contain an array of strings.

**See Also:**

InvalidFieldChoiceList function
SetFieldRequirednessForCurrentAction method

Sets the behavior of a field for the duration of the current action.

**VBScript Syntax:**

```vbnet
[entity.]SetFieldRequirednessForCurrentAction field_name, newValue
```

**Perl Syntax:**

```perl
$entity->SetFieldRequirednessForCurrentAction(field_name, newValue);
```

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entity</td>
<td>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).</td>
</tr>
<tr>
<td>field_name</td>
<td>A String that identifies a valid field name of entity.</td>
</tr>
<tr>
<td>newValue</td>
<td>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.)</td>
</tr>
</tbody>
</table>

**Return value**

None.

**Member of:** Entity object

**Remarks:**

Use this method to set the field behavior to mandatory, optional, or read-only. Once the action has been committed, the behavior of the field reverts to read-only.

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

**Examples:**

```vbnet
' Change all mandatory fields to optional
fieldNameList = GetFieldNames
For Each fieldName in fieldNameList
    fieldReq = GetFieldRequiredness(fieldName)
    if fieldReq = AD_MANDATORY Then
        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.)
```

Entity object 275
SetFieldRequirednessForCurrentAction fieldName, AD_OPTIONAL
End If
Next

See Also:

GetFieldRequiredness method
Behavior
Notation Conventions
SetFieldValue method

Places the specified value in the named field.

**VBScript Syntax:**

`[entity.]SetFieldValue field_name, new_value`

**Perl Syntax:**

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

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entity</td>
<td>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).</td>
</tr>
<tr>
<td>field_name</td>
<td>A String containing a valid field name of this Entity object.</td>
</tr>
<tr>
<td>new_value</td>
<td>A Variant containing the new setting for the field.</td>
</tr>
</tbody>
</table>

**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.

**Member of:** Entity object

**Remarks:**

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.

**Examples:**

' Set two field values, but only check errors for ' the second field.
entity.SetFieldValue "field1", "new value"
returnVal = SetFieldValue("field2", "100")

**See Also:**

AddFieldValue method
GetFieldValue method
ValidityChangedThisSetValue method
ValueChangedThisSetValue method
FieldInfo object
Duplicates Code Example
Entity Code Example
Validate method

Validates the Entity object and reports any errors.

**VBScript Syntax:**

```
[entity.]Validate
```

**Perl Syntax:**

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

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entity</td>
<td>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).</td>
</tr>
</tbody>
</table>

**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.

**Member of:** Entity object

**Remarks:**

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.

**Examples:**

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

' modify the Entity object
entityObj.Validate
entityObj.Commit
' The Entity object is no longer editable

**See Also:**

Commit method
GetInvalidFieldValues method
Revert method
FieldInfo object
Duplicates Code Example
**EntityDef Methods**

<table>
<thead>
<tr>
<th>Method name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DoesTransitionExist method</td>
<td>Returns the list of transitions that exist between two states.</td>
</tr>
<tr>
<td>GetActionDefNames method</td>
<td>Returns the action names defined in the EntityDef object.</td>
</tr>
<tr>
<td>GetActionDefType method</td>
<td>Identifies the type of the specified action.</td>
</tr>
<tr>
<td>GetActionDestStateName method</td>
<td>Returns the name of the destination state of a given action def.</td>
</tr>
<tr>
<td>GetFieldDefNames method</td>
<td>Returns the field names defined in the EntityDef object.</td>
</tr>
<tr>
<td>GetFieldDefType method</td>
<td>Identifies the type of data that can be stored in the specified field.</td>
</tr>
<tr>
<td>GetFieldReferenceEntityDef method</td>
<td>Returns the type of record referenced by the specified field.</td>
</tr>
<tr>
<td>GetHookDefNames method</td>
<td>Returns the list of named hooks associated with records of this type.</td>
</tr>
<tr>
<td>GetLocalFieldPathNames method</td>
<td>Returns the path names of local fields.</td>
</tr>
<tr>
<td>GetName method</td>
<td>Returns the name of the EntityDef object’s corresponding record type.</td>
</tr>
<tr>
<td>GetStateDefNames method</td>
<td>Returns the state names defined in the EntityDef object.</td>
</tr>
<tr>
<td>GetType method</td>
<td>Returns the type (state-based or stateless) of the EntityDef.</td>
</tr>
<tr>
<td>IsActionDefName method</td>
<td>Identifies whether the EntityDef object contains an action with the specified name.</td>
</tr>
<tr>
<td>IsFamily method</td>
<td>Returns true of a given entitydef defines a family.</td>
</tr>
<tr>
<td>IsFieldDefName method</td>
<td>Identifies whether the EntityDef object contains a field with the specified name.</td>
</tr>
<tr>
<td>IsStateDefName method</td>
<td>Identifies whether the EntityDef object contains a state with the specified name.</td>
</tr>
<tr>
<td>IsSystemOwnedFieldName method</td>
<td>Returns a Bool indicating whether the specified field is owned by the system.</td>
</tr>
</tbody>
</table>
See Also:
Entity object
Session object
EntityDef Code Example
**DoesTransitionExist method**

Returns the list of transitions that exist between two states.

**VB Syntax:**

```
[entitydef.]DoesTransitionExist sourceState, destState
```

**Perl Syntax:**

```
$entitydef->DoesTransitionExist(sourceState, destState);
```

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entitydef</td>
<td>An EntityDef object corresponding to a record type in a schema.</td>
</tr>
<tr>
<td>sourceState</td>
<td>A String containing the name of the state that is the source of the transition.</td>
</tr>
<tr>
<td>destState</td>
<td>A String containing the name of the state that is the destination of the transition.</td>
</tr>
</tbody>
</table>

**Return value**

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.

**Member of:** EntityDef object

**Remarks:**

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.

**Examples:**

```javascript
set sessionObj = GetSession
set entityDefObj = sessionObj.GetEntityDef(GetEntityDefName())

transitions = entityDefObj.DoesTransitionExist("open", "resoved")
If transitions <> Empty Then
   ' Simply initiate an action using the first entry.
   sessionObj.EditEntity entity, transitions(0)
```
End If

**See Also:**

GetActionDefNames method
IsActionDefName method
GetActionDefNames method

Returns the action names defined in the EntityDef object.

**VB Syntax:**

```
[entitydef.]GetActionDefNames
```

**Perl Syntax:**

```
$entitydef->GetActionDefNames();
```

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entitydef</td>
<td>An EntityDef object corresponding to a record type in a schema.</td>
</tr>
<tr>
<td>Return value</td>
<td>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.</td>
</tr>
</tbody>
</table>

**Member of:** EntityDef object

**Remarks:**

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.

**Examples:**

```vbnet
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
```
See Also:

GetActionDefType method
IsActionDefName method
ActionType
EntityDef Code Example
GetActionDefType method

Identifies the type of the specified action.

**VB Syntax:**

```
entitydef.GetActionDefType action_def_name
```

**Perl Syntax:**

```
$entitydef->GetActionDefType(action_def_name);
```

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entitydef</td>
<td>An EntityDef object corresponding to a record type in a schema.</td>
</tr>
<tr>
<td>action_def_name</td>
<td>A String that identifies a valid action name of entitydef.</td>
</tr>
<tr>
<td>Return value</td>
<td>A Long that specifies the type of the action specified in action_def_name. The value corresponds to one of the ActionType enumeration constants.</td>
</tr>
</tbody>
</table>

**Member of:** EntityDef object

**Remarks:**

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 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.

**Examples:**

```vbnet
set sessionObj = GetSession
set entityDefObj = sessionObj.GetEntityDef(GetEntityDefName())

sessionObj.OutputDebugString "Modify action names for " & _
    entityDefObj.GetName()
```
' List the action names whose type is "modify"
nameList = entityDefObj.GetActionDefNames()
For Each actionName in nameList
    actionType = entityDefObj.GetActionDefType(actionName)
    If actionType = AD_MODIFY Then
        sessionObj.OutputDebugString actionName
    End If
Next

See Also:
GetActionDefNames method
IsActionDefName method
EntityDef Code Example
Notation Conventions
GetActionDestStateName method

Returns the destination state name associated with the current action.

**VB Syntax:**

`entitydef.GetActionDestStateName actionDefName`

**Perl Syntax:**

`$entitydef->GetActionDestStateName(actionDefName);

### Identifier Description

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>entitydef</code></td>
<td>An EntityDef object corresponding to a record type in a schema.</td>
</tr>
<tr>
<td><code>actionDefName</code></td>
<td>A String that identifies a valid action name.</td>
</tr>
<tr>
<td><strong>Return value</strong></td>
<td>A String that specifies the destination state of a given action def.</td>
</tr>
</tbody>
</table>

**Member of:** EntityDef object

**Remarks:**

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.

**See Also:**

GetDefaultActionName
GetFieldDefNames method

Returns the field names defined in the EntityDef object.

**VB Syntax:**

```vb
entitydef.GetFieldDefNames
```

**Perl Syntax:**

```perl
$entitydef->GetFieldDefNames();
```

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>entitydef</code></td>
<td>An EntityDef object corresponding to a record type in a schema.</td>
</tr>
<tr>
<td><code>Return value</code></td>
<td>A Variant containing an Array whose elements are Strings. Each String contains the name of one field. If the EntityDef object has no fields, the return value is an Empty variant.</td>
</tr>
</tbody>
</table>

**Member of:** EntityDef object

**Remarks:**

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.

**Examples:**

```vbnet
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
```
See Also:

GetFieldDefType method
IsFieldDefName method
EntityDef Code Example
GetFieldDefType method

Identifies the type of data that can be stored in the specified field.

VB Syntax:

```vbnet
entitydef.GetFieldDefType field_def_name
```

Perl Syntax:

```perl
$entitydef->GetFieldDefType(field_def_name);
```

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entitydef</td>
<td>An EntityDef object corresponding to a record type in a schema.</td>
</tr>
<tr>
<td>field_def_name</td>
<td>A String that identifies a valid field name of entitydef.</td>
</tr>
<tr>
<td>Return value</td>
<td>A Long that specifies what type of data can be stored in the named field. The value corresponds to one of the FieldType enumeration constants.</td>
</tr>
</tbody>
</table>

**Member of:** EntityDef object

**Remarks:**

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 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.

**Examples:**

```vbnet
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

See Also:
GetFieldDefNames method
IsFieldDefName method
EntityDef Code Example
Notation Conventions
GetFieldReferenceEntityDef method

Returns the type of record referenced by the specified field.

**VB Syntax:**

`entitydef.GetFieldReferenceEntityDef field_name`

**Perl Syntax:**

`
$entitydef->GetFieldReferenceEntityDef(field_name);
`

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>entitydef</code></td>
<td>An EntityDef object corresponding to a record type in a schema.</td>
</tr>
<tr>
<td><code>field_name</code></td>
<td>A String that identifies a valid field name of entitydef.</td>
</tr>
<tr>
<td><strong>Return value</strong></td>
<td>An EntityDef object corresponding to the type of record referenced</td>
</tr>
<tr>
<td></td>
<td>by the specified field.</td>
</tr>
</tbody>
</table>

**Member of:** EntityDef object

**Remarks:**

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.

**Examples:**

```
set sessionObj = GetSession
def 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 refDefObj = entityDefObj.GetFieldReferenceEntityDef(fieldName)
      sessionObj.OutputDebugString refDefObj.GetName()
   End If
Next
```
See Also:

GetFieldDefType method
Notation Conventions
GetHookDefNames method

Returns the list of named hooks associated with records of this type.

**VB Syntax:**

```vbnet
def.GetHookDefNames field_def_name
```

**Perl Syntax:**

```perl
$entitydef->GetHookDefNames(field_def_name);
```

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>entitydef</code></td>
<td>An EntityDef object corresponding to a record type in a schema.</td>
</tr>
<tr>
<td><code>Return value</code></td>
<td>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.</td>
</tr>
</tbody>
</table>

**Member of:** EntityDef object

**Remarks:**

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.

**Examples:**

```vbnet
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
```
See Also:

- GetActionDefNames method
- GetFieldDefNames method
GetLocalFieldPathNames method

Returns the path names of local fields.

**VB Syntax:**

```
entitydef.GetLocalFieldPathNames visible_only
```

**Perl Syntax:**

```
$entitydef->GetLocalFieldPathNames(visible_only);
```

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entitydef</td>
<td>An EntityDef object corresponding to a record type in a schema.</td>
</tr>
<tr>
<td>visible_only</td>
<td>A Bool, which if true restricts the list of fields to only those that are visible.</td>
</tr>
</tbody>
</table>

**Return value** A Variant containing a list of strings.

**Member of:** EntityDef object

**Remarks:**

Each string in the returned variant contains the path name of a single field.

**Examples:**

```
set sessionObj  = GetSession
set entityDefObj = sessionObj.GetEntityDef(GetEntityDefName())

pathNames = entityDefObj.GetLocalFieldPathNames(False)
For Each name in pathNames
    sessionObj.OutputDebugString "Path name: " & name
Next
```

**See Also:**

GetFieldDefNames method
IsFieldDefName method
GetName method

Returns the name of the EntityDef object’s corresponding record type.

**VB Syntax:**

`entitydef.GetName`

**Perl Syntax:**

`
$entitydef->GetNamed();
`

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>entitydef</code></td>
<td>An EntityDef object corresponding to a record type in a schema.</td>
</tr>
<tr>
<td><strong>Return value</strong></td>
<td>A String whose value is the name of the EntityDef object’s corresponding record type.</td>
</tr>
</tbody>
</table>

**Member of:** EntityDef object

**Remarks:**

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.

**Examples:**

```vbnet
set sessionObj = GetSession
set entityDefObj = sessionObj.GetEntityDef(GetEntityDefName())
sessionObj.OutputDebugString "Name of record type: " & _
    entityDefObj.GetName()
```

**See Also:**

GetType method
EntityDef Code Example
GetStateDefNames method

Returns the state names defined in the EntityDef object.

**VB Syntax:**

`entitydef.GetStateDefNames`

**Perl Syntax:**

`$entitydef->GetStateDefNames();`

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>entitydef</code></td>
<td>An EntityDef object corresponding to a record type in a schema.</td>
</tr>
<tr>
<td>Return value</td>
<td>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.</td>
</tr>
</tbody>
</table>

**Member of:** EntityDef object

**Remarks:**

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.

**Examples:**

```vbnet
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
```
See Also:

- GetType method
- IsStateDefName method
- EntityDef Code Example
- Notation Conventions
**GetType method**

Returns the type (state-based or stateless) of the EntityDef.

**VB Syntax:**

`entitydef.GetType`

**Perl Syntax:**

`$entitydef->GetType();`

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>entitydef</code></td>
<td>An EntityDef object corresponding to a record type in a schema.</td>
</tr>
</tbody>
</table>

**Return value**

A `Long` whose value is an EntityType constant: `REQ_ENTITY` for a state-based EntityDef object or `AUX_ENTITY` for a stateless EntityDef object.

**Member of:** EntityDef object

**Remarks:**

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.

**Examples:**

```vb
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
```
See Also:

GetName method
EntityDef Code Example
Notation Conventions
**IsActionDefName method**

Identifies whether the EntityDef object contains an action with the specified name.

**VB Syntax:**

```vbnet
entitydef.IsActionDefName name
```

**Perl Syntax:**

```perl
$entitydef->IsActionDefName(name);
```

**Member of:** EntityDef object

**Examples:**

```vbnet
set sessionObj = GetSession
set entityDefObj = sessionObj.GetEntityDef(GetEntityDefName())

If entityDefObj.IsActionDefName("open") Then
    sessionObj.OutputDebugString "The record type supports the open action"
End If
```

**See Also:**

GetActionDefNames method
GetActionDefType method
IsFamily method

Returns the boolean value of True if this entitydef defines a family.

**VB Syntax:**

`entitydef.IsFamily entitydef`

**Perl Syntax:**

`$entitydef->IsFamily(entitydef);`

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>entitydef</code></td>
<td>An EntityDef object corresponding to a record type in a schema.</td>
</tr>
<tr>
<td>Return value</td>
<td>A Boolean. True signifies the entitydef does define a family record type.</td>
</tr>
</tbody>
</table>

**Member of:** EntityDef object

**Remarks:**

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.

**See Also:**

GetEntityDef method
GetEntityDefFamilyNames method
IsFieldDefName method

Identifies whether the EntityDef object contains a field with the specified name.

**VB Syntax:**

`entitydef.IsFieldDefName name`

**Perl Syntax:**

`$entitydef->IsFieldDefName(name);`

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>entitydef</code></td>
<td>An EntityDef object corresponding to a record type in a schema.</td>
</tr>
<tr>
<td><code>name</code></td>
<td>A String containing the name of the field to verify.</td>
</tr>
</tbody>
</table>

_Return value_ | True if name is the name of an actual field in the EntityDef object; otherwise False. |

**Member of:** EntityDef object

**See Also:**

GetFieldDefNames method
GetFieldDefType method
IsStateDefName method

Identifies whether the EntityDef object contains a state with the specified name.

**VB Syntax:**

```vbnet
typedef.IsStateDefName name
```

**Perl Syntax:**

```perl
$entitydef->IsStateDefName(name);
```

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entitydef</td>
<td>An EntityDef object corresponding to a record type in a schema.</td>
</tr>
<tr>
<td>name</td>
<td>A String containing the name of the state to verify.</td>
</tr>
<tr>
<td><strong>Return value</strong></td>
<td>True if name is the name of an actual state in the EntityDef object; otherwise False.</td>
</tr>
</tbody>
</table>

**Member of:** EntityDef object

**See Also:**

GetStateDefNames method
EntityDef Code Example
IsSystemOwnedFieldDefName method

Returns a Bool indicating whether the specified field is owned by the system.

**VB Syntax:**

```vbnet
typedef .IsSystemOwnedFieldDefName field_name
```

**Perl Syntax:**

```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 entitydef.

**Return value** | True if the field is owned by the system, otherwise False.

**Member of:** EntityDef object

**Remarks:**

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.

**See Also:**

GetFieldDefNames method
QueryDef object

**QueryDef properties**

<table>
<thead>
<tr>
<th>Property name</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IsAggregated property</td>
<td>Read-only</td>
<td>Returns a Boolean indicating whether any fields of the query are aggregated.</td>
</tr>
<tr>
<td>IsDirty property</td>
<td>Read-only</td>
<td>Returns a Boolean indicating whether the query has changed.</td>
</tr>
<tr>
<td>IsMultiType method</td>
<td>Read-only</td>
<td>Returns a Boolean indicating whether the QueryDef object is multitype.</td>
</tr>
<tr>
<td>Name property</td>
<td>Read/Write</td>
<td>Sets or returns the name associated with the query.</td>
</tr>
<tr>
<td>QueryType property</td>
<td>Read-only</td>
<td>Returns an Integer indicating list, report, or chart.</td>
</tr>
<tr>
<td>SQL property</td>
<td>Read/Write</td>
<td>Sets or returns the SQL string associated with the query.</td>
</tr>
</tbody>
</table>

**QueryDef methods**

<table>
<thead>
<tr>
<th>Method name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BuildField method</td>
<td>Selects a field to include in the query's search results.</td>
</tr>
<tr>
<td>BuildFilterOperator method</td>
<td>Creates the top-level QueryFilterNode object for the query.</td>
</tr>
<tr>
<td>Save method</td>
<td>Saves the query to the specified file.</td>
</tr>
</tbody>
</table>

**See Also:**

BuildQuery method of the Session object  
ResultSet Object  
Session object  
BuildQuery Code Example
IsAggregated property

Returns a Bool indicating whether any fields of the query are aggregated.

**VB Syntax:**

`querydef.IsAggregated`

**Perl Syntax:**

`$querydef->GetIsAggregated();`

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>querydef</code></td>
<td>A QueryDef object.</td>
</tr>
</tbody>
</table>

**Return value**

True if any of the fields in the query are aggregated, otherwise False.

**Member of:** QueryDef object

**Remarks:**

Aggregated fields are grouped together for display in the resulting query or chart. This property is read-only.

**See Also:**

SQL property
IsDirty property

Returns a Boolean indicating whether the query has changed.

**VB Syntax:**

`querydef.IsDirty`

**Perl Syntax:**

`$querydef->GetIsDirty();`

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>querydef</td>
<td>A QueryDef object.</td>
</tr>
</tbody>
</table>

**Return value**

True if the query has changed, otherwise False.

**Member of:** QueryDef object

**Remarks:**

A QueryDef object is considered dirty if any of its fields or filters have changed since the last time it was saved.

**See Also:**

Save method
**IsMultiType method**

Returns a Boolean indicating whether a given querydef has the property of being multitype.

**VB Syntax:**

`querydef.IsMultiType`

**Perl Syntax:**

`$querydef->IsMultiType();`

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>querydef</td>
<td>A QueryDef object.</td>
</tr>
</tbody>
</table>

**Return value**

True if the object is multitype.

**Member of:** QueryDef object

**Remarks:**

One use case for this method is to support querying similar record types (for example, defects and enhancement requests) in a single query. This method can be used in conjunction with GetEntityDefFamily method and GetEntityDefFamilyNames method.

**See Also:**

GetEntityDefFamily method
GetEntityDefFamilyNames method
**Name property**

Sets or returns the name associated with the query.

**VB Syntax:**

```vbnet
querydef.Name [value]
```

**Perl Syntax:**

```perl
$querydef->GetName();
$querydef->SetName(newName);
```

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>querydef</td>
<td>A QueryDef object.</td>
</tr>
<tr>
<td>value</td>
<td>A String containing the name of the query.</td>
</tr>
</tbody>
</table>

**Member of:** QueryDef object

**See Also:**

Save method
**QueryType property**

Returns an integer indicating whether the saved query has the property of being a list, a report, or a chart.

**VB Syntax:**

`querydef.QueryType`

**Perl Syntax:**

`$querydef->GetQueryType();`

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>querydef</td>
<td>A QueryDef object.</td>
</tr>
</tbody>
</table>

**Return value**

An Integer indicating whether a saved query is a list (_LIST_QUERY_), a report (_REPORT_QUERY_), or or query (_CHART_QUERY_).

**Member of:** QueryDef object

**See Also:**

QueryType enumerated constants
**SQL property**

Sets or returns the SQL string associated with the query.

**VB Syntax:**

`querydef.SQL [= Value]`

**Perl Syntax:**

`$querydef->GetSQL();`

`$querydef->SetSQL(string_of_SQL_statements);`

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>querydef</td>
<td>A QueryDef object.</td>
</tr>
<tr>
<td>value</td>
<td>A String containing the SQL that will be executed when the query is run.</td>
</tr>
</tbody>
</table>

**Member of:** QueryDef object

**Remarks:**

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.

**See Also:**

BuildSQLQuery method of the Session object

Session object
BuildField method

Selects a field to include in the query’s search results.

**VB Syntax:**

```vbnet
querydef.BuildField field_name
```

**Perl Syntax:**

```perl
$querydef->BuildField(field_name);
```

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>querydef</td>
<td>A QueryDef object.</td>
</tr>
<tr>
<td>field_name</td>
<td>A String identifying a valid field of the associated EntityDef object.</td>
</tr>
<tr>
<td>Return value</td>
<td>None.</td>
</tr>
</tbody>
</table>

**Member of:** QueryDef object

**Remarks:**

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).
See Also:

BuildFilterOperator method
BuildQuery method of the Session object
GetFieldDefNames method of the EntityDef object
EntityDef object
ResultSet Object
Session object
BuildQuery Code Example
BuildFilterOperator method

Creates the top-level QueryFilterNode object for the query.

**VB Syntax:**

querydef.BuildFilterOperator bool_operator

**Perl Syntax:**

$querydef->BuildFilterOperator(bool_operator);

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>querydef</td>
<td>A QueryDef object.</td>
</tr>
<tr>
<td>bool_operator</td>
<td>A Long whose value is one of the BoolOp enumeration constants.</td>
</tr>
</tbody>
</table>

**Return value** The newly created QueryFilterNode object.

**Member of:** QueryDef object

**Remarks:**

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.

For example:

(submitter = jJones OR submitter = clopez OR submitter = kwong) AND submit_date < 01/03/2000

In this expression, the top-level Boolean operator is the AND operator. To start constructing this query expression, you use this method to create the filter that has the top-level operator:

filterNode1 = myQueryDef. BuildFilterOperator (AD_BOOL_OP_AND)
You use this method just once to construct the root of the tree. To continue adding filters, you call the methods of the returned QueryFilterNode objects. For example, to complete the previous expression, you would write the following code:

```python
filterNode1.BuildFilter ('submit_date', AD_COMP_OP_LT, '1997-11-19')
filterNode2 = filterNode1.BuildFilterOperator (AD_BOOL_OP_OR)
filterNode2.BuildFilter ('submitter', AD_COMP_OP_EQ, 'jjones')
filterNode2.BuildFilter ('submitter', AD_COMP_OP_EQ, 'clopez')
filterNode2.BuildFilter ('submitter', AD_COMP_OP_EQ, 'kwong')
```

More-complicated expressions are created by recursively attaching more nodes as needed. For more information, see the QueryFilterNode object.

If a node contains only one condition, the value of the bool_operator parameter is irrelevant. For example, if the entire query expression is ‘SUBMITTER = JJONES’, you could construct the query expression as follows:

```python
' You could use either AD_BOOL_OP_AND or AD_BOOL_OP_OR for this
' expression since there is only one condition.
filterNode = myQueryDef.BuildFilterOperator (AD_BOOL_OP_AND)
filterNode.BuildFilter ('submitter', AD_COMP_OP_EQ, 'jjones')
```

**Note:** It is perfectly legal to create a QueryDef object that has no filtering (in other words, no query expression). In this case, all of the records in the database are retrieved.

**See Also:**

BuildFilter method of the QueryFilterNode object
BuildFilterOperator method of the QueryFilterNode object
BoolOp enumerated type
QueryFilterNode object
BuildQuery Code Example
Notation Conventions
Save method

Saves the query to the specified file.

**VB Syntax:**

`querydef.Save fileName`

**Perl Syntax:**

`$querydef->Save(fileName);`

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>querydef</td>
<td>A QueryDef object.</td>
</tr>
<tr>
<td>fileName</td>
<td>A String containing the name of the file.</td>
</tr>
<tr>
<td><strong>Return value</strong></td>
<td>A Bool containing the value True if the query was successfully saved, otherwise False.</td>
</tr>
</tbody>
</table>

**Member of:** QueryDef object

**See Also:**

Name property
ResultSet Object

ResultSet methods

<table>
<thead>
<tr>
<th>Method name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddParamValue method</td>
<td>Assigns one or more values to a parameter.</td>
</tr>
<tr>
<td>ClearParamValues method</td>
<td>Clears all values associated with a parameter.</td>
</tr>
<tr>
<td>Execute method</td>
<td>Runs the query and fills the result set with data.</td>
</tr>
<tr>
<td>GetColumnLabel method</td>
<td>Returns the heading text for the specified column.</td>
</tr>
<tr>
<td>GetColumnType method</td>
<td>Returns the type of data stored in the specified column.</td>
</tr>
<tr>
<td>GetColumnValue method</td>
<td>Returns the value stored in the specified column of the current row.</td>
</tr>
<tr>
<td>GetNumberOfColumns method</td>
<td>Returns the number of columns in each row of the result set.</td>
</tr>
<tr>
<td>GetNumberOfParams method</td>
<td>Returns the number of parameters in this query.</td>
</tr>
<tr>
<td>GetParamChoiceList method</td>
<td>Returns a list of permitted values for the parameter.</td>
</tr>
<tr>
<td>GetParamComparisonOperator method</td>
<td>Returns the comparison operator associated with the parameter.</td>
</tr>
<tr>
<td>GetParamFieldType method</td>
<td>Returns the field type of the parameter.</td>
</tr>
<tr>
<td>GetParamLabel method</td>
<td>Returns the name of the parameter.</td>
</tr>
<tr>
<td>GetParamPrompt method</td>
<td>Returns the prompt string displayed to the user for the given parameter.</td>
</tr>
<tr>
<td>GetSQL method</td>
<td>Returns the SQL string that expresses the query.</td>
</tr>
<tr>
<td>LookupPrimaryEntityDefName method</td>
<td>Returns the name of the EntityDef object on which the query is based.</td>
</tr>
<tr>
<td>MoveNext method</td>
<td>Moves the cursor to the next record in the data set.</td>
</tr>
</tbody>
</table>

See Also:

BuildResultSet method of the Session object
QueryDef object
Session object
ResultSet Code Example
**AddParamValue method**

Assigns one or more values to a parameter.

**VB Syntax:**

```vbnet
resultset.AddParamValue param_number, value
```

**Perl Syntax:**

```perl
$resultset->AddParamValue(param_number, value);
```

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>resultset</td>
<td>A ResultSet object, representing the rows and columns of data resulting from a query.</td>
</tr>
<tr>
<td>param_number</td>
<td>A Long identifying the parameter. See Remarks.</td>
</tr>
<tr>
<td>value</td>
<td>A Variant containing one or more values for the parameter.</td>
</tr>
</tbody>
</table>

**Return value** None.

**Member of:** ResultSet Object

**Remarks:**

The parameter number is a Long whose value is between 1 and the total number of parameters.

**See Also:**

ClearParamValues method
**ClearParamValues method**

Clears all values associated with a parameter.

**VB Syntax:**

```vbnet
resultset.ClearParamValues param_number
```

**Perl Syntax:**

```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. See Remarks.
`Return value` | None.

**Member of:** ResultSet Object

**Remarks:**

The parameter number is a Long whose value is between 1 and the total number of parameters.

**See Also:**

AddParamValue method
Execute method

Runs the query and fills the result set with data.

**VB Syntax:**

`resultset.Execute`

**Perl Syntax:**

`$resultset->Execute();`

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>resultset</code></td>
<td>A ResultSet object, representing the rows and columns of data resulting from a query.</td>
</tr>
</tbody>
</table>

**Return value**  
None.

**Member of:** ResultSet Object

**Remarks:**

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.

**See Also:**

GetColumnValue method  
GetSQL method
MoveNext method
BuildResultSet method of the Session object
Session object
ResultSet Code Example
**GetColumnLabel method**

Returns the heading text for the specified column.

**VB Syntax:**

```vbnet
resultset.GetColumnLabel columnNum
```

**Perl Syntax:**

```perl
$resultset->GetColumnLabel(columnNum);
```

### Identifier Description

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>resultset</code></td>
<td>A ResultSet object, representing the rows and columns of data resulting from a query.</td>
</tr>
<tr>
<td><code>columnNum</code></td>
<td>A Long that specifies the desired index (1-based) into the array of columns.</td>
</tr>
</tbody>
</table>

**Return value**

A String containing the column label.

**Member of:** ResultSet Object

**Remarks:**

Columns are numbered from 1 to N, not 0 to N-1.

**See Also:**

GetColumnType method
GetColumnValue method
GetNumberOfColumns method
ResultSet Code Example
GetColumnType method

Returns the type of data stored in the specified column.

VB Syntax:

```vbnet
resultset.GetColumnType columnNum
```

Per1 Syntax:

```
$resultset->GetColumnType(columnNum);
```

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>resultset</code></td>
<td>A ResultSet object, representing the rows and columns of data resulting from a query.</td>
</tr>
<tr>
<td><code>columnNum</code></td>
<td>A Long that specifies the desired index (1-based) into the array of columns.</td>
</tr>
<tr>
<td><strong>Return value</strong></td>
<td>A Long whose value is a CType enumeration constant representing this column’s underlying storage type in the database.</td>
</tr>
</tbody>
</table>

**Member of:** ResultSet Object

**Remarks:**

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.

**See Also:**

GetColumnLabel method
GetColumnValue method
GetNumberOfColumns method
GetColumnValue method

Returns the value stored in the specified column of the current row.

**VB Syntax:**

```
resultset.GetColumnValue columnNum
```

**Perl Syntax:**

```
$resultset->GetColumnValue(columnNum);
```

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>resultset</code></td>
<td>A ResultSet object, representing the rows and columns of data resulting from a query.</td>
</tr>
<tr>
<td><code>columnNum</code></td>
<td>A Long that specifies the desired index (1-based) into the array of columns.</td>
</tr>
</tbody>
</table>

**Return value**

A Variant that contains the value stored in the specified column of the current row.

**Member of:** ResultSet Object

**Remarks:**

If the cursor is not positioned at a record, or the field’s value has not been set, the returned Variant will be NULL. To advance the cursor to the next row, you must call the MoveNext method.

In the current version of the ClearQuest API, the Variant is always set as a string, but future versions might let you initialize the Variant to the most appropriate native type.

Columns are numbered from 1 to N, not 0 to N-1.

**See Also:**

GetColumnLabel method
GetColumnType method
GetNumberOfColumns method
MoveNext method
ResultSet Code Example
**GetNumberOfColumns method**

Returns the number of columns in each row of the result set.

**VB Syntax:**

```vb
resultset.GetNumberOfColumns
```

**Perl Syntax:**

```perl
$resultset->GetNumberOfColumns();
```

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>resultset</code></td>
<td>A ResultSet object, representing the rows and columns of data resulting from a query.</td>
</tr>
<tr>
<td><strong>Return value</strong></td>
<td>A Long indicating the number of columns in the result set.</td>
</tr>
</tbody>
</table>

**Member of:** ResultSet Object

**See Also:**

- GetColumnLabel method
- GetColumnType method
- GetColumnValue method
- ResultSet Code Example
GetNumberOfParams method

Returns the number of parameters in this query.

**VB Syntax:**

```vbnet
resultset.GetNumberOfParams
```

**Perl Syntax:**

```perl
$resultset->GetNumberOfParams();
```

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>resultset</code></td>
<td>A ResultSet object, representing the rows and columns of data resulting from a query.</td>
</tr>
</tbody>
</table>

**Return value**

A Long indicating the number of parameters in the query.

**Member of:** ResultSet Object

**See Also:**

GetParamChoiceList method  
GetParamComparisonOperator method  
GetParamFieldType method  
GetParamLabel method  
GetParamPrompt method
GetParamChoiceList method

Returns a list of permitted values for the parameter.

**VB Syntax:**

`resultset.GetParamChoiceList param_number`

**Perl Syntax:**

```perl
$resultset->GetParamChoiceList(param_number);
```

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>resultset</code></td>
<td>A ResultSet object, representing the rows and columns of data resulting from a query.</td>
</tr>
<tr>
<td><code>param_number</code></td>
<td>A Long identifying the parameter. See Remarks.</td>
</tr>
<tr>
<td><strong>Return value</strong></td>
<td>A Variant containing the list of permitted values for the parameter. If the Variant is empty, there are no restrictions on the parameter values.</td>
</tr>
</tbody>
</table>

**Member of:** ResultSet Object

**Remarks:**

The parameter number is a Long whose value is between 1 and the total number of parameters.

**See Also:**

- GetNumberOfParams method
- GetParamComparisonOperator method
- GetParamFieldType method
- GetParamLabel method
- GetParamPrompt method
GetParamComparisonOperator method

Returns the comparison operator associated with the parameter.

**VB Syntax:**

```vbnet
resultset.GetParamComparisonOperator param_number
```

**Perl Syntax:**

```perl
$resultset->GetParamComparisonOperator($param_number);
```

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>resultset</td>
<td>A ResultSet object, representing the rows and columns of data resulting from a query.</td>
</tr>
<tr>
<td>param_number</td>
<td>A Long identifying the parameter. See Remarks.</td>
</tr>
<tr>
<td>Return value</td>
<td>A Long indicating the comparison operator for the parameter. The value corresponds to a value in the CompOp enumerated type.</td>
</tr>
</tbody>
</table>

**Member of:** ResultSet Object

**Remarks:**

The parameter number is a Long whose value is between 1 and the total number of parameters.

**See Also:**

GetNumberOfParams method
GetParamChoiceList method
GetParamFieldType method
GetParamLabel method
GetParamPrompt method
CompOp enumerated type
GetParamFieldMethod method

Returns the field type of the parameter.

**VB Syntax:**

```vbnet
resultset.GetParamFieldMethod param_number
```

**Perl Syntax:**

```perl
$resultset->GetParamFieldMethod(param_number);
```

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>resultset</code></td>
<td>A ResultSet object, representing the rows and columns of data resulting from a query.</td>
</tr>
<tr>
<td><code>param_number</code></td>
<td>A Long identifying the parameter. See Remarks.</td>
</tr>
</tbody>
</table>

**Return value**

A Long indicating the field type of the parameter. The value corresponds to a value in the FieldType enumerated type.

**Member of:** ResultSet Object

**Remarks:**

The parameter number is a Long whose value is between 1 and the total number of parameters.

**See Also:**

GetNumberOfParams method
GetParamChoiceList method
GetParamComparisonOperator method
GetParamLabel method
GetParamPrompt method
FieldType enumerated type
GetParamLabel method

Returns the name of the parameter.

**VB Syntax:**

```vbnet
resultset.GetParamLabel param_number
```

**Perl Syntax:**

```perl
$resultset->GetParamLabel($param_number);
```

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>resultset</td>
<td>A ResultSet object, representing the rows and columns of data resulting from a query.</td>
</tr>
<tr>
<td>param_number</td>
<td>A Long identifying the parameter. See Remarks.</td>
</tr>
<tr>
<td>return value</td>
<td>A String containing the name of the parameter.</td>
</tr>
</tbody>
</table>

**Member of:** ResultSet Object

**Remarks:**

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.

**See Also:**

GetNumberOfParams method
GetParamChoiceList method
GetParamComparisonOperator method
GetParamFieldType method
GetParamPrompt method
GetParamPrompt method

Returns the prompt string displayed to the user for the given parameter.

VB Syntax:

`resultset.GetParamPrompt param_number`

Perl Syntax:

`$resultset->GetParamPrompt(param_number);`

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>resultset</td>
<td>A ResultSet object, representing the rows and columns of data resulting from a query.</td>
</tr>
<tr>
<td>param_number</td>
<td>A Long identifying the parameter. See Remarks.</td>
</tr>
<tr>
<td>Return value</td>
<td>A String containing the prompt string displayed to the user.</td>
</tr>
</tbody>
</table>

Member of: ResultSet Object

Remarks:

The parameter number is a Long whose value is between 1 and the total number of parameters.

See Also:

GetNumberOfParams method
GetParamChoiceList method
GetParamComparisonOperator method
GetParamField method
GetParamLabel method
GetRowEntityDefName method

Based on the result set, this method returns a String with the record type (EntityDef) name of the current row.

**VB Syntax:**

```vbnet
resultset.GetRowEntityDefName
```

**Perl Syntax:**

```perl
$resultset->GetRowEntityDefName();
```

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>resultset</code></td>
<td>A ResultSet object, representing the rows of data that meet query criteria.</td>
</tr>
</tbody>
</table>

**Return value**

A String containing the name of the record type of the specified row.

**Member of:** ResultSet Object

**Remarks:**

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.

**Examples:**

See *Running a query against more than one record type (multitype query)*.

**See Also:**

IsMultiType method
GetSQL method

Returns the SQL string that expresses the query.

**VB Syntax:**

`resultset.GetSQL`

**Perl Syntax:**

`$resultset->GetSQL();`

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>resultset</code></td>
<td>A ResultSet object, representing the rows and columns of data resulting from a query.</td>
</tr>
<tr>
<td><code>Return value</code></td>
<td>A String containing the raw SQL that expresses the query upon which this ResultSet is based.</td>
</tr>
</tbody>
</table>

**Member of:** ResultSet Object

**Remarks:**

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.

**See Also:**

Execute method
LookupPrimaryEntityDefName method

Returns the name of the EntityDef object on which the query is based.

**VB Syntax:**

```
resultset.LookupPrimaryEntityDefName
```

**Perl Syntax:**

```
$resultset->LookupPrimaryEntityDefName();
```

**Identifier** | **Description**  
---|---  
`resultset` | A ResultSet object, representing the rows and columns of data resulting from a query.  

**Return value** | A String containing the name of the EntityDef object. If the query was defined using the BuildSQLQuery method of Session, the resulting String is Empty.  

**Member of:** ResultSet Object

**Remarks:**

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.

**See Also:**

BuildQuery method of the Session object  
EntityDef object  
QueryDef object  
Session object  
ResultSet Code Example
MoveNext method

Moves the cursor to the next record in the data set.

**VB Syntax:**

```vbnet
resultset.MoveNext
```

**Perl Syntax:**

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

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>resultset</code></td>
<td>A ResultSet object, representing the rows and columns of data resulting from a query.</td>
</tr>
<tr>
<td><code>Return value</code></td>
<td>A Long whose value is a FetchStatus enumeration constant indicating whether the cursor movement was successful.</td>
</tr>
</tbody>
</table>

**Member of:** ResultSet Object

**Remarks:**

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.

**See Also:**

Execute method
GetColumnValue method
ResultSet Code Example
QueryFilterNode object

**QueryFilterNode methods**

<table>
<thead>
<tr>
<th>Method name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BuildFilter method</td>
<td>Adds a comparison operand to the node.</td>
</tr>
<tr>
<td>BuildFilterOperator method</td>
<td>Creates a nested QueryFilterNode object that contains the specified Boolean operator.</td>
</tr>
</tbody>
</table>

**See Also:**

BuildQuery method of the Session object
ResultSet Object
QueryDef object
Session object
BuildQuery Code Example
BuildFilter method

Adds a comparison operand to the node.

**VB Syntax:**

```vb
node.BuildFilter field_name, comparison_operator, value
```

**Perl Syntax:**

```perl
$node->BuildFilter(field_name, comparison_operator, value);
```

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>node</td>
<td>A QueryFilterNode object, representing one node in the query expression.</td>
</tr>
<tr>
<td>field_name</td>
<td>A String containing the name of a valid field in the EntityDef object on which the current QueryDef object is based.</td>
</tr>
<tr>
<td>comparison_operator</td>
<td>A Long whose value is one of the CompOp enumeration constants.</td>
</tr>
<tr>
<td>value</td>
<td>[A VB Variant or] Perl reference to a string array containing the value that you want to compare to the value in the specified field.</td>
</tr>
</tbody>
</table>

Return value : None.

**Member of:** QueryFilterNode object

**Remarks:**

The operand created by this method consists of a field name, a comparison operator, and a value. When the query is run, the value in the field is compared to the specified value using the given comparison operator. The comparison yields a boolean value, which the node uses in its own boolean comparison.

The value argument is a [VB Variant or] Perl reference to a string array to allow you to specify an Array of values when appropriate. For example, if you wanted to find the defects submitted between December 1 and December 15, 2000, you could construct the following filter:
VB

Dim dateRange as Variant(2)
dateRange(0) = 2000-12-01
dateRange(1) = 2000-12-15
node.BuildFilter("submit_date", AD_COMP_OP_IN, dateRange)

Perl

@dateRange = ("2000-12-01", "2000-12-15");
$node->BuildFilter("submit_date", CQPerlExt::COMP_OP_IN, \@dateRange);

Query expressions are not limited to being binary trees; you can call this method as
many times as you want for a given QueryFilterNode object. See the example given for
QueryDef's BuildFilterOperator method.

To obtain valid values for the field_name argument, call the GetFieldDefNames method
of the EntityDef object upon which the query was based.

See Also:

BuildFilterOperator method
BuildFilterOperator method of the QueryDef object
QueryDef object
BuildQuery Code Example
Notation Conventions
BuildFilterOperator method

Creates a nested QueryFilterNode object that contains the specified Boolean operator.

VB Syntax:

`node.BuildFilterOperator bool_operator`

Perl Syntax:

`$node->BuildFilterOperator(bool_operator);`

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>node</code></td>
<td>The QueryFilterNode object to which the newly created node will be attached.</td>
</tr>
<tr>
<td><code>bool_operator</code></td>
<td>A Long whose value is one of the BoolOp enumeration constants.</td>
</tr>
</tbody>
</table>

Return value

The newly created QueryFilterNode object.

Member of: QueryFilterNode object

Remarks:

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.

See Also:

BuildFilter method
BuildFilterOperator method of the QueryDef object
QueryDef object
BuildQuery Code Example
AdminSession object

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

Remarks:

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:

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

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

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

Working With Databases

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

<table>
<thead>
<tr>
<th>Property name</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Databases property</td>
<td>Read-only</td>
<td>Returns the collection of databases associated with the schema repository.</td>
</tr>
<tr>
<td>Groups property</td>
<td>Read-only</td>
<td>Returns the collection of groups associated with the schema repository.</td>
</tr>
<tr>
<td>Schemas property</td>
<td>Read-only</td>
<td>Returns the collection of schemas associated with the schema repository.</td>
</tr>
<tr>
<td>Users property</td>
<td>Read-only</td>
<td>Returns the collection of users associated with the schema repository.</td>
</tr>
</tbody>
</table>

Methods

<table>
<thead>
<tr>
<th>Method name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CreateDatabase method</td>
<td>Creates a new database and associates it with the schema repository.</td>
</tr>
<tr>
<td>CreateGroup method</td>
<td>Creates a new group and associates it with the schema repository.</td>
</tr>
<tr>
<td>CreateUser method</td>
<td>Creates a new user and associates it with the schema repository.</td>
</tr>
<tr>
<td>DeleteDatabase method</td>
<td>Disassociates the specified database from the schema repository.</td>
</tr>
<tr>
<td>GetDatabase method</td>
<td>Returns the database object with the specified name.</td>
</tr>
<tr>
<td>GetGroup method</td>
<td>Returns the group object with the specified name.</td>
</tr>
<tr>
<td>GetUser method</td>
<td>Returns the user object with the specified name.</td>
</tr>
<tr>
<td>Logon method</td>
<td>Logs the specified user into the schema repository.</td>
</tr>
</tbody>
</table>

See Also:

Session object
**Databases property**

Returns the collection of databases associated with the schema repository.

**VB Syntax:**

```vbnet
adminSession.Databases
```

**Perl Syntax:**

```perl
@adminSession-&gt;GetDatabases();
```

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>adminSession</td>
<td>The AdminSession object representing the current schema repository access session.</td>
</tr>
<tr>
<td>Return value</td>
<td>A Databases collection object containing the collection of all databases defined in this schema repository.</td>
</tr>
</tbody>
</table>

**Member of:** AdminSession object

**Remarks:**

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

Each element in the returned collection is a Database object.

**Examples:**

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

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

```perl
set @adminSession-&gt;GetDatabases();
```
See Also:

Database object
Databases collection object
Groups property

Returns the collection of groups associated with the schema repository.

**VB Syntax:**

`adminSession.Groups`

**Perl Syntax:**

`$adminSession-&gt;GetGroups();`

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>adminSession</code></td>
<td>The AdminSession object representing the current schema repository access session.</td>
</tr>
</tbody>
</table>

**Return value**

A Groups collection object containing all of the groups in the schema repository.

**Member of:** AdminSession object

**Remarks:**

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

Each element in the returned collection is a Group object.

**Examples:**

```vbscript
set adminSession = CreateObject("ClearQuest.AdminSession")
adminSession.Logon "admin", "admin", ""  
set groupList = adminSession.Groups
numGroups = groupList.Count
For x = 0 to numGroups
    set groupObj= groupList.Item(x)
    groupName = groupObj.Name
    OutputDebugString "Found group: " & groupName
Next
```
See Also:

Group object
Groups collection object
Schemas property

Returns the collection of schemas associated with the schema repository.

**VB Syntax:**

`adminSession.Schemas`

**Perl Syntax:**

`$adminSession->GetSchemas();`

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>adminSession</code></td>
<td>The AdminSession object representing the current schema repository access session.</td>
</tr>
<tr>
<td><strong>Return value</strong></td>
<td>A Schemas collection object containing all of the schemas in the master database.</td>
</tr>
</tbody>
</table>

**Member of:** AdminSession object

**Remarks:**

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

Each element in the returned collection is a Schema object.

**Examples:**

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

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

Schema object
Schemas collection object
Users property

Returns the collection of users associated with the schema repository.

**VB Syntax:**

```vbnet
adminSession.Users
```

**Peri Syntax:**

```perl
adminSession->GetUsers();
```

---

**Identifier** | **Description**
--- | ---
`adminSession` | The AdminSession object representing the current schema repository access session.

**Return value** | A Users collection object containing all of the users in the schema repository.

**Member of:** AdminSession object

**Remarks:**

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

Each element in the returned collection is a User object.

**Examples:**

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

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

User object
Users collection object
CreateDatabase method

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

**VB Syntax:**

```vbnet
adminSession.CreateDatabase databaseName
```

**Perl Syntax:**

```perl
$adminSession->CreateDatabase(databaseName);
```

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>adminSession</code></td>
<td>The AdminSession object representing the current schema repository access session.</td>
</tr>
<tr>
<td><code>databaseName</code></td>
<td>A String containing the name you want to give to the new database.</td>
</tr>
<tr>
<td><code>Return value</code></td>
<td>A Database object representing the new database.</td>
</tr>
</tbody>
</table>

**Member of:** AdminSession object

**Remarks:**

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

**Examples:**

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

call newDatabaseObj = adminSession.CreateDatabase "NEWDB"
```
See Also:

DeleteDatabase method
GetDatabase method
Databases property
SetInitialSchemaRev method of the Database object
Database object
CreateGroup method

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

**VB Syntax:**

```vbnet
adminSession.CreateGroup groupName
```

**Perl Syntax:**

```perl
$adminSession->CreateGroup(groupName);
```

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>adminSession</td>
<td>The AdminSession object representing the current schema repository access session.</td>
</tr>
<tr>
<td>groupName</td>
<td>A String containing the name you want to give to the new group.</td>
</tr>
<tr>
<td>Return value</td>
<td>A new Group object.</td>
</tr>
</tbody>
</table>

**Member of:** AdminSession object

**Remarks:**

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.

**Examples:**

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

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

**See Also:**

GetGroup method
Groups property
Group object
CreateUser method

Creates a new user and associates it with the schema repository

**VB Syntax:**

\[\text{adminSession}.\text{CreateUser}\ \text{userName}\]

**Perl Syntax:**

\[$\text{adminSession} \rightarrow \text{CreateUser}(\text{userName});\]

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>adminSession</td>
<td>The AdminSession object representing the current schema repository access session.</td>
</tr>
<tr>
<td>userName</td>
<td>A String containing the name you want to give to the new user.</td>
</tr>
<tr>
<td>Return value</td>
<td>A new User object.</td>
</tr>
</tbody>
</table>

**Member of:** AdminSession object

**Remarks:**

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.

**Examples:**

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

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

**See Also:**

GetUser method
Users property
User object
DeleteDatabase method

Disassociates the specified database from the schema repository.

**VB Syntax:**

```vbnet
adminSession.DeleteDatabase databaseName
```

**Perl Syntax:**

```perl
$adminSession->DeleteDatabase(databaseName);
```

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>adminSession</td>
<td>The AdminSession object representing the current schema repository access session.</td>
</tr>
<tr>
<td>databaseName</td>
<td>A String containing the name of the database you want to delete.</td>
</tr>
<tr>
<td>Return value</td>
<td>The Database object that was disassociated from the schema repository.</td>
</tr>
</tbody>
</table>

**Member of:** AdminSession object

**Remarks:**

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

**Examples:**

```plaintext
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"
```
See Also:

CreateDatabase method
GetDatabase method
Databases property
Database object
GetDatabase method

Returns the database object with the specified name.

**VB Syntax:**

```vbnet
adminSession.GetDatabase databaseName
```

**Perl Syntax:**

```perl
$adminSession->GetDatabase(databaseName);
```

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>adminSession</code></td>
<td>The AdminSession object representing the current schema repository access session.</td>
</tr>
<tr>
<td><code>databaseName</code></td>
<td>A String containing the name of the database object you want.</td>
</tr>
<tr>
<td><strong>Return value</strong></td>
<td>The Database object with the specified name, or null if no such database exists.</td>
</tr>
</tbody>
</table>

**Member of:** AdminSession object

**Remarks:**

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

**Examples:**

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

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

**See Also:**

CreateDatabase method  
Databases property
Name property of the Database object
Database object
GetGroup method

Returns the group object with the specified name.

**VB Syntax:**

```vbnet
adminSession.GetGroup groupName
```

**Perl Syntax:**

```perl
$adminSession->GetGroup(groupName);
```

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>adminSession</code></td>
<td>The AdminSession object representing the current schema repository access session.</td>
</tr>
<tr>
<td><code>groupName</code></td>
<td>A String containing the name of the database object you want.</td>
</tr>
<tr>
<td><strong>Return value</strong></td>
<td>The Group object with the specified name, or null if no such group exists.</td>
</tr>
</tbody>
</table>

**Member of:** AdminSession object

**Remarks:**

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

**Examples:**

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

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

**See Also:**

CreateGroup method
Groups property
Name property of the Group object
Group object
**GetUser method**

Returns the user object with the specified name.

**VBVB Syntax:**

\`
adminSession.GetUser userName
\`

**Perl Syntax:**

\`
$adminSession->GetUser(userName);    
\`

**Member of:** AdminSession object

**Remarks:**

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

**Examples:**

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

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

**See Also:**

CreateUser method

Users property
Name property of the User object
User object
Logon method

Logs the specified user into the schema repository.

**VB Syntax:**

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

**Perl Syntax:**

```perl
$actionSession->Logon(login_name, password, databaseSetName);
```

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>adminSession</td>
<td>The AdminSession object representing the current schema repository access session.</td>
</tr>
<tr>
<td>login_name</td>
<td>A String that specifies the login name of the user.</td>
</tr>
<tr>
<td>password</td>
<td>A String that specifies the user's password.</td>
</tr>
<tr>
<td>databaseSetName</td>
<td>A String that specifies the name of the schema repository. Normally, you should set this string to the empty string (&quot;&quot;).</td>
</tr>
</tbody>
</table>

**Return value**

None.

**Member of:** AdminSession object

**Remarks:**

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

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

**See Also:**

- CreateUser method
- Users property
- UserLogon method of the Session object
- Session object
Database object

**Database object properties**

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

<table>
<thead>
<tr>
<th>Method name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ApplyPropertyChanges</td>
<td>Updates the database’s writable properties with any recent changes.</td>
</tr>
<tr>
<td>SetInitialSchemaRev</td>
<td>Sets the initial schema revision of a new database.</td>
</tr>
<tr>
<td>Upgrade</td>
<td>Upgrade this database to the specified schema revision.</td>
</tr>
<tr>
<td>UpgradeMasterUserInfo</td>
<td>Upgrade this database’s user information.</td>
</tr>
</tbody>
</table>

### See Also:

CreateDatabase method of the AdminSession object
AdminSession object
Schema object
SchemaRev object

<table>
<thead>
<tr>
<th>Property name</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SubscribedGroups</td>
<td>Read-only</td>
<td>Returns the groups that are explicitly subscribed to this database.</td>
</tr>
<tr>
<td>SubscribedUsers</td>
<td>Read-only</td>
<td>Returns the users that are explicitly subscribed to this database.</td>
</tr>
<tr>
<td>TimeoutInterval</td>
<td>Read/Write</td>
<td>Returns the user timeout interval.</td>
</tr>
<tr>
<td>Vendor</td>
<td>Read/Write</td>
<td>Returns the vendor type of the database.</td>
</tr>
</tbody>
</table>
CheckTimeoutInterval property

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

**VB Syntax:**

```vbnet
database.CheckTimeoutInterval [= value]
```

**Perl Syntax:**

```perl
$database->GetCheckTimeoutInterval();
$database->SetCheckTimeoutInterval(newValue);
```

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>database</td>
<td>A Database object.</td>
</tr>
<tr>
<td>value</td>
<td>A Long value indicating the number of milliseconds between checks.</td>
</tr>
</tbody>
</table>

**Member of:** Database object

**Remarks:**

This property can be returned or set.

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.

**See Also:**

TimeoutInterval property
**ConnectHosts property**

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

**VB Syntax:**

`database.ConnectHosts [= value]`

**Perl Syntax:**

```perl
$database->GetConnectHosts();
$database->SetConnectHosts(string_for_the_host_connection);
```

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>database</code></td>
<td>A Database object.</td>
</tr>
<tr>
<td><code>value</code></td>
<td>[A VB array of Strings or] a Perl reference to a string array. Each String in the array contains a physical location of a database server.</td>
</tr>
</tbody>
</table>

**Member of:** Database object

**Remarks:**

This property can be returned or set. This property is used only in conjunction with databases who Vendor property is SQL ANYWHERE. This property corresponds to the SQL Anywhere HOST database server option.

**See Also:**

ConnectProtocols property
ConnectProtocols property

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

**VB Syntax:**

```vb
database.ConnectProtocols [= value]
```

**Perl Syntax:**

```perl
$database->GetConnectProtocols();
$database->SetConnectProtocols(string_for_the_host_connection);
```

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>database</td>
<td>A Database object.</td>
</tr>
<tr>
<td>value</td>
<td>[A VB array of Strings or] a Perl reference to an array of Strings. Each String in the array contains the name of a network protocol for the database server.</td>
</tr>
</tbody>
</table>

**Member of:** Database object

**Remarks:**

This property can be returned or set. This property is used only in conjunction with databases who Vendor property is SQL_ANYWHERE. This property corresponds to the SQL Anywhere \(-x\) database server option.

**See Also:**

ConnectHosts property
DatabaseName property

Sets or returns the physical name of the database.

**VB Syntax:**

```vbnet
database.DatabaseName [= value]
```

**Perl Syntax:**

```perl
$database->GetDatabaseName();
$database->SetDatabaseName(string_for_physical_database_name);
```

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>database</td>
<td>A Database object.</td>
</tr>
<tr>
<td>value</td>
<td>A String containing the file name of the database, including any associated path information.</td>
</tr>
</tbody>
</table>

**Member of:** Database object

**Remarks:**

This property can be returned or set.

**See Also:**

Name property
DBOLogin property

Sets or returns the database owner’s login name.

**VB Syntax:**

```vbnet
database.DBOLogin [= value]
```

**Perl Syntax:**

```perl
$database->GetDBOLogin();
$database->SetDBOLogin(string_for_db-owner_login_name);
```

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>database</td>
<td>A Database object.</td>
</tr>
<tr>
<td>value</td>
<td>A String containing the database owner’s login name.</td>
</tr>
</tbody>
</table>

**Member of:** Database object

**Remarks:**

This property can be returned or set. The database owner is the same as the database administrator. This property is used primarily in conjunction with SQL Server databases.

**See Also:**

DBOPassword property
**DBOPassword property**

Sets or returns the database owner’s password.

**VB Syntax:**

```vbnet
database.DBOPassword [= value]
```

**Perl Syntax:**

```perl
$database->GetDBOPassword();
$database->SetDBOPassword(string_for_db-owner_password);
```

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>database</code></td>
<td>A Database object.</td>
</tr>
<tr>
<td><code>value</code></td>
<td>A String containing the database owner's password.</td>
</tr>
</tbody>
</table>

**Member of:** Database object

**Remarks:**

This property can be returned or set. The database owner is the same as the database administrator. This property is used primarily in conjunction with SQL Server databases.

**See Also:**

DBOLogin property
Description property

Sets or returns the descriptive comment associated with the database.

**VB Syntax:**

```vbnet
database.Description [= value]
```

**Perl Syntax:**

```perl
$database->GetDescription();
$database->SetDescription(string_describing_database);
```

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>database</td>
<td>A Database object.</td>
</tr>
<tr>
<td>value</td>
<td>A String containing the database’s comment text.</td>
</tr>
</tbody>
</table>

**Member of:** Database object

**Remarks:**

This property can be returned or set.

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

**See Also:**

Name property
Name property

Sets or returns the logical database name.

**VB Syntax:**

\[\text{database}.\text{Name} \ [= \text{value}]\]

**Perl Syntax:**

\$database->\text{Name}();
\$database->\text{Name}(\text{string_for_database_name});

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>database</td>
<td>A Database object.</td>
</tr>
<tr>
<td>value</td>
<td>A String containing the logical database name.</td>
</tr>
</tbody>
</table>

**Member of:** Database object

**Remarks:**

This property can be returned or set.

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.

**See Also:**

DatabaseName property
ROLogin property

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

**VB Syntax:**

database.ROLogin [= value]

**Perl Syntax:**

$database->GetROLogin();
$database->SetROLogin(string_for_read-only_login_name);

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>database</td>
<td>A Database object.</td>
</tr>
<tr>
<td>value</td>
<td>A String containing the read-only login name.</td>
</tr>
</tbody>
</table>

**Member of:** Database object

**Remarks:**

This property can be returned or set. This property is used only in conjunction with databases whose Vendor property is SQL_SERVER.

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

**See Also:**

ROPassword property
RWLogin property
Vendor property
Notation Conventions
**ROPassword property**

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

**VB Syntax:**

```vbnet
database.ROPASSWORD [= value]
```

**Perl Syntax:**

```perl
$database->GetROPassword();
$database->SetROPassword(string_for_read-only_password);
```

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>database</code></td>
<td>A Database object.</td>
</tr>
<tr>
<td><code>value</code></td>
<td>A String containing the read-only password.</td>
</tr>
</tbody>
</table>

**Member of:** Database object

**Remarks:**

This property can be returned or set. This property is used only in conjunction with databases whose Vendor property is set to SQL_SERVER.

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

**See Also:**

- ROLogin property
- RWPassword property
- Vendor property
- Notation Conventions
**RWLogin property**

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

**VB Syntax:**

```
database.RWLogin [= value]
```

**Perl Syntax:**

```
$database->GetRWLogin();
$database->SetRWLogin(string_for_read-write_login_name);
```

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>database</td>
<td>A Database object.</td>
</tr>
<tr>
<td>value</td>
<td>A String containing the read/write login name.</td>
</tr>
</tbody>
</table>

**Member of:** Database object

**Remarks:**

This property can be returned or set. This property is used in conjunction with SQL Server and SQL Anywhere databases.

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

**See Also:**

ROLLogin property
RWPassword property
Vendor property
RWPassword property

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

VB Syntax:

database.RWPassword [= value]

Perl Syntax:

$database->GetRWPassword();
$database->SetRWPassword(string_for_read-write-user_password);

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>database</td>
<td>A Database object.</td>
</tr>
<tr>
<td>value</td>
<td>A String containing the read/write password name.</td>
</tr>
</tbody>
</table>

Member of: Database object

Remarks:

This property can be returned or set. This property is used in conjunction with SQL Server and SQL Anywhere databases.

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

See Also:

ROPassword property
RWLogin property
Vendor property
SchemaRev property

Returns the schema revision currently in use by the database.

**VB Syntax:**

```vbnet
database.SchemaRev
```

**Perl Syntax:**

```perl
$database->GetSchemaRev();
```

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>database</code></td>
<td>A Database object.</td>
</tr>
<tr>
<td><strong>Return value</strong></td>
<td>An SchemaRev object corresponding to the schema revision in use by this database.</td>
</tr>
</tbody>
</table>

**Member of:** Database object

**Remarks:**

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.

**See Also:**

SetInitialSchemaRev method
Upgrade method
SchemaRev object
Server property

Returns the name of the server on which the database resides.

**VB Syntax:**

\[\text{database.Server} [= \text{value}]\]

**Perl Syntax:**

\$database->GetServer();
\$database->SetServer(\text{string_for_database_server_name});

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>database</td>
<td>A Database object.</td>
</tr>
<tr>
<td>value</td>
<td>A String containing the name of the server.</td>
</tr>
</tbody>
</table>

**Member of:** Database object

**Remarks:**

This property can be returned or set.

**See Also:**

DatabaseName property
SubscribedGroups property

Returns the groups explicitly subscribed to this database.

**VB Syntax:**

`database.SubscribedGroups`

**Perl Syntax:**

`$database->GetSubscribedGroups();`

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>database</code></td>
<td>A Database object.</td>
</tr>
<tr>
<td><strong>Return value</strong></td>
<td>A Groups collection object containing the groups explicitly subscribed to this database.</td>
</tr>
</tbody>
</table>

**Member of:** Database object

**Remarks:**

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.

**See Also:**

Group object
Groups collection object
SubscribedUsers property

Returns the users that are explicitly subscribed to this database.

**VB Syntax:**

```
database.SubscribedUsers
```

**Perl Syntax:**

```
$database->GetSubscribedUsers();
```

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>database</td>
<td>A Database object.</td>
</tr>
<tr>
<td>Return value</td>
<td>A Users collection object containing the users explicitly subscribed to this database.</td>
</tr>
</tbody>
</table>

**Member of:** Database object

**Remarks:**

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.

**See Also:**

User object
Users collection object
**TimeoutInterval property**

Returns the user timeout interval.

**VB Syntax:**

```vb
database. TimeoutInterval [= value]
```

**Perl Syntax:**

```perl
$database->GetTimeoutInterval();
$database->SetTimeoutInterval(timeout_inverval);
```

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>database</td>
<td>A Database object.</td>
</tr>
<tr>
<td>value</td>
<td>A Long value indicating the number of milliseconds a user may be idle before being disconnected from the database.</td>
</tr>
</tbody>
</table>

**Member of:** Database object

**Remarks:**

This property can be returned or set.

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.

**See Also:**

CheckTimeoutInterval property
Vendor property

Sets or returns the vendor type of the database.

**VB Syntax:**

```vbnet
database.Vendor [= value]
```

**Perl Syntax:**

```perl
$database->GetVendor();
$database->SetVendor(constant_for_a_database_vendor);
```

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>database</td>
<td>A Database object.</td>
</tr>
<tr>
<td>value</td>
<td>A Short containing one of the DatabaseVendor enumerated constants.</td>
</tr>
</tbody>
</table>

**Member of:** Database object

**Remarks:**

This property can be returned or set.

**See Also:**

DatabaseVendor
Enumerated Constants
ApplyPropertyChanges method

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

**VB Syntax:**

`database.ApplyPropertyChanges`

**Perl Syntax:**

`$database->ApplyPropertyChanges();`

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>database</code></td>
<td>A Database object.</td>
</tr>
<tr>
<td><code>Return value</code></td>
<td>None.</td>
</tr>
</tbody>
</table>

**Member of:** Database object

**Remarks:**

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

**See Also:**

SetInitialSchemaRev method  
Upgrade method
SetInitialSchemaRev method

Sets the initial schema revision of a new database.

**VB Syntax:**

database.SetInitialSchemaRev schemaRev

**Perl Syntax:**

$database->SetInitialSchemaRev(schemaRev);

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>database</td>
<td>A Database object.</td>
</tr>
<tr>
<td>schemaRev</td>
<td>A SchemaRev object corresponding to the desired schema revision.</td>
</tr>
</tbody>
</table>

**Return value** None.

**Member of:** Database object

**Remarks:**

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.

**See Also:**

Upgrade method  
SchemaRev property
Upgrade method

Upgrade this database to the specified schema revision.

**VB Syntax:**

```vbnet
database.Upgrade schemaRev
```

**Perl Syntax:**

```perl
$database->Upgrade(schemaRev);
```

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>database</td>
<td>A Database object.</td>
</tr>
<tr>
<td>schemaRev</td>
<td>A SchemaRev object corresponding to the desired schema revision.</td>
</tr>
<tr>
<td>Return value</td>
<td>None.</td>
</tr>
</tbody>
</table>

**Member of:** Database object

**Remarks:**

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.

**See Also:**

SetInitialSchemaRev method
UpgradeMasterUserInfo method
SchemaRev property
UpgradeMasterUserInfo method

Upgrade this database’s user information.

VB Syntax:

database.UpgradeMasterUserInfo

Perl Syntax:

$database->UpgradeMasterUserInfo();

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>database</td>
<td>A Database object.</td>
</tr>
</tbody>
</table>

Return value: None.

Member of: Database object

Remarks:

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

See Also:

Upgrade method
## Schema Object Properties

<table>
<thead>
<tr>
<th>Property name</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name property</td>
<td>Read-only</td>
<td>Returns a string containing the name of the schema.</td>
</tr>
<tr>
<td>SchemaRevs property</td>
<td>Read-only</td>
<td>Returns the collection containing schema revisions.</td>
</tr>
</tbody>
</table>
Name property

Returns the name of this schema.

**VBSyntax:**

`schema.Name`

**Perl Syntax:**

`$schema->GetName();`

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>schema</td>
<td>A Schema object.</td>
</tr>
</tbody>
</table>

*Return value* A String containing the name of this schema.

**Member of:** Schema object

**Remarks:**

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

**See Also:**

SchemaRevs property
SchemaRevs property

Returns the schema revisions associated with this schema.

**VBSyntax:**

```
schema.SchemaRevs
```

**Perl Syntax:**

```
$schema->GetSchemaRevs();
```

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>schema</code></td>
<td>A Schema object.</td>
</tr>
<tr>
<td><code>Return value</code></td>
<td>A SchemaRevs collection object containing the schema revisions associated with this schema.</td>
</tr>
</tbody>
</table>

**Member of:** Schema object

**Remarks:**

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

Each element in the returned collection is a SchemaRev object.

**See Also:**

SchemaRev object
SchemaRev object

**SchemaRev object properties**

<table>
<thead>
<tr>
<th>Property name</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description property</td>
<td>Read-only</td>
<td>Returns a description of this schema revision.</td>
</tr>
<tr>
<td>RevID property</td>
<td>Read-only</td>
<td>Returns the version ID of this schema revision.</td>
</tr>
<tr>
<td>Schema property</td>
<td>Read-only</td>
<td>Returns the schema to which this revision belongs.</td>
</tr>
</tbody>
</table>

**SchemaRev object methods**

<table>
<thead>
<tr>
<th>Method name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetEnabledEntityDefs method</td>
<td>Returns the EntityDefs collection object enabled in the current schema for a given package revision.</td>
</tr>
<tr>
<td>GetEnabledPackageRevs method</td>
<td>Returns the package name and revision string for the current package revision in an EntityDefs collection object.</td>
</tr>
</tbody>
</table>

**See Also:**

SetInitialSchemaRev method of the Database object
Upgrade method of the Database object
Database object
Schema object
Description property

Returns a description of this schema revision.

VBSyntax:

\texttt{schemaRev.Description}

Perl Syntax:

\texttt{$schemaRev->GetDescription();}

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{schemaRev}</td>
<td>A SchemaRev object.</td>
</tr>
<tr>
<td>\texttt{Return value}</td>
<td>A String containing the description of the schema revision.</td>
</tr>
</tbody>
</table>

Member of: SchemaRev object

Remarks:

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.

See Also:

Schema object
Revid property

Returns the version number of this schema revision.

**VBSyntax:**

```
schemaRev.Revid
```

**Perl Syntax:**

```
$schemaRev->GetRevID();
```

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Return value</strong></td>
<td>A Long indicating the version number associated with this schema revision.</td>
</tr>
</tbody>
</table>

**Member of:** SchemaRev object

**Remarks:**

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

**See Also:**

Schema object
Schema property

Returns the schema to which this revision belongs.

**VBSyntax:**

`schemaRev.Schema`

**Perl Syntax:**

`$schemaRev->GetSchema();`

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Return value</code></td>
<td>A Schema object corresponding to the schema to which this revision belongs.</td>
</tr>
</tbody>
</table>

**Member of:** SchemaRev object

**Remarks:**

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

**See Also:**

Schema object
GetEnabledEntityDefs method

Returns the EntityDefs collection object enabled in the current schema for a given package revision.

VB Syntax:

\texttt{schemaRev.GetEnabledEntityDefs packName, rev}

Perl Syntax:

\texttt{$schemaRev->GetEnabledEntityDefs(packName, rev):}

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>packName</td>
<td>A String that specifies the package name.</td>
</tr>
<tr>
<td>rev</td>
<td>A String that specifies the package revision.</td>
</tr>
</tbody>
</table>

\textit{Return value} The EntityDefs object for the current package revision.

\textbf{Member of:} SchemaRev object, Session object

\textbf{Remarks:}

Use with GetEnabledPackageRevs method to discover which packages and package revisions apply to the current user database.

\textbf{See Also:}

GetEnabledPackageRevs method
GetEnabledPackageRevs method

Returns a collection object representing the packageRev set that is enabled in the current revision of the schema.

**VB Syntax:**

```vbnet
schemaRev.GetEnabledPackageRevs PackageName, RevString
```

**Perl Syntax:**

```perl
$schemaRev->GetEnabledPackageDefs(PackageName, RevString);
```

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PackageName</td>
<td>Name of the package.</td>
</tr>
<tr>
<td>RevString</td>
<td>Represents the revision of the package.</td>
</tr>
<tr>
<td><strong>Return values</strong></td>
<td>The collection object of the packageRev set.</td>
</tr>
</tbody>
</table>

**Member of:** SchemaRev object, Session object

**Remarks:**

Use with GetEnabledEntityDefs method to discover which packages and package revisions apply to the current user database.

You can also call this method from the Session object, in which case the schema revision is already known when you log onto the user database.

**See Also:**

GetEnabledEntityDefs method
GetEnabledPackageRevs method in Session object
## User object

### User object properties

<table>
<thead>
<tr>
<th>Property name</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Active property</td>
<td>Read/Write</td>
<td>Indicates whether or not the user's account is active.</td>
</tr>
<tr>
<td>AppBuilder property</td>
<td>Read/Write</td>
<td>Indicates whether or not the user has AppBuilder privileges.</td>
</tr>
<tr>
<td>Email property</td>
<td>Read/Write</td>
<td>Sets or returns the user's email address.</td>
</tr>
<tr>
<td>Fullname property</td>
<td>Read/Write</td>
<td>Sets or returns the user's full name.</td>
</tr>
<tr>
<td>Groups property</td>
<td>Read-only</td>
<td>Returns a collection of groups to which the user belongs.</td>
</tr>
<tr>
<td>MiscInfo property</td>
<td>Read/Write</td>
<td>Sets or returns the user's miscellaneous information.</td>
</tr>
<tr>
<td>Name property</td>
<td>Read-only</td>
<td>Returns the user's login name.</td>
</tr>
<tr>
<td>Phone property</td>
<td>Read/Write</td>
<td>Sets or returns the user's phone number.</td>
</tr>
<tr>
<td>SubscribedDatabases property</td>
<td>Read-only</td>
<td>Returns the collection of databases to which the user is subscribed.</td>
</tr>
<tr>
<td>SuperUser property</td>
<td>Read/Write</td>
<td>Indicates whether or not the user has SuperUser privileges.</td>
</tr>
<tr>
<td>UserMaintainer property</td>
<td>Read/Write</td>
<td>Indicates whether or not the user has user administration privileges.</td>
</tr>
</tbody>
</table>
**User object methods**

<table>
<thead>
<tr>
<th>Method name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SubscribeDatabase method</td>
<td>Subscribes this user to the specified database.</td>
</tr>
<tr>
<td>UnsubscribeAllDatabases method</td>
<td>Unsubscribes the user from all databases.</td>
</tr>
<tr>
<td>UnsubscribeDatabase method</td>
<td>Unsubscribes the user from the specified database.</td>
</tr>
</tbody>
</table>

*See Also:*

Users collection object
Active property

Indicates whether or not the user’s account is active.

**VB Syntax:**

`user.Active [= value]`

**Perl Syntax:**

`$user->GetActive();
$user->SetActive(boolean_value);`

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>user</code></td>
<td>A User object.</td>
</tr>
<tr>
<td><code>value</code></td>
<td>A Bool indicating whether the user’s account is active.</td>
</tr>
</tbody>
</table>

**Member of:**

User object

**Remarks:**

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.

**See Also:**

SubscribeDatabase method
UnsubscribeDatabase method
SubscribedDatabases property
AppBuilder property
SuperUser property
UserMaintainer property
**AppBuilder property**

Indicates whether or not the user has AppBuilder privileges.

**VB Syntax:**

```vbnet
user.AppBuilder [= value]
```

**Perl Syntax:**

```perl
$user->GetAppBuilder();
$user->SetAppBuilder(boolean_value);
```

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>user</td>
<td>User object.</td>
</tr>
<tr>
<td>value</td>
<td>A Bool indicating whether or not the user’s account has AppBuilder privileges.</td>
</tr>
</tbody>
</table>

**Member of:**

User object

**Remarks:**

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.)

**See Also:**

Active property
SuperUser property
UserMaintainer property
Schema object
Email property

Sets or returns the user’s e-mail address.

**Syntax:**

**VB Syntax:**

user.SuperUser [= value]

**Perl Syntax:**

$user->GetEmail();
$user->SetEmail(e-mail_address_string);

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>user</td>
<td>A User object.</td>
</tr>
<tr>
<td>value</td>
<td>A String containing the user's email address.</td>
</tr>
</tbody>
</table>

**Member of:**

User object

**Remarks:**

This property can be returned or set.

**See Also:**

Fullname property
Name property
**Fullname property**

Sets or returns the user's full name.

**VB Syntax:**

```
user.Fullname [= value]
```

**Perl Syntax:**

```
$user->GetFullname();
$user->SetFullname(full_name_string);
```

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>user</td>
<td>A User object.</td>
</tr>
<tr>
<td>value</td>
<td>A String containing the user's full name, as opposed to the user's login name.</td>
</tr>
</tbody>
</table>

**Member of:**

User object

**Remarks:**

This property can be returned or set.

**See Also:**

Email property
Name property
Groups property

Returns a collection of groups to which the user belongs.

**VB Syntax:**

```vbnet
 user. Groups
```

**Perl Syntax:**

```perl
$user->GetGroups();
```

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>user</td>
<td>A User object.</td>
</tr>
<tr>
<td>Return value</td>
<td>A Groups collection object containing the groups to which this user belongs.</td>
</tr>
</tbody>
</table>

**Member of:**

User object

**Remarks:**

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.

**See Also:**

AddUser method of the Group object  
Users collection object  
Group object  
Groups collection object
**MiscInfo property**

Sets or returns the user’s miscellaneous information.

**Syntax:**

**VB Syntax:**

```vbnet
user.MiscInfo [ = value ]
```

**Perl Syntax:**

```perl
$user->GetMiscInfo();
$user->SetMiscInfo(user_info_string);
```

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>user</td>
<td>A User object.</td>
</tr>
<tr>
<td>value</td>
<td>A String containing miscellaneous information about the user.</td>
</tr>
</tbody>
</table>

**Member of:**

User object

**Remarks:**

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.

**See Also:**

Fullname property
Name property
Name property

Returns the user’s login name.

**VB Syntax:**

`user.Name`

**Perl Syntax:**

`$user->getName();`

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>user</code></td>
<td>A User object.</td>
</tr>
<tr>
<td><strong>Return value</strong></td>
<td>A String containing the user’s login name.</td>
</tr>
</tbody>
</table>

**Member of:**

User object

**Remarks:**

This property is read-only.

**See Also:**

Fullname property
Phone property

Sets or returns the user’s phone number.

**VB Syntax:**

```vbnet
user.Phone [= value]
```

**Perl Syntax:**

```perl
$user->GetPhone();
$user->SetPhone(phone_number_string);
```

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>user</td>
<td>A User object.</td>
</tr>
<tr>
<td>value</td>
<td>A String containing the user’s phone number.</td>
</tr>
</tbody>
</table>

**Member of:**

User object

**Remarks:**

This property can be returned or set.

**See Also:**

Fullname property
Name property
SubscribedDatabases property

Returns the collection of databases to which the user is subscribed.

**VB Syntax:**

```vbnet
user.SubscribedDatabases
```

**Perl Syntax:**

```perl
$user->GetSubscribedDatabases();
```

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>user</code></td>
<td>A User object.</td>
</tr>
</tbody>
</table>

**Return value**

A Databases collection object containing the databases to which the user is subscribed.

**Member of:**

User object

**Remarks:**

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.

**See Also:**

SubscribeDatabase method
UnsubscribeAllDatabases method
UnsubscribeDatabase method
Database object
Databases collection object
**SuperUser property**

Indicates whether or not the user has SuperUser privileges.

**VB Syntax:**

user.SuperUser [= value]

**Perl Syntax:**

$user->GetSuperUser();
$user->SetSuperUser(boolean_value);

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>user</td>
<td>A User object.</td>
</tr>
<tr>
<td>value</td>
<td>A Bool indicating whether or not the user's account has SuperUser privileges.</td>
</tr>
</tbody>
</table>

**Member of:**

User object

**Remarks:**

This property can be returned or set.

Users with SuperUser privileges have full access to the master database and can perform user administration tasks or create and modify schemas.

**See Also:**

Active property  
AppBuilder property  
UserMaintainer property
UserMaintainer property

Indicates whether or not the user has user administration privileges.

VB Syntax:

user.UserMaintainer [= value]

Perl Syntax:

$user->user.GetUserMaintainer();
$user->user.SetUserMaintainer(boolean_value);

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>user</td>
<td>A User object.</td>
</tr>
<tr>
<td>value</td>
<td>A Bool indicating whether or not the user’s account has user administration privileges.</td>
</tr>
</tbody>
</table>

Member of:

User object

Remarks:

This property can be returned or set.

Users with SuperUser privileges can perform user administration tasks, such as adding new users or modifying the accounts of existing users.

See Also:

Active property
AppBuilder property
SuperUser property
**SubscribeDatabase method**

Subscribes this user to the specified database.

**VB Syntax:**

```vbnet
user.SubscribeDatabase database
```

**Perl Syntax:**

```perl
$user->SubscribeDatabase($database);
```

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>user</td>
<td>A User object.</td>
</tr>
<tr>
<td>database</td>
<td>The Database object to which the user will be subscribed.</td>
</tr>
<tr>
<td>Return value</td>
<td>None.</td>
</tr>
</tbody>
</table>

**Member of:** User object

**Remarks:**

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.

**See Also:**

UnsubscribeAllDatabases method  
UnsubscribeDatabase method  
SubscribedDatabases property
UnsubscribeAllDatabases method

Unsubscribes the user from all databases.

VB Syntax:

```vbnet
user.UnsubscribeAllDatabases
```

Perl Syntax:

```perl
$user->UnsubscribeAllDatabases();
```

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>user</td>
<td>A User object.</td>
</tr>
<tr>
<td>Return value</td>
<td>None.</td>
</tr>
</tbody>
</table>

Member of: User object

Remarks:

Calling this method disassociates the user from all user databases in the master database. The user’s account is still active.

See Also:

SubscribeDatabase method
UnsubscribeDatabase method
Active property
SubscribedDatabases property
UnsubscribeDatabase method

Unsubscribes the user from the specified database.

**VB Syntax:**

```vbnet
user.UnsubscribeDatabase database
```

**Perl Syntax:**

```perl
$user->UnsubscribeDatabase($database);
```

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>user</td>
<td>A User object.</td>
</tr>
<tr>
<td>database</td>
<td>The Database object from which the user will be unsubscribed.</td>
</tr>
</tbody>
</table>

**Return value**

None.

**Member of:** User object

**Remarks:**

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.

**See Also:**

- SubscribeDatabase method
- UnsubscribeAllDatabases method
- SubscribedDatabases property
Group object

**Group object properties**

<table>
<thead>
<tr>
<th>Property name</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Active property</td>
<td>Read/Write</td>
<td>Indicates whether or not the group is active.</td>
</tr>
<tr>
<td>SubscribedDatabases property</td>
<td>Read-only</td>
<td>Returns the collection of databases to which this group is subscribed.</td>
</tr>
<tr>
<td>Name property</td>
<td>Read/Write</td>
<td>Sets or returns the name of the group.</td>
</tr>
<tr>
<td>Users property</td>
<td>Read-only</td>
<td>Returns the collection of users belonging to this group.</td>
</tr>
</tbody>
</table>

**Methods**

<table>
<thead>
<tr>
<th>Method name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddUser method</td>
<td>Adds a user to this group.</td>
</tr>
<tr>
<td>SubscribeDatabase method</td>
<td>Subscribes this group to the specified database.</td>
</tr>
<tr>
<td>UnsubscribeAllDatabases method</td>
<td>Unsubscribes the group from all databases.</td>
</tr>
<tr>
<td>UnsubscribeDatabase method</td>
<td>Unsubscribes the group from the specified database.</td>
</tr>
</tbody>
</table>

**See Also:**

Database object
User object
**Active property**

Indicates whether or not the group is active.

**VB Syntax:**

```vbnet
group.Active [= value]
```

**Perl Syntax:**

```perl
$group->GetActive();
$group->SetActive(boolean_value);
```

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>group</code></td>
<td>A Group object, representing the set of groups associated with the current master database.</td>
</tr>
<tr>
<td><code>value</code></td>
<td>A Bool indicating whether or not the group is active.</td>
</tr>
</tbody>
</table>

**Member of:**

Group object

**Remarks:**

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.

**See Also:**

Active property of the User object

User object
AddUser method

Adds a user to this group.

**VB Syntax:**

group.AddUser user

**Perl Syntax:**

$group-&gt;AddUser(user);

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>group</td>
<td>A Group object.</td>
</tr>
<tr>
<td>user</td>
<td>The User object corresponding to the user account.</td>
</tr>
<tr>
<td>Return value</td>
<td>None.</td>
</tr>
</tbody>
</table>

**Member of:** Group object

**See Also:**

User object
SubscribedDatabases property

Returns the collection of databases to which this group is subscribed.

VB Syntax:

```vbnet
group.SubscribedDatabases
```

Perl Syntax:

```perl
$group->GetSubscribedDatabases();
```

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>group</code></td>
<td>A Group object, representing the set of groups associated with the current master database.</td>
</tr>
<tr>
<td><strong>Return value</strong></td>
<td>A Databases collection object containing the databases to which this group is subscribed.</td>
</tr>
</tbody>
</table>

Member of:

Group object

Remarks:

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

Each element in the returned collection is a Database object.

See Also:

SubscribeDatabase method
UnsubscribeAllDatabases method
SubscribedDatabases property of the User object
Database object
Databases collection object
User object
Name property

Sets or returns the name of the group.

**VB Syntax:**

`group.Name [= value]`

**Perl Syntax:**

$group->getGroups();
$group->setGroups(string_of_group_name);

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>group</td>
<td>A Group object, representing the set of groups associated with the current master database.</td>
</tr>
<tr>
<td>value</td>
<td>A String containing the name of the group.</td>
</tr>
</tbody>
</table>

**Member of:**

Group object

**Remarks:**

This property can be returned or set.

**See Also:**

Active property
Users property

Returns the collection of users belonging to this group.

**VB Syntax:**

`group. Users`

**Perl Syntax:**

`$group->GetUsers();`

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>group</code></td>
<td>A Group object, representing the set of groups associated with the current master database.</td>
</tr>
</tbody>
</table>

**Return value**

An Users collection object containing the users belonging to this group.

**Member of:**

Group object

**Remarks:**

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

Each element in the returned collection is an User object. To add users to a group, use the AddUser method.

**See Also:**

AddUser method
User object
Users collection object
SubscribeDatabase method

Subscribes this group to the specified database.

**VB Syntax:**

```
group.SubscribeDatabase database
```

**Perl Syntax:**

```
$group->SubscribeDatabase(database);
```

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>group</td>
<td>A Group object.</td>
</tr>
<tr>
<td>database</td>
<td>The Database object to which the group will be subscribed.</td>
</tr>
<tr>
<td>Return value</td>
<td>None.</td>
</tr>
</tbody>
</table>

**Member of:** Group object

**Remarks:**

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.

**See Also:**

UnsubscribeAllDatabases method  
UnsubscribeDatabase method  
SubscribedDatabases property
UnsubscribeAllDatabases method

Unsubscribes the group from all databases.

**VB Syntax:**

group.UnsubscribeAllDatabases

**Perl Syntax:**

$group->UnsubscribeAllDatabases();

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>group</td>
<td>A Group object.</td>
</tr>
</tbody>
</table>

**Return value**: None.

**Member of**: Group object

**Remarks:**

Calling this method disassociates the group from all user databases in the master database. The group is still active.

**See Also:**

SubscribeDatabase method
UnsubscribeDatabase method
Active property
SubscribedDatabases property
**UnsubscribeDatabase method**

Unsubscribes the group from the specified database.

**VB Syntax:**

```vbnet
group.UnsubscribeDatabase database
```

**Perl Syntax:**

```perl
$group->UnsubscribeDatabase(database);
```

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>group</td>
<td>A Group object.</td>
</tr>
<tr>
<td>database</td>
<td>The Database object from which the group will be unsubscribed.</td>
</tr>
</tbody>
</table>

**Return value**

None.

**Member of:** Group object

**Remarks:**

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.

**See Also:**

SubscribeDatabase method
UnsubscribeAllDatabases method
SubscribedDatabases property
Databases collection object

Databases collection object properties

<table>
<thead>
<tr>
<th>Property name</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count property</td>
<td>Read-only</td>
<td>Returns the number of items in the collection.</td>
</tr>
</tbody>
</table>

Databases collection object methods

<table>
<thead>
<tr>
<th>Method name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Item method</td>
<td>Returns the item at the specified index in the collection.</td>
</tr>
</tbody>
</table>

See Also:

Database object
Count property

Returns the number of items in the collection.

**VB Syntax:**

collection.Count

**Perl Syntax:**

$collection->Count();

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>collection</td>
<td>A Databases collection object, representing the set of databases associated with the current master database.</td>
</tr>
<tr>
<td>Return value</td>
<td>A Long indicating the number of items in the collection object. This method returns zero if the collection contains no items.</td>
</tr>
</tbody>
</table>

**Member of:** Databases collection object

**Remarks:**

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

**See Also:**

Item method
Item method

Returns the specified item in the collection.

**VB Syntax:**

```vbnet
collection.Item itemNum
collection.Item name
```

**Perl Syntax:**

```perl
$collection->Item(itemNum);
$collection->ItemByName(name);
```

### Identifier | Description
--- | ---
**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.

**Member of:** Databases collection object  

**Remarks:**

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

**See Also:**

Count property
### EntityDefs collection object properties

<table>
<thead>
<tr>
<th>Property name</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count property</td>
<td>Read-only</td>
<td>Returns the number of items in the collection.</td>
</tr>
</tbody>
</table>

### EntityDefs collection object method

<table>
<thead>
<tr>
<th>Method name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Item method</td>
<td>Returns the specified item in the collection.</td>
</tr>
</tbody>
</table>

### See Also:

EntityDef object
Count property

Returns the number of items in the collection.

VB Syntax:

collection.Count

Perl Syntax:

$collection->Count();

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>collection</td>
<td>An EntityDefs collection object, representing the set of EntityDefs available for fetching as a collection.</td>
</tr>
</tbody>
</table>

Return value: A Long that specifies the number of items in the collection object. This method returns zero if the collection contains no items.

Member of: EntityDefs collection object

Remarks:

This property is read-only.

See Also:

Item method
EntityDef object
**Item method**

Returns the specified item in the collection.

**Syntax:**

```
collection.Item itemNum
collection.Item name
```

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>collection</code></td>
<td>An EntityDefs collection object representing the set of EntityDefs in a schema.</td>
</tr>
<tr>
<td><code>itemNum</code></td>
<td>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.</td>
</tr>
<tr>
<td><code>name</code></td>
<td>A String that serves as a key into the collection. This string corresponds to the unique key returned by the DisplayName property of the desired EntityDefs.</td>
</tr>
</tbody>
</table>

**Return value**

The EntityDef object at the specified location in the collection.

**Member of:** EntityDefs collection object

**Remarks:**

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

**See Also:**

Count property
Groups collection object

Properties

<table>
<thead>
<tr>
<th>Property name</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count property</td>
<td>Read-only</td>
<td>Returns the number of items in the collection.</td>
</tr>
</tbody>
</table>

Methods

<table>
<thead>
<tr>
<th>Method name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Item method</td>
<td>Returns the item at the specified index in the collection.</td>
</tr>
</tbody>
</table>

See Also:

Group object
Count property

Returns the number of items in the collection.

**VB Syntax:**

collection.Count

**Perl Syntax:**

$collection->Count();

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>collection</td>
<td>A Groups collection object, representing the set of groups associated with the current master database.</td>
</tr>
</tbody>
</table>

**Return value**

A Long indicating the number of items in the collection object. This method returns zero if the collection contains no items.

**Member of:** Groups collection object

**Remarks:**

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

**See Also:**

Item method
Item method

Returns the specified item in the collection.

VB Syntax:

collection.Item itemNum
collection.Item name

Perl Syntax:

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

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>collection</td>
<td>A Groups collection object, representing the set of groups associated with the current master database.</td>
</tr>
<tr>
<td>itemNum</td>
<td>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.</td>
</tr>
<tr>
<td>name</td>
<td>A String that serves as a key into the collection. This string corresponds to the unique key of the desired Group object.</td>
</tr>
<tr>
<td>Return value</td>
<td>The Group object at the specified location in the collection.</td>
</tr>
</tbody>
</table>

Member of: Groups collection object

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

See Also:
Count property
Schemas collection object

_Schemas collection properties_

<table>
<thead>
<tr>
<th>Property name</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count property</td>
<td>Read-only</td>
<td>Returns the number of items in the collection.</td>
</tr>
</tbody>
</table>

_Schemas collection methods_

<table>
<thead>
<tr>
<th>Method name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Item method</td>
<td>Returns the item at the specified index in the collection.</td>
</tr>
</tbody>
</table>

See Also:

Schema object
SchemaRev object
Count property

Returns the number of items in the collection.

**VB Syntax:**

collection.Count

**Perl Syntax:**

$collection->Count();

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>collection</code></td>
<td>A Schemas collection object, representing the set of schemas associated with the current master database.</td>
</tr>
</tbody>
</table>

**Return value**

A Long indicating the number of items in the collection object. This method returns zero if the collection contains no items.

**Member of:** Schemas collection object

**Remarks:**

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

**See Also:**

Item method
Item method

Returns the specified item in the collection.

VB Syntax:

```vbnet
collection.Item itemNum
collection.Item name
```

Perl Syntax:

```perl
$collection->Item(itemNum);
$collection->ItemByName(name);
```

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>collection</code></td>
<td>A Schemas collection object, representing the set of schemas associated with the current master database.</td>
</tr>
<tr>
<td><code>itemNum</code></td>
<td>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.</td>
</tr>
<tr>
<td><code>name</code></td>
<td>A String that serves as a key into the collection. This string corresponds to the unique key of the desired Schema object.</td>
</tr>
</tbody>
</table>

**Return value**

The Schema object at the specified location in the collection.

**Member of:** Schemas collection object

**Remarks:**

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

**See Also:**

Count property
## SchemaRevs collection object

### SchemaRevs object property and method

<table>
<thead>
<tr>
<th>Property and method</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count property</td>
<td>Read-only</td>
<td>Returns the number of items in the collection.</td>
</tr>
<tr>
<td>Item method</td>
<td>Read-only</td>
<td>Returns the item at the specified index in the collection.</td>
</tr>
</tbody>
</table>

### See Also:
- Schemas property of the AdminSession object
- AdminSession object
- SchemaRev object
- Schemas collection object
Count property

Returns the number of items in the collection.

**VB Syntax:**

```vbnet
collection.Count
```

**Perl Syntax:**

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

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>collection</code></td>
<td>A SchemaRevs collection object, representing the set of schema revisions associated with the current master database.</td>
</tr>
<tr>
<td><strong>Return value</strong></td>
<td>A Long indicating the number of items in the collection object. This method returns zero if the collection contains no items.</td>
</tr>
</tbody>
</table>

**Member of:** SchemaRevs collection object

**Remarks:**

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

**See Also:**

Item method
Item method

Returns the specified item in the collection.

**VB Syntax:**

collection.Item itemNum
collection.Item name

**Perl Syntax:**

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

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>collection</td>
<td>A SchemaRevs collection object, representing the set of schema revisions associated with the current master database.</td>
</tr>
<tr>
<td>itemNum</td>
<td>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.</td>
</tr>
<tr>
<td>name</td>
<td>A String that serves as a key into the collection. This string corresponds to the unique key of the desired SchemaRev object.</td>
</tr>
</tbody>
</table>

**Return value**
The SchemaRev object at the specified location in the collection.

**Member of:** SchemaRevs collection object

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

**See Also:**
Count property
Users collection object

*Users Collection Properties*

<table>
<thead>
<tr>
<th>Property name</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count property</td>
<td>Read-only</td>
<td>Returns the number of items in the collection.</td>
</tr>
</tbody>
</table>

*Users Collection Methods*

<table>
<thead>
<tr>
<th>Method name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Item Method</td>
<td>Returns the item at the specified index in the collection.</td>
</tr>
</tbody>
</table>

*See Also:*

User object
Count property

Returns the number of items in the collection.

VB Syntax:

collection.Count

Perl Syntax:

$collection->Count();

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>collection</td>
<td>A Users collection object, representing the set of users associated with the current master database.</td>
</tr>
<tr>
<td>Return value</td>
<td>A Long indicating the number of items in the collection object. This method returns zero if the collection contains no items.</td>
</tr>
</tbody>
</table>

Member of: Users collection object

Remarks:

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

See Also:

Item Method
**Item Method**

Returns the specified item in the collection.

**VB Syntax:**

```
collection.Item itemNum
collection.Item name
```

**Perl Syntax:**

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

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>collection</td>
<td>A Users collection object, representing the set of users associated with the current master database.</td>
</tr>
<tr>
<td>itemNum</td>
<td>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.</td>
</tr>
<tr>
<td>name</td>
<td>A String that serves as a key into the collection. This string corresponds to the unique key of the desired User object.</td>
</tr>
<tr>
<td><strong>Return value</strong></td>
<td>The User object at the specified location in the collection.</td>
</tr>
</tbody>
</table>

**Member of:** Users collection object

**Remarks:**

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

**See Also:**

Count property
Attachment-Related Objects

The ClearQuest API provides four objects related to attachments:

- AttachmentFields collection object
- AttachmentField object
- Attachments collection object
- Attachment object

Remarks:

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

To support this functionality, the API provides four objects: AttachmentFields, AttachmentField, Attachments, and Attachment.

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

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

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

An Attachment object contains a single attached file.

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

See Also:

Attachments Code Example
AttachmentFields collection object

AttachmentFields object properties

<table>
<thead>
<tr>
<th>Property name</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count Property</td>
<td>Read-only</td>
<td>Returns the number of items in the collection.</td>
</tr>
</tbody>
</table>

AttachmentFields object methods

<table>
<thead>
<tr>
<th>Method name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Item Method</td>
<td>Returns the specified item in the collection.</td>
</tr>
</tbody>
</table>

See Also:

Attachment object
AttachmentField object
Attachments collection object
Attachment-Related Objects
Attachments Code Example
Count Property

Returns the number of items in the collection.

**VB Syntax:**

```
collection.Count
```

**Perl Syntax:**

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

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>collection</code></td>
<td>An AttachmentFields collection object, representing all of the attachment fields in a record.</td>
</tr>
<tr>
<td><strong>Return value</strong></td>
<td>A Long indicating the number of items in the collection object. This collection always contains at least one item.</td>
</tr>
</tbody>
</table>

**Member of:** AttachmentFields collection object

**Remarks:**

This property is read-only.

**Examples:**

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

**See Also:**

Item Method
Attachments Code Example
**Item Method**

Returns the specified item in the collection.

**VB Syntax:**

```
collection.Item itemNum
collection.Item name
```

**Perl Syntax:**

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

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>collection</td>
<td>An AttachmentFields collection object, representing all of the attachment fields in a record.</td>
</tr>
<tr>
<td>itemNum</td>
<td>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.</td>
</tr>
<tr>
<td>name</td>
<td>A String that serves as a key into the collection. This string corresponds to the FieldName property of the desired AttachmentField.</td>
</tr>
</tbody>
</table>

**Return value**

The AttachmentField object at the specified location in the collection.

**Member of:** AttachmentFields collection object

**Remarks:**

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

**Examples:**

```
set attachFields = entity.AttachmentFields
numFields = attachFields.Count
For x = 0 to numFields
    set oneField = attachFields.Item x
```
See Also:

Count Property
Attachments Code Example

...
Attachment object

**Attachment object properties**

<table>
<thead>
<tr>
<th>Property name</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description property</td>
<td>Read/Write</td>
<td>Sets or returns the description of the attached file.</td>
</tr>
<tr>
<td>DisplayName property</td>
<td>Read-only</td>
<td>Returns the unique key used to identify the attachment.</td>
</tr>
<tr>
<td>FileName property</td>
<td>Read-only</td>
<td>Returns the path name of the attached file.</td>
</tr>
<tr>
<td>FileSize property</td>
<td>Read-only</td>
<td>Returns the size of the attached file in bytes.</td>
</tr>
</tbody>
</table>

**Attachment object methods**

<table>
<thead>
<tr>
<th>Method name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load method</td>
<td>Writes this object’s contents to the specified file</td>
</tr>
</tbody>
</table>

**See Also:**
AttachmentField object
AttachmentFields collection object
Attachments collection object
Attachment-Related Objects
Attachments Code Example
Description property

Sets or returns the description of the attached file.

**VB Syntax:**

```
attachment.Description [= value]
```

**Perl Syntax:**

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

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>attachment</td>
<td>An Attachment object, representing the attachment of a file to a record.</td>
</tr>
</tbody>
</table>

**Return value**

A String containing the descriptive text.

**Member of:** Attachment object

**Remarks:**

This is a read-write property: Its value can be set, which is unlike the other Attachment properties.

**Examples:**

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

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

See Also:

FileName property
Attachments Code Example
**DisplayName property**

Returns the unique key used to identify the attachment.

**VB Syntax:**

`attachment.DisplayName`

**Perl Syntax:**

`$attachment->GetDisplayName()`

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>attachment</td>
<td>An Attachment object, representing the attachment of a file to a record.</td>
</tr>
</tbody>
</table>

**Member of:** Attachment object

**Remarks:**

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.

**Examples:**

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

set attachments = attachField1.Attachments
numAttachments = attachments.Count
For x = 0 to numAttachments
   set attachment = attachments.Item(x)
   description = attachment.Description
   key = attachment.DisplayName
```

currentSession.OutputDebugString "Unique key: " & key & "$ - 
  description: " & description 
  & description
Next

**See Also:**

Description property
FileName property
FileName property

Returns the path name of the attached file.

**VB Syntax:**

```
attachment.FileName
```

**Perl Syntax:**

```
$attachment->GetFileName()
```

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>attachment</td>
<td>An Attachment object, representing the attachment of a file to a record.</td>
</tr>
</tbody>
</table>

**Return value**

A String containing the name of the attached file.

**Remarks:**

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

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

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

**Examples:**

' This example assumes there is at least 1 attachment field
' and 1 attachment associated with the record.
set currentSession = GetSession
set attachFields = entity.AttachmentFields
set attachField1 = attachFields.Item(0)
set attachments = attachField1.Attachments
numAttachments = attachments.Count
For x = 0 to numAttachments
  set attachment = attachments.Item(x)
  fileName = attachment.FileName
  fileSize = attachment.FileSize
Next

**See Also:**

FileSize property
Add method of the Attachments object
Commit method of the Entity object
Attachments collection object
Entity object
Attachments Code Example
**FileSize property**

Returns the size of the attached file in bytes.

**VB Syntax:**

`attachment.FileSize`

**Perl Syntax:**

`$attachment->GetFileSize();`

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>attachment</code></td>
<td>An Attachment object, representing the attachment of a file to a record.</td>
</tr>
</tbody>
</table>

**Return value**

A Long indicating the file’s size in bytes.

**Remarks:**

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

This method should be called only after the attachment has been committed to the database. If you call it earlier, the return value will be empty.

**Examples:**

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

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

FileName property
Attachments Code Example
Load method

Writes this object’s contents to the specified file.

VB Syntax:

\texttt{attachment.Load filename}

Perl Syntax:

\texttt{$attachment->Load(filename);}

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{attachment}</td>
<td>An Attachment object, representing the attachment of a file to a record.</td>
</tr>
<tr>
<td>\texttt{filename}</td>
<td>A String containing the path name of the file you want to write. This path name can be an absolute or relative path.</td>
</tr>
</tbody>
</table>

Return value

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

Remarks:

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

Examples:

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

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

**See Also:**

Add method of the Attachments object
Attachments collection object
Attachments Code Example
AttachmentField object

AttachmentField object properties

<table>
<thead>
<tr>
<th>Property name</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attachments property</td>
<td>Read-only</td>
<td>Returns this attachment field’s collection of attachments.</td>
</tr>
<tr>
<td>DisplayNameHeader property</td>
<td>Read-only</td>
<td>Returns the unique keys of the attachments in this field.</td>
</tr>
<tr>
<td>FieldName property</td>
<td>Read-only</td>
<td>Returns the name of the attachment field.</td>
</tr>
</tbody>
</table>

See Also:

Attachment object
AttachmentFields collection object
Attachments collection object
Attachment-Related Objects
Attachments Code Example
Attachments property

Returns this attachment field’s collection of attachments.

VB Syntax:

\[ attachmentField.Attachments \]

Perl Syntax:

\[ $attachmentField->GetAttachments(); \]

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>field</td>
<td>An AttachmentField object representing one attachment field of a record.</td>
</tr>
</tbody>
</table>

Return value

An Attachments collection object, which itself contains a set of Attachment objects.

Member of: AttachmentField object

Remarks:

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

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

Examples:

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

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

' Do something with the attachments
Next

See Also:
DisplayNameHeader property
FieldName property
Attachments Code Example
Attachment object
DisplayNameHeader property

Returns the unique keys of the attachments in this field.

**VB Syntax:**

`attachmentField.DisplayNameHeader`

**Perl Syntax:**

`$attachmentField->GetDisplayNameHeader();`

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>field</td>
<td>An AttachmentField object representing one attachment field of a record.</td>
</tr>
</tbody>
</table>

**Return value**

A Variant containing an Array whose elements are Strings. Each String contains the DisplayName property of one Attachment object associated with this field.

**Member of:** AttachmentField object

**Remarks:**

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.

**Examples:**

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

keys = attachField1.DisplayNameHeader
x = 0
For Each key in keys
    OutputDebugString "Displaying key number " & x & " - " & key
    x = x + 1
Next
```
See Also:
Attachments property
FieldName property
DisplayName property of the Attachment object
Attachment object
Attachments Code Example
**FieldName property**

Returns the name of the attachment field.

**VB Syntax:**

`attachmentField.FieldName`

**Perl Syntax:**

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

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>field</code></td>
<td>An AttachmentField object representing one attachment field of a record.</td>
</tr>
</tbody>
</table>

**Member of:** AttachmentField object

**Remarks:**

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.

**Examples:**

```
' 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
```

**See Also:**

Attachments property
DisplayNameHeader property
Attachments Code Example
### Attachments collection object

#### Attachments object properties

<table>
<thead>
<tr>
<th>Property name</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count property</td>
<td>Read-only</td>
<td>Returns the number of items in the collection.</td>
</tr>
</tbody>
</table>

#### Attachments object methods

<table>
<thead>
<tr>
<th>Method name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add method</td>
<td>Adds an Attachment object to the collection.</td>
</tr>
<tr>
<td>Delete method</td>
<td>Deletes an attached file from the collection.</td>
</tr>
<tr>
<td>Item method</td>
<td>Returns the specified item in the collection.</td>
</tr>
</tbody>
</table>

#### See Also:

- Attachment object
- AttachmentField object
- Attachment-Related Objects
- Attachments Code Example
Count property

Returns the number of items in the collection.

VB Syntax:

\[\text{collection}.\text{Count}\]

Perl Syntax:

\$collection->\text{Count}();

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\text{collection}</td>
<td>An Attachments collection object, representing the set of attachments in one field of a record.</td>
</tr>
</tbody>
</table>

Return value

A Long indicating the number of items in the collection object. This method returns zero if the collection contains no items.

Member of: Attachments collection object

Remarks:

This property is read-only.

Examples:

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

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

' Do something with the attachments
Next
See Also:

Item method
Attachments Code Example
Add method

Adds an Attachment object to the collection.

**VB Syntax:**

```
attachments.Add filename, description
```

**Perl Syntax:**

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

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>attachments</code></td>
<td>An Attachments collection object, representing the set of attachments in one field of a record.</td>
</tr>
<tr>
<td><code>filename</code></td>
<td>A String containing the absolute or relative pathname of the file to be attached to this field.</td>
</tr>
<tr>
<td><code>description</code></td>
<td>A String that contains arbitrary text describing the nature of the attached file.</td>
</tr>
</tbody>
</table>

**Return value**

A Boolean that is True if the file was added successfully, otherwise False.

**Member of:** Attachments collection object

**Remarks:**

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.

**Examples:**

```
' 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 property
Delete method
Item method
Attachments Code Example
Delete method

Deletes an attached file from the collection.

**VB Syntax:**

```vbnet
attachments.Delete itemNum
attachments.Delete displayName
```

**Perl Syntax:**

```perl
$attachments->Delete(itemNum);
$attachments->DeleteByName(displayName);
```

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>attachments</td>
<td>An Attachments collection object, representing the set of attachments in one field of a record.</td>
</tr>
<tr>
<td>itemNum</td>
<td>A Long that is an index into the collection. This index is 0-based and points to the file that you want to delete.</td>
</tr>
<tr>
<td>displayName</td>
<td>A String that serves as a key into the collection. Its value specifies the DisplayName property of the Attachment object you want to delete.</td>
</tr>
</tbody>
</table>

**Return value**

A Boolean that is True if the file was deleted successfully, otherwise False.

**Member of:** Attachments collection object

**Remarks:**

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

**Examples:**

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

set attachments = attachField1.Attachments
If Not attachments.Delete(0) Then
    OutputDebugString "Error deleting the attachment."
End If

See Also:

Count property
Add method
Item method
Attachments Code Example
Item method

Returns the specified item in the collection.

VB Syntax:

collection.Item itemNum
collection.Item name

Perl Syntax:

$collection-&gt;Item(itemNum);
$collection-&gt;ItemByName(name);

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>collection</td>
<td>An Attachments collection object, representing the set of attachments in one field of a record.</td>
</tr>
<tr>
<td>itemNum</td>
<td>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.</td>
</tr>
<tr>
<td>name</td>
<td>A String that serves as a key into the collection. This string corresponds to the DisplayName property of the desired Attachment.</td>
</tr>
</tbody>
</table>

Return value: The Attachment object at the specified location in the collection.

Member of: Attachments collection object

Remarks:

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

Examples:

' 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 property
Attachments Code Example
DatabaseDescription object

Database description object methods

<table>
<thead>
<tr>
<th>Method name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetDatabaseConnectString method</td>
<td>Returns the &quot;direct connect&quot; string for logging into the database.</td>
</tr>
<tr>
<td>GetDatabaseName method</td>
<td>Returns the name of the database.</td>
</tr>
<tr>
<td>GetDatabaseSetName method</td>
<td>Returns the name of the database set of which this database is a member.</td>
</tr>
<tr>
<td>GetDescription method</td>
<td>Returns a string describing the contents of the database.</td>
</tr>
<tr>
<td>GetIsMaster method</td>
<td>Returns a Bool indicating whether this database is a master database.</td>
</tr>
<tr>
<td>GetLogin method</td>
<td>Returns the database login associated with the current user.</td>
</tr>
</tbody>
</table>

See Also:

GetSessionDatabase method of the Session object
Session object
GetDatabaseConnectString method

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

**VB Syntax:**

```
dbDesc.GetDatabaseConnectString
```

**Perl Syntax:**

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

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dbDesc</td>
<td>A DatabaseDescription object containing information about one of the installed databases.</td>
</tr>
</tbody>
</table>

**Return value**

A String whose value is the "direct connect" string.

**Member of:** DatabaseDescription object

**Remarks:**

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

**Examples:**

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

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

    ...  
Next
```
See Also:

UserLogon method of the Session object
Session object
Session and DatabaseDescription Code Example
GetDatabaseName method

Returns the name of the database.

**VB Syntax:**

```
dbDesc.GetDatabaseName
```

**Perl Syntax:**

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

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dbDesc</td>
<td>A DatabaseDescription object containing information about one of the installed databases.</td>
</tr>
<tr>
<td>Return value</td>
<td>A String containing the name of the database.</td>
</tr>
</tbody>
</table>

**Member of:** DatabaseDescription object

**Remarks:**

You can use the Session object’s GetAccessibleDatabases method to obtain a list of DatabaseDescription 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.

**Examples:**

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

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

See Also:
GetDatabaseSet Name method
GetAccessibleDatabases method of the Session object
Session object
Session and DatabaseDescription Code Example
Notation Conventions
GetDatabaseSetName method

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

**VB Syntax:**

\[dbDesc.GetDatabaseSetName\]

**Perl Syntax:**

\[$dbDesc->GetDatabaseSetName();\]

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(dbDesc)</td>
<td>A DatabaseDescription object containing information about one of the installed databases.</td>
</tr>
</tbody>
</table>

**Return value** A String containing the name of the database set.

**Member of:** DatabaseDescription object

**Remarks:**

You can use this method to get the database set name of this database. You can pass this name to the 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.

**Examples:**

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

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

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

**See Also:**

GetDatabaseName method
GetAccessibleDatabases method of the Session object
Session object
Session and DatabaseDescription Code Example
Notation Conventions
GetDescription method

Returns a string describing the contents of the database.

**VB Syntax:**

\[ dbDesc.GetDescription \]

**Perl Syntax:**

\[ $dbDesc->GetDescription(); \]

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dbDesc</td>
<td>A DatabaseDescription object containing information about one of the installed databases.</td>
</tr>
</tbody>
</table>

**Return value**

A String containing descriptive comments about the database.

**Member of:** DatabaseDescription object

**Remarks:**

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.

**Examples:**

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

```vbnet
set sessionObj = CreateObject("CLEARQUEST.SESSION")
' Login to each database successively.
set databases = sessionObj.GetAccessibleDatabases
For Each db in databases
   dbDescription = db.GetDescription
   ...
Next
```
See Also:
Description property of the Database object
Database object
GetIsMaster method

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

**VB Syntax:**

`dbDesc.GetIsMaster`

**Perl Syntax:**

`$dbDesc->GetIsMaster();`

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dbDesc</td>
<td>A DatabaseDescription object containing information about one of the installed databases.</td>
</tr>
</tbody>
</table>

**Return value**

True if this database is a master database, otherwise false.

**Member of:** DatabaseDescription object

**Remarks:**

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.

**Examples:**

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

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

See Also:

Logon method of the AdminSession object
AdminSession object
GetLogin method

Returns the database login associated with the current user.

**VB Syntax:**

```
dbDesc.GetLogin
```

**Perl Syntax:**

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

**Identifier** | **Description**
---|---
$dbDesc  | A DatabaseDescription object containing information about one of the installed databases.

**Return value** | A String containing the database login associated with the current user.

**Member of:** DatabaseDescription object

**Remarks:**

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

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

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

If you have access to the schema repository, you can retrieve information about this user database by accessing the properties of the corresponding Database object.
Examples:
The following example shows you how to logon to the database from a Visual Basic application.

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

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

        ...
    End If
Next
```

See Also:

- DBOLogin property of the Database object
- ROLogin property of the Database object
- RWLogin property of the Database object
- UserLogon method of the Session object
- Database object
- Session object
- Notation Conventions
EventObject object

**EventObject object properties**

<table>
<thead>
<tr>
<th>Property name</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CheckState</td>
<td>[post-beta]</td>
<td></td>
</tr>
<tr>
<td>EditText property</td>
<td>[post-beta]</td>
<td></td>
</tr>
<tr>
<td>EventType property</td>
<td>Read-only</td>
<td>Returns the type of event that caused the hook invocation.</td>
</tr>
<tr>
<td>ItemName property</td>
<td>Read-only</td>
<td>Returns the name of the form item that caused the hook to be invoked.</td>
</tr>
<tr>
<td>ObjectItem property</td>
<td>Read-only</td>
<td>Returns the entity object associated with the current selection.</td>
</tr>
<tr>
<td>StringItem property</td>
<td>Read-only</td>
<td>Returns the name of the menu item hook.</td>
</tr>
</tbody>
</table>

**See Also**

Entity object
**EventType property**

Returns the type of event that caused the hook invocation.

**VB Syntax:**

```
eteventObject.Type
```

**Perl Syntax:**

```
$eventObject->Type();
```

**Identifier** | **Description**
---|---
`eventObject` | An instance of EventObject.

**Return value**

A Long whose value is one of the EventType enumerated constants.

**Member of:** EventObject object

**Remarks:**

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

**See Also:**

EventType
ItemName property

Returns the name of the form item that caused the hook to be invoked.

**VB Syntax:**

```
eventObject.ItemName
```

**Perl Syntax:**

```
$eventObject->ItemName();
```

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>eventObject</td>
<td>An instance of EventObject.</td>
</tr>
<tr>
<td>Return value</td>
<td>A String containing the name of the form item from which the hook was invoked.</td>
</tr>
</tbody>
</table>

**Member of:** EventObject object

**Remarks:**

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

**See Also:**

ObjectItem property
ObjectItem property

Returns the entity object associated with the current selection.

**VB Syntax:**

```
eventObject.ObjectItem
```

**Identifier** | **Description**
--- | ---
`eventObject` | An instance of EventObject.

**Return value** | The Entity object associated with the current selection.

**Member of:** EventObject object

**Remarks:**

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

The Entity object contained in this property may not be the same object that invoked the current hook. This property is set only when the EventType property contains the value ITEM_SELECTION or ITEM_DLBCLICK.

**See Also:**

EventType property
Entity object
StringItem property

Returns the name of the menu item hook.

**VB Syntax:**

```vb
eventObject.StringItem
```

**Perl Syntax:**

```perl
$eventObject->StringItem();
```

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>eventObject</td>
<td>An instance of EventObject.</td>
</tr>
<tr>
<td><strong>Return value</strong></td>
<td>A String whose value indicates the menu item hook name when the EventType property contains the value CONTEXTMENU_ITEM_CONDITION; otherwise, this property contains an empty value.</td>
</tr>
</tbody>
</table>

**Member of:** EventObject object

**Remarks:**

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

**See Also:**

EventType property
Notation Conventions
FieldInfo object

**FieldInfo object methods**

<table>
<thead>
<tr>
<th>Method name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetMessageText method</td>
<td>Returns a String explaining why the value stored in the field is invalid.</td>
</tr>
<tr>
<td>GetName method</td>
<td>Returns the name of the field.</td>
</tr>
<tr>
<td>GetRequiredness method</td>
<td>Identifies the behavior of the specified field.</td>
</tr>
<tr>
<td>GetType method</td>
<td>Identifies the type of data that can be stored in this field.</td>
</tr>
<tr>
<td>GetValidationStatus method</td>
<td>Identifies whether the field's value is valid.</td>
</tr>
<tr>
<td>GetValue method</td>
<td>Returns the field's value as a single String.</td>
</tr>
<tr>
<td>GetValueAsList method</td>
<td>Returns an Array of Strings containing the values stored in the field.</td>
</tr>
<tr>
<td>GetValueStatus method</td>
<td>Identifies whether the field currently has a value.</td>
</tr>
<tr>
<td>ValidityChangedThisAction method</td>
<td>Returns True if the field's validity was changed by the most recent action.</td>
</tr>
<tr>
<td>ValidityChangedThisGroup method</td>
<td>Returns True if the field's validity was changed by the most recent group of SetFieldValue calls.</td>
</tr>
<tr>
<td>ValidityChangedThisSetValue method</td>
<td>Returns True if the field's validity was changed by the most recent SetFieldValue method call.</td>
</tr>
<tr>
<td>ValueChangedThisAction method</td>
<td>Returns True if this field's value was modified by the most recent action.</td>
</tr>
<tr>
<td>ValueChangedThisGroup method</td>
<td>Returns True if the field's value was modified by the most recent group of SetFieldValue calls.</td>
</tr>
<tr>
<td>ValueChangedThisSetValue method</td>
<td>Returns True if the field's value was modified by the most recent SetFieldValue method call.</td>
</tr>
</tbody>
</table>

**See Also:**

- GetFieldsUpdatedThisAction method
- GetFieldsUpdatedThisGroup method
- GetFieldsUpdatedThisSetValue method
- Entity object
- FieldInfo Code Example
- Notification Hook Code Example
GetMessageText method

Returns a String explaining why the value stored in the field is invalid.

**VB Syntax:**

`fieldInfo.GetMessageText`

**Perl Syntax:**

`$fieldInfo->GetMessageText();`

**Identifier** | **Description**
--- | ---
`fieldInfo` | A FieldInfo object, which contains information about one field of a user data record.

**Return value** | A String that explains why this field's value is invalid.

**Member of:** FieldInfo object

**Remarks:**

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.

**See Also:**

GetValidationStatus method
FieldInfo Code Example
Notation Conventions
**Get Name method**

Returns the name of the field.

**VB Syntax:**

`fieldInfo.Get_Name`

**Perl Syntax:**

`$fieldInfo->Get_Name();`

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>fieldInfo</code></td>
<td>A FieldInfo object, which contains information about one field of a user data record.</td>
</tr>
</tbody>
</table>

**Return value**

A String containing the name of the field.

**Member of:** FieldInfo object

**Remarks:**

Field names are used by various methods to identify a specific field in an Entity object.

**See Also:**

GetFieldNames method of the Entity object
Entity object
FieldInfo Code Example
GetRequiredness method

Identifies the behavior of the specified field.

**VB Syntax:**

`fieldInfo.GetRequiredness`

**Perl Syntax:**

`$fieldInfo->GetRequiredness();`

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>fieldInfo</code></td>
<td>A FieldInfo object, which contains information about one field of a user data record.</td>
</tr>
<tr>
<td><strong>Return value</strong></td>
<td>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).</td>
</tr>
</tbody>
</table>

**Member of:** FieldInfo object

**Remarks:**

A field can be mandatory, optional, or read-only. If the Entity does not have an action running at the time this method is called, the return value will always be READONLY. If an action is running, the return value can be READONLY, MANDATORY, or OPTIONAL.

This method never returns the value USE_HOOK. If the behavior of the field is determined by a permission hook, ClearQuest will have already executed that hook and cached the resulting value. This method then returns the cached value.

**See Also:**

GetFieldRequiredness method of the Entity object

Entity object

Notation Conventions
**GetType method**

Identifies the type of data that can be stored in this field.

**VB Syntax:**

`fieldInfo.GetField

**Perl Syntax:**

`$fieldInfo->GetField();`

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>fieldInfo</code></td>
<td>A FieldInfo object, which contains information about one field of a user data record.</td>
</tr>
</tbody>
</table>

**Return value**

A Long that specifies what type of data can be stored in this field. The value corresponds to one of the FieldType enumeration constants.

**Member of:** FieldInfo object

**Remarks:**

Fields can store strings, numbers, timestamps, references, and several other types. (See FieldType for the complete list.)

**See Also:**

GetFieldType method of the Entity object
Entity object
GetValidationStatus method

Identifies whether the field’s value is valid.

VB Syntax:

`fieldInfo.GetValidationStatus`

Perl Syntax:

`$fieldInfo->GetValidationStatus();`

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>fieldInfo</code></td>
<td>A FieldInfo object, which contains information about one field of a user data record.</td>
</tr>
</tbody>
</table>

**Return value**

A Long that identifies the validation status of this field. The value corresponds to one of the FieldValidationStatus enumeration constants.

**Member of:** FieldInfo object

**Remarks:**

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.

**See Also:**

GetMessageText method
FieldInfo Code Example
Notation Conventions
GetValue method

Returns the field's value as a single String.

**VB Syntax:**

`fieldInfo.GetValue`

**Perl Syntax:**

`$fieldInfo->GetValue();`

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>fieldInfo</code></td>
<td>A FieldInfo object, which contains information about one field of a user data record.</td>
</tr>
<tr>
<td>Return value</td>
<td>A String that contains the value or values stored in the field.</td>
</tr>
</tbody>
</table>

**Member of:** FieldInfo object

**Remarks:**

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.

**See Also:**

GetValueAsList method  
GetFieldV alue method of the Entity object  
Entity object  
FieldInfo Code Example  
Notification Hook Code Example
GetValueAsList method

Returns an Array of Strings containing the values stored in the field.

**VB Syntax:**

```vbnet
fieldInfo.GetValueAsList
```

**Perl Syntax:**

```perl
$fieldInfo->GetValueAsList();
```

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fieldInfo</td>
<td>A FieldInfo object, which contains information about one field of a user data record.</td>
</tr>
</tbody>
</table>

**Return value**

A Variant containing an Array whose elements are Strings. Each String contains the a single value stored in the field. If the field contains no values, this method returns an Empty Variant.

**Member of:** FieldInfo object

**Remarks:**

It is legal to use this method for a scalar field (that is, one that contains a single value). In that case, this method returns only one element in the Array (unless the field is empty in which case an Empty Variant is returned).

To determine if a field can contain multiple values, call the GetType method on the corresponding FieldInfo object. If the type of the field is REFERENCE_LIST, ATTACHMENT_LIST, or JOURNAL, the field can contain multiple values.

**Note:** Fields whose type is either ATTACHMENT_LIST or JOURNAL cannot be modified programmatically.

**See Also:**

GetValue method
AddFieldValue method of the Entity object
GetFieldValue method of the Entity object
FieldType
Entity object
Notation Conventions
GetValueStatus method

Identifies whether the field currently has a value.

**VB Syntax:**

`fieldInfo.GetValueStatus`

**Perl Syntax:**

`$fieldInfo->GetValueStatus();`

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>fieldInfo</code></td>
<td>A FieldInfo object, which contains information about one field of a user data record.</td>
</tr>
</tbody>
</table>

**Return value**

A Long that identifies the status of this field. The value corresponds to one of the ValueStatus enumeration constants.

**Member of:** FieldInfo object

**See Also:**

GetValue method
GetFieldValue method of the Entity object
SetFieldValue method of the Entity object
Entity object
FieldInfo Code Example
Notification Hook Code Example
ValidityChangedThisAction method

Returns True if the field’s validity was changed by the most recent action.

**VB Syntax:**

```vbnet
fieldInfo.ValidityChangedThisAction
```

**Perl Syntax:**

```perl
$fieldInfo->ValidityChangedThisAction();
```

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fieldInfo</td>
<td>A FieldInfo object, which contains information about one field of a user data record.</td>
</tr>
</tbody>
</table>

**Return value**

A Boolean that is True if the field’s validity changed since the most recent action was initiated, otherwise False.

**Member of:** FieldInfo object

**Remarks:**

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.

**See Also:**

ValidityChangedThisGroup method

ValidityChangedThisSetValue method

ValueChangedThisAction method

GetFieldsUpdatedThisAction method of the Entity object
Entity object
FieldInfo Code Example
ValidityChangedThisGroup method

Returns True if the field’s validity was changed by the most recent group of SetFieldValue method calls.

**VB Syntax:**

fieldInfo.ValidityChangedThisGroup

**Perl Syntax:**

$fieldInfo->ValidityChangedThisGroup();

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fieldInfo</td>
<td>A FieldInfo object, which contains information about one field of a user data record.</td>
</tr>
<tr>
<td>Return value</td>
<td>A Boolean that is True if the field's validity changed since the most recent call to the BeginNewFieldUpdateGroup method, otherwise False.</td>
</tr>
</tbody>
</table>

**Member of:** FieldInfo object

**Remarks:**

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.

**See Also:**

ValidityChangedThisAction method
ValidityChangedThisSetValue method
ValueChangedThisGroup method
BeginNewFieldUpdateGroup method of the Entity object
GetFieldsUpdatedThisGroup method of the Entity object
FieldValidationStatus
Entity object
FieldInfo Code Example
ValidityChangedThisSetValue method

Returns True if the field’s validity was changed by the most recent SetFieldValue call.

**VB Syntax:**

`fieldInfo.ValidityChangedThisSetValue`

**Perl Syntax:**

`$fieldInfo->ValidityChangedThisSetValue();`

**Identifier** | **Description**
---|---
`fieldInfo` | A FieldInfo object, which contains information about one field of a user data record.

**Return value** | A Boolean that is True if the field’s validity was changed by the most recent call to SetFieldValue, otherwise False.

**Member of:** FieldInfo object

**Remarks:**

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.

**See Also:**

ValidityChangedThisAction method
ValidityChangedThisGroup method
GetFieldsUpdatedThisSetValue method of the Entity object
SetFieldValue method of the Entity object
Entity object
FieldInfo Code Example
**ValueChangedThisAction method**

Returns True if this field’s value was modified by the most recent action.

**VB Syntax:**

`fieldInfo.ValueChangedThisAction`

**Perl Syntax:**

`$fieldInfo->ValueChangedThisAction();`

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>fieldInfo</code></td>
<td>A <code>FieldInfo</code> object, which contains information about one field of a user data record.</td>
</tr>
</tbody>
</table>

**Return value**  
A Boolean that is True if the field’s value was changed since the most recent action was initiated, otherwise False.

**Member of:** FieldInfo object

**Remarks:**

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.

**See Also:**

ValueChangedThisGroup method  
ValueChangedThisSetValue method  
BuildEntity method of the Session object  
EditEntity method of the Session object  
GetFieldsUpdatedThisAction method of the Entity object  
Entity object
Session object
FieldInfo Code Example
**ValueChangedThisGroup method**

Returns True if the field’s value was modified by the most recent group of SetFieldValue calls.

**VB Syntax:**

`fieldInfo.ValueChangedThisGroup`

**Perl Syntax:**

`$fieldInfo->ValueChangedThisGroup();`

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fieldInfo</td>
<td>A FieldInfo object, which contains information about one field of a user data record.</td>
</tr>
<tr>
<td><strong>Return value</strong></td>
<td>A Boolean that is True if the field’s value was changed since the most recent invocation of BeginNewFieldUpdateGroup, otherwise False.</td>
</tr>
</tbody>
</table>

**Member of:** FieldInfo object

**Remarks:**

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.

**See Also:**

ValueChangedThisAction method
ValueChangedThisSetValue method
BeginNewFieldUpdateGroup method of the Entity object
GetFieldsUpdatedThisGroup method of the Entity object
SetFieldValue method of the Entity object
Entity object
FieldInfo Code Example
**ValueChangedThisSetValue method**

Returns True if the field’s value was modified by the most recent SetFieldValue call.

**VB Syntax:**

```vbnet
fieldInfo.ValueChangedThisSetValue
```

**Perl Syntax:**

```perl
$fieldInfo->ValueChangedThisSetValue();
```

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fieldInfo</td>
<td>A FieldInfo object, which contains information about one field of a user data record.</td>
</tr>
</tbody>
</table>

**Return value**

A Boolean that is True if the field’s value was changed by the most recent call to SetFieldValue, otherwise False.

**Member of:** FieldInfo object

**Remarks:**

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.

**See Also:**

GetFieldsUpdatedThisSetValue method of the Entity object
SetFieldValue method of the Entity object
FieldTypeEntity object
FieldInfo Code Example
History-Related Objects

The ClearQuest API provides four objects related to history:

HistoryFields collection object
HistoryField object
Histories collection object
History object

Remarks

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

To support this functionality, the API provides four objects: HistoryFields, HistoryField, Histories, and History:

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

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

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

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

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

See Also:

HistoryFields property of the Entity object
Entity object
HistoryFields collection object

### HistoryFields collection properties

<table>
<thead>
<tr>
<th>Property name</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count property</td>
<td>Read-only</td>
<td>Returns the number of items in the collection.</td>
</tr>
</tbody>
</table>

### HistoryFields collection methods

<table>
<thead>
<tr>
<th>Method name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Item method</td>
<td>Returns the specified item in the collection.</td>
</tr>
</tbody>
</table>

**See Also:**

History object
Histories collection object
HistoryField object
Count property

Returns the number of items in the collection.

VB Syntax:

collection.Count

Perl Syntax:

$collection->Count();

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>collection</td>
<td>A HistoryFields collection object, representing all of the history fields of a record.</td>
</tr>
</tbody>
</table>

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

Member of: HistoryFields collection object

Remarks:

This property is read-only.

See Also:

Item method
Item method

Returns the specified item in the collection.

**VB Syntax:**

```vbnet
collection.Item itemNum
collection.Item name
```

**Perl Syntax:**

```perl
$collection->Item(itemNum);
$collection->ItemByName(name);
```

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>collection</td>
<td>A HistoryFields collection object, representing all of the history fields of a record.</td>
</tr>
<tr>
<td>itemNum</td>
<td>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.</td>
</tr>
<tr>
<td>name</td>
<td>A String that serves as a key into the collection. This string corresponds to the field name of the desired HistoryField.</td>
</tr>
</tbody>
</table>

**Return value**

The HistoryField object at the specified location in the collection.

**Member of:** HistoryFields collection object

**Remarks:**

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

**See Also:**

Count property
HistoryField object

**HistoryField object properties**

<table>
<thead>
<tr>
<th>Property name</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DisplayNameHeader property</td>
<td>Read-only</td>
<td>Returns the unique keys of the history items in this field.</td>
</tr>
<tr>
<td>FieldName property</td>
<td>Read-only</td>
<td>Returns the name of the history field.</td>
</tr>
<tr>
<td>Histories property</td>
<td>Read-only</td>
<td>Returns this history field’s collection of History objects.</td>
</tr>
</tbody>
</table>

**See Also:**

- HistoryFields property of the Entity object
- Entity object
- History object
- Histories collection object
- HistoryFields collection object
DisplayNameHeader property

Returns the unique keys of the history items in this field.

**VB Syntax:**

```vbnet
historyField.DisplayNameHeader
```

**Perl Syntax:**

```perl
$historyField->GetDisplayNameHeader();
```

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>field</code></td>
<td>A HistoryField object, representing one field of a record.</td>
</tr>
</tbody>
</table>

**Return value**

A Variant containing an Array whose elements are Strings. Each String contains the unique key of the corresponding item in the field's collection of Histories objects.

**Member of:** HistoryField object

**Remarks:**

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.

**See Also:**

FieldName property
FieldName property

Returns the name of the history field.

**VB Syntax:**

```
historyField.FieldName
```

**Perl Syntax:**

```
$historyField->GetFieldName();
```

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>field</code></td>
<td>A HistoryField object, representing one field of a record.</td>
</tr>
<tr>
<td><code>Return value</code></td>
<td>A String that contains the name of the field.</td>
</tr>
</tbody>
</table>

**Member of:** HistoryField object

**Remarks:**

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

**See Also:**

DisplayNameHeader property
Histories property

Returns this history field’s collection of History objects.

**VB Syntax:**

`historyField.Histories`

**Perl Syntax:**

`$historyField->GetHistories();`

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>historyField</code></td>
<td>A HistoryField object, representing one history field of a record.</td>
</tr>
<tr>
<td><code>Return value</code></td>
<td>A Histories collection object, which itself contains a set of History object objects.</td>
</tr>
</tbody>
</table>

**Member of:** HistoryField object

**Remarks:**

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

**See Also:**

Histories collection object
History object

**History object properties**

<table>
<thead>
<tr>
<th>Property name</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value property</td>
<td>Read-only</td>
<td>Returns the String that contains information about one modification to a record, as displayed on one line of a history field.</td>
</tr>
</tbody>
</table>

**See Also:**
- Histories collection object
- HistoryField object
- HistoryFields collection object
**Value property**

Returns the String that contains information about one modification to a record, as displayed on one line of a history field.

**VB Syntax:**

`history.Value`

**Perl Syntax:**

`$history->GetValue();`

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>History</td>
<td>A History object, representing one modification to a record.</td>
</tr>
<tr>
<td>Return value</td>
<td>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.</td>
</tr>
</tbody>
</table>

**Member of:** History object

**Remarks:**

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

**See Also:**

History object
Histories collection object

**Properties**

<table>
<thead>
<tr>
<th>Property name</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count property</td>
<td>Read-only</td>
<td>Returns the number of items in the collection.</td>
</tr>
</tbody>
</table>

**Methods**

<table>
<thead>
<tr>
<th>Method name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Item method</td>
<td>Returns the specified item in the collection.</td>
</tr>
</tbody>
</table>

**See Also:**

- History object
- HistoryField object
- HistoryFields collection object
Count property

Returns the number of items in the collection.

**VB Syntax:**

`collection.Count`

**Perl Syntax:**

`$collection->Count();`

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>collection</td>
<td>A Histories collection object, representing the set of history entries in one history field of a record.</td>
</tr>
<tr>
<td><strong>Return value</strong></td>
<td>A Long indicating the number of items in the collection object. This method returns zero if the collection contains no items.</td>
</tr>
</tbody>
</table>

**Member of:** Histories collection object

**Remarks:**

This property is read-only.

**See Also:**

Item method
**Item method**

Returns the specified item in the collection.

**VB Syntax:**

\`
\textit{collection}\texttt{.Item itemNum}
\textit{collection}\texttt{.Item name}
\`'

**Perl Syntax:**

\`
\texttt{$collection->Item(itemNum)};
\texttt{$collection->ItemByName(name)};
\`'

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textit{collection}</td>
<td>A Histories collection object, representing the set of history entries in one history field of a record.</td>
</tr>
<tr>
<td>\textit{itemNum}</td>
<td>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.</td>
</tr>
<tr>
<td>\textit{name}</td>
<td>A String that serves as a key into the collection. This string corresponds to the value of the desired History object.</td>
</tr>
<tr>
<td>\textit{Return value}</td>
<td>The History object at the specified location in the collection.</td>
</tr>
</tbody>
</table>

**Member of:** Histories collection object

**Remarks:**

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

**See Also:**

Count property
HookChoices object

*HookChoices object methods*

<table>
<thead>
<tr>
<th>Method name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddItem method</td>
<td>Adds a new item to the list of choices created by a CHOICE_LIST hook.</td>
</tr>
<tr>
<td>Sort method</td>
<td>Sorts the entries in the choice list.</td>
</tr>
</tbody>
</table>

*See Also:*

FieldInfo object
Hook Choices Code Example
AddItem method

Adds a new item to the list of choices created by a CHOICE_LIST hook.

**VB Syntax:**

```
choices.AddItem newChoice
```

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>choices</td>
<td>A special HookChoices object; see the remarks below.</td>
</tr>
<tr>
<td>newChoice</td>
<td>A String containing the new text to be added to the list of choices displayed to the user.</td>
</tr>
</tbody>
</table>

**Return value** None.

**Member of:** HookChoices object

**Remarks:**

The pre-existing HookChoices object is stored in a variable called choices that is visible only within a CHOICE_LIST hook. In the syntax section of this method, choices is a variable name that must be typed literally; it is not a placeholder for an arbitrary name or expression. This is the only way to access a HookChoices object.

A CHOICE_LIST hook should call this method repeatedly to build up a list of choices for the user. The object contains no items when you first access it. Add items in the order in which you want them to appear, because the list is not automatically sorted.

There is no corresponding RemoveItem method. Duplicate items are not automatically removed, but empty values are.

**See Also:**

HookChoices object
Hook Choices Code Example
Sort method

Sorts the entries in the choice list.

**VB Syntax:**

`choices.Sort [sortAscending]`

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>choices</code></td>
<td>A special HookChoices object; see the remarks below.</td>
</tr>
<tr>
<td><code>sortAscending</code></td>
<td>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.</td>
</tr>
</tbody>
</table>

**Return value**

None.

**Member of:** HookChoices object

**Remarks:**

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.

**See Also:**

AddItem method
Link object

**Link object methods**

<table>
<thead>
<tr>
<th>Method name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetChildEntity method</td>
<td>Returns the Entity object that is the child (duplicate) in a pair of linked Entity objects.</td>
</tr>
<tr>
<td>GetChildEntityDef method</td>
<td>Returns the EntityDef object that is the template for the child (duplicate) Entity object.</td>
</tr>
<tr>
<td>GetChildEntityDefName method</td>
<td>Returns the name of the EntityDef object that is the template for the child (duplicate) Entity object.</td>
</tr>
<tr>
<td>GetChildEntityID method</td>
<td>Returns the ID String of the Entity object that is the child (duplicate) in a pair of linked Entity objects.</td>
</tr>
<tr>
<td>GetParentEntity method</td>
<td>Returns the record that is the parent (original) in a pair of linked Entity objects.</td>
</tr>
<tr>
<td>GetParentEntityDef method</td>
<td>Returns the EntityDef object that is the template for the parent (original) in a pair of linked Entity objects.</td>
</tr>
<tr>
<td>GetParentEntityDefName method</td>
<td>Returns the name of the EntityDef object that is the template for the parent (original) Entity object.</td>
</tr>
<tr>
<td>GetParentEntityID method</td>
<td>Returns the ID String of the Entity object that is the parent (original) in a pair of linked Entity objects.</td>
</tr>
</tbody>
</table>

**See Also:**

GetAllDuplicates method of the Entity object
GetDuplicates method of the Entity object
HasDuplicates method of the Entity object
IsDuplicate method of the Entity object
Entity object
GetChildEntity method

Returns the Entity object that is the child (duplicate) in a pair of linked Entity objects.

**VB Syntax:**

`link.GetChildEntity`

**Perl Syntax:**

`$link->GetChildEntity();`

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>link</code></td>
<td>A Link object, which connects a parent and child Entity object to each other.</td>
</tr>
</tbody>
</table>

| Return value | The Entity object that is the child (duplicate). |

**Member of:** Link object

**See Also:**

GetAllDuplicates method of the Entity object
GetDuplicates method of the Entity object
IsOriginal method of the Entity object
Entity object
Duplicates Code Example
GetChildEntityDef method

Returns the EntityDef object that is the template for the child (duplicate) in a pair of linked Entity objects.

**VB Syntax:**

```
link.GetChildEntityDef
```

**Perl Syntax:**

```
$link-&gt;GetChildEntityDef();
```

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>link</td>
<td>A Link object, which connects a parent and child Entity object to each other.</td>
</tr>
</tbody>
</table>

**Return value**

An EntityDef object representing the record type of the child (duplicate) record.

**Member of:** Link object

**See Also:**

GetParentEntityDef method
GetEntityDef method of the Session object
Session object
**GetChildEntityDefName method**

Returns the name of the EntityDef object that is the template for the child (duplicate) Entity object.

**VB Syntax:**

`link.GetChildEntityDefName`

**Perl Syntax:**

`$link->GetChildEntityDefName();`

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>link</code></td>
<td>A Link object, which connects a parent and child Entity object to each other.</td>
</tr>
</tbody>
</table>

**Return value**

A String containing the name of the EntityDef object that was used as a template for the child (duplicate) Entity object.

**Member of:** Link object

**See Also:**

GetParentEntityDefName method
GetEntityDefName method of the Entity object
Entity object
**GetChildEntityID method**

Returns the ID String of the Entity object that is the child (duplicate) in a pair of linked Entity objects.

**VB Syntax:**

`link`.GetChildEntityID

**Perl Syntax:**

`$link->GetChildEntityID();`

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>link</code></td>
<td>A Link object, which connects a parent and child Entity object to each other.</td>
</tr>
<tr>
<td><strong>Return value</strong></td>
<td>A String that identifies the child (duplicate) Entity object. This ID is the unique key returned by the GetDisplayName method of Entity.</td>
</tr>
</tbody>
</table>

**Member of:** Link object

**See Also:**

GetParentEntityID method
GetDisplayName method of the Entity object
Entity object
GetParentEntity method

Returns the record that is the parent (original) in a pair of linked Entity objects.

VB Syntax:

`link.GetParentEntity`

Perl Syntax:

`$link->GetParentEntity();`

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>link</code></td>
<td>A Link object, which connects a parent and child Entity object to each other.</td>
</tr>
</tbody>
</table>

Return value The Entity object that is the parent (original).

Member of: Link object

See Also:

GetOriginal method of the Entity object
IsDuplicate method of the Entity object
Entity object
GetParentEntityDef method

Returns the EntityDef object that is the template for the parent (original) in a pair of linked Entity objects.

**VB Syntax:**

`link.GetParentEntityDef`

**Perl Syntax:**

`$link->GetParentEntityDef();`

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>link</td>
<td>A Link object, which connects a parent and child Entity object to each other.</td>
</tr>
</tbody>
</table>

**Return value**

An EntityDef object representing the record type of the parent (original) record.

**Member of:** Link object

**See Also:**

GetChildEntityDef method  
GetEntityDef method of the Session object  
Session object
GetParentEntityDefName method

Returns the name of the EntityDef object that is the template for the parent (original) Entity object.

**VB Syntax:**

`link.GetParentEntityDefName`

**Perl Syntax:**

`$link->GetParentEntityDefName();`

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>link</code></td>
<td>A Link object, which connects a parent and child Entity object to each other.</td>
</tr>
<tr>
<td><strong>Return value</strong></td>
<td>A String containing the name of the EntityDef object that was used as a template for the parent (original) Entity object.</td>
</tr>
</tbody>
</table>

**Member of:** Link object

**See Also:**

GetChildEntityDefName method
GetEntityDefName method of the Entity object
Entity object
GetParentEntityID method

Returns the ID String of the Entity object that is the parent (original) in a pair of linked Entity objects.

**VB Syntax:**

`link.GetParentEntityID`

**Perl Syntax:**

`$link->GetParentEntityID();`

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>link</code></td>
<td>A Link object, which connects a parent and child Entity object to each other.</td>
</tr>
</tbody>
</table>

**Return value**

The String that identifies the parent (original) Entity object. This ID is the unique key returned by the GetDisplayName method of Entity.

**Member of:** Link object

**See Also:**

GetChildEntityID method  
GetDisplayName method of the Entity object  
Entity object
OleMailMsg object

**OleMailMsg object methods**

<table>
<thead>
<tr>
<th>Method name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddBcc method</td>
<td>Add the e-mail address of a blind carbon-copy recipient to the mail message.</td>
</tr>
<tr>
<td>AddCc method</td>
<td>Add the e-mail address of a carbon-copy recipient to the mail message.</td>
</tr>
<tr>
<td>AddTo method</td>
<td>Add the e-mail address of a primary recipient to the mail message.</td>
</tr>
<tr>
<td>ClearAll method</td>
<td>Resets the contents of the mail message object.</td>
</tr>
<tr>
<td>Deliver method</td>
<td>Delivers the mail message.</td>
</tr>
<tr>
<td>MoreBody method</td>
<td>Appends additional body text to the mail message.</td>
</tr>
<tr>
<td>SetBody method</td>
<td>Sets the body text of the mail message.</td>
</tr>
<tr>
<td>SetFrom method</td>
<td>Sets the return address of the mail message.</td>
</tr>
<tr>
<td>SetSubject method</td>
<td>Sets the subject line of the e-mail message.</td>
</tr>
</tbody>
</table>
AddBcc method

Add the e-mail address of a blind carbon-copy recipient to the mail message.

**VB Syntax:**

```
OleMailMsg.AddBcc newAddress
```

**Perl Syntax:**

```
$OleMailMsg->AddBcc(newAddress);
```

**Identifier** | **Description** |
--- | --- |
OleMailMsg | An OleMailMsg object, representing the mail message to be sent. |
newAddress | A String containing the e-mail address of the recipient. |

**Return value** | None. |

**Member of:** OleMailMsg object

**Remarks:**

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.

**See Also:**

AddCc method  
AddTo method  
ClearAll method  
SetFrom method  
SetSubject method
AddCc method

Add the e-mail address of a carbon-copy recipient to the mail message.

**VB Syntax:**

```
OleMailMsg.AddCc newAddress
```

**Perl Syntax:**

```
$OleMailMsg->AddCc(newAddress);
```

### Identifier Description

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OleMailMsg</td>
<td>An OleMailMsg object, representing the mail message to be sent.</td>
</tr>
<tr>
<td>newAddress</td>
<td>A String containing the e-mail address of the recipient.</td>
</tr>
</tbody>
</table>

**Return value**

None.

**Member of:** OleMailMsg object

**Remarks:**

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.

**See Also:**

AddBcc method  
AddTo method  
ClearAll method  
SetFrom method  
SetSubject method
AddTo method

Add the e-mail address of a primary recipient to the mail message.

**VB Syntax:**

```vbnet
OleMailMsg.AddTo newAddress
```

**Perl Syntax:**

```perl
$OleMailMsg->AddTo(newAddress);
```

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OleMailMsg</td>
<td>An OleMailMsg object, representing the mail message to be sent.</td>
</tr>
<tr>
<td>newAddress</td>
<td>A String containing the e-mail address of the recipient.</td>
</tr>
</tbody>
</table>

**Return value**

None.

**Member of:** OleMailMsg object

**Remarks:**

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.

**See Also:**

AddBcc method  
AddCc method  
ClearAll method  
SetFrom method  
SetSubject method
ClearAll method

Resets the contents of the mail message object.

**VB Syntax:**

```
OleMailMsg.ClearAll
```

**Peri Syntax:**

```
$OleMailMsg->ClearAll();
```

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>OleMailMsg</code></td>
<td>An OleMailMsg object, representing the mail message to be sent.</td>
</tr>
<tr>
<td><code>Return value</code></td>
<td>None.</td>
</tr>
</tbody>
</table>

**Member of:** OleMailMsg object

**Remarks:**

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.

**See Also:**

AddBcc method  
AddCc method  
AddTo method  
MoreBody method  
SetBody method  
SetFrom method  
SetSubject method
**Deliver method**

Delivers the mail message.

**VB Syntax:**

`OleMailMsg.Deliver`

**Perl Syntax:**

`$OleMailMsg->Deliver();`

**Identifier** | **Description**
--- | ---
`OleMailMsg` | An OleMailMsg object, representing the mail message to be sent.

**Return value** | A Long indicating the success or failure of the delivery. A value of 1 indicates that the message was sent successfully. A value of 0 indicates that the message could not be delivered.

**Member of:** OleMailMsg object

**Remarks:**

After calling this method, you can make changes to the object without affecting the e-mail message that was just sent.

**See Also:**

- AddBcc method
- AddCc method
- AddTo method
- ClearAll method
- MoreBody method
- SetBody method
- SetFrom method
- SetSubject method
MoreBody method

Appends additional body text to the mail message.

**VB Syntax:**

`OleMailMsg.MoreBody bodyText`

**Perl Syntax:**

`$OleMailMsg->MoreBody(bodyText);`

### Identifier Description

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OleMailMsg</td>
<td>An OleMailMsg object, representing the mail message to be sent.</td>
</tr>
<tr>
<td>bodyText</td>
<td>A String containing the body text to add to the mail message.</td>
</tr>
</tbody>
</table>

**Return value** None.

**Member of:** OleMailMsg object

**Remarks:**

Use this method to add body text above and beyond what you added with the SetBody method. You can call this method as many times as you like. Each call to this method appends the specified text to the end of the message content.

This method does not add end-of-line characters or any other formatting characters when appending the text; you must add these characters yourself to the string you pass in to the bodyText parameter.

**See Also:**

ClearAll method
SetBody method
SetBody method

Sets the body text of the mail message.

**VB Syntax:**

```
OleMailMsg.SetBody bodyText
```

**Perl Syntax:**

```perl
$OleMailMsg->SetBody($bodyText);
```

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OleMailMsg</td>
<td>An OleMailMsg object, representing the mail message to be sent.</td>
</tr>
<tr>
<td>bodyText</td>
<td>A String containing the main body text of the mail message.</td>
</tr>
</tbody>
</table>

**Return value**

None.

**Member of:** OleMailMsg object

**Remarks:**

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

**See Also:**

ClearAll method
MoreBody method
SetFrom method

Sets the return address of the mail message.

**VB Syntax:**

\[\text{OleMailMsg.SetFrom returnAddress}\]

**Perl Syntax:**

\[$\text{OleMailMsg}\rightarrow\text{SetFrom}(\text{returnAddress})\];

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OleMailMsg</td>
<td>An OleMailMsg object, representing the mail message to be sent.</td>
</tr>
<tr>
<td>returnAddress</td>
<td>A String containing the e-mail address to add to the From field of the mail message.</td>
</tr>
</tbody>
</table>

**Return value**

None.

**Member of:** OleMailMsg object

**Remarks:**

If you do not call this method, ClearQuest automatically sets the return address to the e-mail address of the submitter of the record. You can call this method only once to add a return address to the e-mail message.

**See Also:**

AddBcc method
AddCc method
AddTo method
ClearAll method
SetSubject method
SetSubject method

Sets the subject line of the e-mail message.

**VB Syntax:**

```
OleMailMsg.SetSubject subjectText
```

**Perl Syntax:**

```
$OleMailMsg->SetSubject(subjectText);
```

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OleMailMsg</td>
<td>An OleMailMsg object, representing the mail message to be sent.</td>
</tr>
<tr>
<td>subjectText</td>
<td>A String containing the subject text to add to the message.</td>
</tr>
</tbody>
</table>

**Return value** None.

**Member of:** OleMailMsg object

**Remarks:**

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.

**See Also:**

AddBcc method
AddCc method
AddTo method
ClearAll method
SetFrom method
CHARTMGR object

ChartMgr object properties

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

ChartMgr object methods

<table>
<thead>
<tr>
<th>Method name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MakeJPEG method</td>
<td>Creates a JPEG image of the chart.</td>
</tr>
<tr>
<td>MakePNG method</td>
<td>Creates a PNG image of the chart.</td>
</tr>
<tr>
<td>SetResultSet method</td>
<td>Sets the result set used to generate the chart.</td>
</tr>
</tbody>
</table>

See Also:

ResultSet Object
WORKSPACE object
**GrayScale property**

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

**VB Syntax:**

```vbnet
chartMgr.GrayScale
chartMgr.GrayScale = isGrayScale
```

**Perl Syntax:**

```perl
$chartMgr->GetGrayScale();
$chartMgr->SetGrayScale(boolean_value);
```

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>chartMgr</code></td>
<td>The CHARTMGR object associated with the current session.</td>
</tr>
<tr>
<td><code>isGrayScale</code></td>
<td>True if the image should be created in grayscale, otherwise False.</td>
</tr>
</tbody>
</table>

**Return value**

True if the image should be rendered as a grayscale image, otherwise False to indicate the image will be rendered in color.

**Member of:** CHARTMGR object

**Remarks:**

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

**See Also:**

MakeJPEG method
MakePNG method
Height property

Sets or gets the height of the image.

**VB Syntax:**

```vbnet
chartMgr.Height
chartMgr.Height newHeight
```

**Perl Syntax:**

```perl
$chartMgr->GetHeight();
$chartMgr->SetHeight(integer_for_height_in_pixels);
```

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>chartMgr</code></td>
<td>The CHARTMGR object associated with the current session.</td>
</tr>
<tr>
<td><code>newHeight</code></td>
<td>An INT indicating the new height of the image in pixels.</td>
</tr>
<tr>
<td><strong>Return value</strong></td>
<td>An INT indicating the height of the image in pixels</td>
</tr>
</tbody>
</table>

**Member of:** CHARTMGR object

**Remarks:**

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

**See Also:**

Width property
MakeJPEG method
MakePNG method
Interlaced property

Sets or returns whether or not PNG images are interlaced.

**VB Syntax:**

```vbnet
cartMgr.Interlaced
cartMgr.Interlaced = isInterlaced
```

**Perl Syntax:**

```perl
$chartMgr->GetInterlaced();
$chartMgr->SetInterlaced(boolean_value);
```

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>chartMgr</code></td>
<td>The CHARTMGR object associated with the current session.</td>
</tr>
<tr>
<td><code>isInterlaced</code></td>
<td>A Bool indicating whether or not PNG images should be created in multiple passes.</td>
</tr>
<tr>
<td><strong>Return value</strong></td>
<td>True if the MakePNG method will create an interlaced PNG image, otherwise False.</td>
</tr>
</tbody>
</table>

**Member of:** CHARTMGR object

**Remarks:**

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

**See Also:**

Progressive property
MakePNG method
OptimizeCompression property

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

Syntax:

```
chartMgr.OptimizeCompression
chartMgr.OptimizeCompression useCompression
```

Perl Syntax:

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

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>chartMgr</code></td>
<td>The CHARTMGR object associated with the current session.</td>
</tr>
<tr>
<td><code>useCompression</code></td>
<td>A Bool indicating whether or not the image compression should be optimized</td>
</tr>
<tr>
<td><code>Return value</code></td>
<td>True if the image compression should be optimized, otherwise False.</td>
</tr>
</tbody>
</table>

Member of: CHARTMGR object

Remarks:

By default, this property is set to True.

See Also:

Quality property
Progressive property

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

Syntax:

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

Perl Syntax:

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

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>chartMgr</td>
<td>The CHARTMGR object associated with the current session.</td>
</tr>
<tr>
<td>isProgressive</td>
<td>A Bool indicating whether or not JPEG images should be created in multiple passes.</td>
</tr>
</tbody>
</table>

**Return value**

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

**Member of:** CHARTMGR object

**Remarks:**

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

**See Also:**

Interlaced property
MakeJPEG method
Quality property

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

Syntax:

chartMgr.Quality
chartMgr.Quality newValue

Perl Syntax:

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

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>chartMgr</td>
<td>The CHARTMGR object associated with the current session.</td>
</tr>
<tr>
<td>newValue</td>
<td>An INT between 1 and 100 indicating the new compression factor of the image.</td>
</tr>
<tr>
<td>Return value</td>
<td>An INT between 1 and 100 indicating the compression factor of the image.</td>
</tr>
</tbody>
</table>

Member of: CHARTMGR object

Remarks:

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.

See Also:

OptimizeCompression property
Width property

Sets or gets the width of the image.

VB Syntax:

`chartMgr.Width`
`chartMgr.Width newHeight`

Perl Syntax:

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

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>chartMgr</code></td>
<td>The CHARTMGR object associated with the current session.</td>
</tr>
<tr>
<td><code>newWidth</code></td>
<td>An INT indicating the new width of the image in pixels.</td>
</tr>
<tr>
<td><strong>Return value</strong></td>
<td>An INT indicating the new width of the image in pixels.</td>
</tr>
</tbody>
</table>

Member of: CHARTMGR object

Remarks:

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

See Also:

Height property
MakeJPEG method
MakePNG method
MakeJPEG method

Creates a JPEG image of the chart.

**VB Syntax:**

`chartMgr.MakeJPEG chartName`

**Perl Syntax:**

`$chartMgr->MakeJPEG(chartName);`

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>chartMgr</td>
<td>The CHARTMGR object associated with the current session.</td>
</tr>
<tr>
<td>chartName</td>
<td>A String containing the pathname of the chart to use when generating the image.</td>
</tr>
</tbody>
</table>

**Return value**

True if the image was created successfully, otherwise False.

**Member of:** CHARTMGR object

**Remarks:**

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

**See Also:**

Height property  
OptimizeCompression property  
Progressive property  
Quality property  
Width property  
SetResultSet method
MakePNG method

Creates a PNG image of the chart.

**VB Syntax:**

`chartMgr.MakePNG chartName`

**Perl Syntax:**

`
$chartMgr->MakePNG(chartName);
`

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>chartMgr</code></td>
<td>The CHARTMGR object associated with the current session.</td>
</tr>
<tr>
<td><code>chartName</code></td>
<td>A String containing the pathname of the chart to use when generating the image.</td>
</tr>
</tbody>
</table>

**Return value**  None.

**Member of:** CHARTMGR object

**Remarks:**

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

**See Also:**

Height property
Interlaced property
OptimizeCompression property
Quality property
Width property
SetResultSet method
**SetResultSet method**

Sets the result set used to generate the chart.

**VB Syntax:**

`chartMgr.SetResultSet resultSet`

**Perl Syntax:**

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

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>chartMgr</code></td>
<td>The CHARTMGR object associated with the current chartMgr.</td>
</tr>
<tr>
<td><code>resultSet</code></td>
<td>The ResultSet object containing the data to use when generating charts.</td>
</tr>
</tbody>
</table>

**Return value**: None.

**Member of**: CHARTMGR object

**Remarks:**

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.

**See Also:**

- MakeJPEG method
- MakePNG method
- Execute method of the ResultSet object
- ResultSet Object
ReportMgr object

_Authored by: ftvmsbh, 08/22/2020 at 14:53:03_  
_Author's name: ftvmsbh_  
_Author's contact: ftvmsbh@netserv.org_  
_Author's office: 9th Floor, Building T, NetServ, 123 Main Street, Anytown USA_  
_Author's phone: 555-1234_  
_Author's email: ftvmsbh@netserv.org_  
_Author's status: Manager_  
_Author's notes: This report was prepared for internal use only._

*ReportMgr object methods*

<table>
<thead>
<tr>
<th>Method name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ExecuteReport method</td>
<td>Executes the report and generates the resulting HTML file.</td>
</tr>
<tr>
<td>GetQueryDef method</td>
<td>Returns the QueryDef object associated with the report.</td>
</tr>
<tr>
<td>SetHTMLFileName method</td>
<td>Sets the output file name for the report.</td>
</tr>
</tbody>
</table>

*See Also:*

WORKSPACE object
ExecuteReport method

Executes the current report and generates the resulting HTML file.

**VB Syntax:**

`reportMgr.ExecuteReport`

**Perl Syntax:**

`$reportMgr->ExecuteReport();`

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>chartMgr</code></td>
<td>The ReportMgr object associated with the current session.</td>
</tr>
</tbody>
</table>

**Return value**

None.

**Member of:** ReportMgr object

**Remarks:**

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.

**See Also:**

GetReportMgr method
SetHTMLFileName method
GetQueryDef method

Returns the QueryDef object associated with the report.

**VB Syntax:**

```
reportMgr.GetQueryDef
```

**Perl Syntax:**

```
$reportMgr->GetQueryDef();
```

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>chartMgr</code></td>
<td>The ReportMgr object associated with the current session.</td>
</tr>
<tr>
<td>Return value</td>
<td>The QueryDef object associated with the report.</td>
</tr>
</tbody>
</table>

**Member of:** ReportMgr object

**Remarks:**

You can use the returned QueryDef object to get information about the query that was used to generate the report.

**See Also:**

QueryDef object
**SetHTMLFileName method**

Sets the output file name for the report.

**VB Syntax:**

`reportMgr.SetHTMLFileName htmlPath`

**Perl Syntax:**

`$reportMgr->SetHTMLFileName(htmlPath);`

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>chartMgr</td>
<td>The ReportMgr object associated with the current session.</td>
</tr>
<tr>
<td>htmlPath</td>
<td>A String containing the pathname for the report file.</td>
</tr>
<tr>
<td><strong>Return value</strong></td>
<td>None.</td>
</tr>
</tbody>
</table>

**Member of:** ReportMgr object

**Remarks:**

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.

**See Also:**

ExecuteReport method
WORKSPACE object

Workspace object methods

<table>
<thead>
<tr>
<th>Method name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetAllQueriesList method</td>
<td>Returns the complete list of queries in the workspace.</td>
</tr>
<tr>
<td>GetChartDef method</td>
<td>Returns the QueryDef object associated with the specified chart.</td>
</tr>
<tr>
<td>GetChartList method</td>
<td>Returns the specified list of charts.</td>
</tr>
<tr>
<td>GetChartMgr method</td>
<td>Returns the CHARTMGR object associated with the current session.</td>
</tr>
<tr>
<td>GetQueryDef method</td>
<td>Returns the QueryDef object associated with the specified workspace query.</td>
</tr>
<tr>
<td>GetQueryList method</td>
<td>Returns the specified list of workspace queries.</td>
</tr>
<tr>
<td>GetReportList method</td>
<td>Returns the specified list of reports.</td>
</tr>
<tr>
<td>GetReportMgr method</td>
<td>Returns the ReportMgr object associated with the current session.</td>
</tr>
<tr>
<td>SaveQueryDef method</td>
<td>Saves the query to the specified location in the workspace.</td>
</tr>
<tr>
<td>SetSession method</td>
<td>Associates the specified Session object with this object.</td>
</tr>
<tr>
<td>SetUserName method</td>
<td>Sets the current user name when searching for queries, charts, or reports.</td>
</tr>
<tr>
<td>ValidateQueryDefName method</td>
<td>Verifies that the specified query name and path info are correct.</td>
</tr>
</tbody>
</table>

See Also:

GetWorkSpace method of the Session object
CHARTMGR object
ReportMgr object
Session object
GetAllQueriesList method

Returns the complete list of queries in the workspace.

**VB Syntax:**

```vbnet
workspace.GetAllQueriesList
```

**Perl Syntax:**

```perl
$workspace->GetAllQueriesList();
```

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>workspace</td>
<td>The Workspace object obtained from the current session.</td>
</tr>
</tbody>
</table>

**Return value**

An array of strings, each of which contains the pathname of a query.

**Member of:** WORKSPACE object

**Remarks:**

This method returns both the public queries defined by the ClearQuest administrator and personal queries created by individual users.

**See Also:**

GetQueryDef method
GetQueryList method
QueryDef object
GetChartDef method

Returns the QueryDef object associated with the specified chart.

VB Syntax:

`workspace.GetChartDef chartName`

Perl Syntax:

`$workspace->GetChartDef(chartName);`

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>workspace</code></td>
<td>The Workspace object obtained from the current session.</td>
</tr>
<tr>
<td><code>chartName</code></td>
<td>A String containing the workspace pathname of the chart.</td>
</tr>
<tr>
<td><code>Return value</code></td>
<td>The QueryDef object associated with the chart.</td>
</tr>
</tbody>
</table>

Member of: WORKSPACE object

Remarks:

You use this method to get the query information associated with the specified chart. 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.

See Also:

GetChartList method
GetChartMgr method
CHARTMGR object
QueryDef object
GetChartList method

Returns the specified list of charts.

**VB Syntax:**

```vbnet
workspace.GetChartList typeOfCharts
```

**Perl Syntax:**

```perl
$workspace->GetChartList(typeOfCharts);
```

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>workspace</td>
<td>The Workspace object obtained from the current session.</td>
</tr>
<tr>
<td>typeOfCharts</td>
<td>An INT indicating which types of charts should be returned. This value corresponds to one of the OLEWKSPCQUERYTYPE enumerated constants.</td>
</tr>
</tbody>
</table>

**Return value**
An array of Strings, each of which contains the pathname of a single chart.

**Member of:** WORKSPACE object

**Remarks:**

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 `OLEWKSPC_SYSTEMQUERIES` returns only the public charts defined by the ClearQuest administrator. Specifying the constant `OLEWKSPC_BOTHQUERIES` 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 `OLEWKSPC_USERQUERIES` for the `typeOfCharts` parameter.
See Also:

GetChartDef method
GetChartMgr method
SetUserName method
CHARTMGR object
GetChartMgr method

Returns the CHARTMGR object associated with the current session.

**VB Syntax:**

`workspace.GetChartMgr`

**Perl Syntax:**

`$workspace->GetChartMgr();`

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>workspace</code></td>
<td>The Workspace object obtained from the current session.</td>
</tr>
</tbody>
</table>

**Return value**

The CHARTMGR object associated with the current session.

**Member of:** WORKSPACE object

**Remarks:**

You can use the CHARTMGR object to generate charts and control the appearance of the output files.

**See Also:**

GetChartDef method
GetChartList method
CHARTMGR object
GetQueryDef method

Returns the QueryDef object associated with the specified workspace query.

VB Syntax:

workspace.GetQueryDef queryName

Perl Syntax:

$workspace->GetQueryDef(queryName);

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>workspace</td>
<td>The Workspace object obtained from the current session.</td>
</tr>
<tr>
<td>queryName</td>
<td>A String containing the workspace pathname of the query</td>
</tr>
<tr>
<td>Return value</td>
<td>The QueryDef object associated with the query.</td>
</tr>
</tbody>
</table>

Member of: WORKSPACE object

Remarks:

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.

See Also:

GetQueryList method
QueryDef object
GetQueryList method

Returns the specified list of workspace queries.

**VB Syntax:**

```vbnet
workspace.GetQueryList typeOfQuery
```

**Perl Syntax:**

```perl
$workspace->GetQueryList(typeOfQuery);
```

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>workspace</td>
<td>The Workspace object obtained from the current session.</td>
</tr>
<tr>
<td>typeOfQuery</td>
<td>An INT indicating which types of queries should be returned. This value corresponds to one of the The ValueStatus constants identify the status of a field. OLEWKSPCQUERYTYPE enumerated constants.</td>
</tr>
</tbody>
</table>

**Return value**

An array of Strings, each of which contains the pathname of a single query.

**Member of:** WORKSPACE object

**Remarks:**

This method returns the pathnames of the public or personal queries defined in the ClearQuest workspace. The `typeOfCharts` parameter lets you specify the type of queries to return. Specifying the constant `OLEWKSPCSYSTEMQUERIES` returns only the public queries defined by the ClearQuest administrator. Specifying the constant `OLEWKSPCBOTHQUERIES` returns a list of all of the queries in the workspace (including those of all users).

To return only the queries defined by a particular user, you must first set the current user name by calling the SetUserName method. You can then call this method, specifying the constant `OLEWKSPCUSERQUERIES` for the `typeOfCharts` parameter.
See Also:

GetQueryDef method
SetUserName method
QueryDef object
GetReportList method

Returns the specified list of reports.

**VB Syntax:**

`workspace.GetReportList typesOfReports`

**Perl Syntax:**

`$workspace->GetReportList(typesOfReports);`

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>workspace</code></td>
<td>The Workspace object obtained from the current session.</td>
</tr>
<tr>
<td><code>typesOfReports</code></td>
<td>An INT indicating which types of reports should be returned. This value corresponds to one of the The ValueStatus constants identify the status of a field. OLEWKSPCQUERYTYPE enumerated constants.</td>
</tr>
</tbody>
</table>

**Return value**

An array of Strings, each of which contains the pathname of a single report.

**Member of:** WORKSPACE object

**Remarks:**

This method returns the pathnames of the public or personal reports defined in the ClearQuest workspace. The `typeOfCharts` parameter lets you specify the type of reports to return. Specifying the constant OLEWKSPCSYSTEMREPORTS returns only the public reports defined by the ClearQuest administrator. Specifying the constant OLEWKSPCBothReports returns a list of all of the reports in the workspace (including those of all users).

To return only the reports defined by a particular user, you must first set the current user name by calling the SetUserName method. You can then call this method, specifying the constant OLEWKSPCUSERREPORTS for the `typeOfCharts` parameter.
See Also:

GetReportMgr method
ReportMgr object
GetReportMgr method

Returns the ReportMgr object associated with the current session.

**VB Syntax:**

`workspace.GetReportMgr reportName`

**Perl Syntax:**

`$workspace->GetReportMgr(reportName);`

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>workspace</code></td>
<td>The Workspace object obtained from the current session.</td>
</tr>
<tr>
<td><code>reportName</code></td>
<td>A String containing the name of the report to run with the returned ReportMgr object.</td>
</tr>
<tr>
<td><em>Return value</em></td>
<td>A ReportMgr object you can use to run the specified report.</td>
</tr>
</tbody>
</table>

**Member of:** WORKSPACE object

**Remarks:**

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.

**See Also:**

ReportMgr object
SaveQueryDef method

Saves the query to the specified location in the workspace.

**VB Syntax:**

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

**Perl Syntax:**

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

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>workspace</td>
<td>The Workspace object obtained from the current session.</td>
</tr>
<tr>
<td>qdefName</td>
<td>A String containing the name of the query.</td>
</tr>
<tr>
<td>qdefPath</td>
<td>A String containing the pathname of the folder in which you want to save the query.</td>
</tr>
<tr>
<td>queryDef</td>
<td>The QueryDef object representing the query you want to save.</td>
</tr>
<tr>
<td>overwrite</td>
<td>A Bool indicating whether this query should overwrite a query with the same name and path information.</td>
</tr>
</tbody>
</table>

**Return value**

None.

**Member of:** WORKSPACE object

**Remarks:**

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.
See Also:

GetQueryDef method
GetQueryList method
QueryDef object
SetSession method

Associates the specified Session object with this object.

**VB Syntax:**

```vbnet
workspace.SetSession sessionObj
```

**Perl Syntax:**

```perl
$workspace->SetSession(sessionObj);
```

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>workspace</td>
<td>The Workspace object obtained from the current session.</td>
</tr>
<tr>
<td>sessionObj</td>
<td>The Session object to associate with this object.</td>
</tr>
<tr>
<td>Return value</td>
<td>None.</td>
</tr>
</tbody>
</table>

**Member of:** WORKSPACE object

**Remarks:**

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.

**See Also:**

Session object
**SetUserName method**

Sets the current user name when searching for queries, charts, or reports.

**VB Syntax:**

`workspace.SetUserName userName`

**Perl Syntax:**

`$workspace->SetUserName(userName);`

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>workspace</code></td>
<td>The Workspace object obtained from the current session.</td>
</tr>
<tr>
<td><code>userName</code></td>
<td>A String containing the login ID of the user.</td>
</tr>
</tbody>
</table>

**Return value**

None.

**Member of:** WORKSPACE object

**Remarks:**

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.

**See Also:**

GetChartList method
GetQueryList method
GetReportList method
ValidateQueryDefName method

Verifies that the specified query name and path information are correct.

**VB Syntax:**

```vb
workspace.ValidateQueryDefName qdefName, qdefPath
```

**Perl Syntax:**

```perl
$workspace->ValidateQueryDefName(qdefName, qdefPath);
```

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>workspace</code></td>
<td>The Workspace object obtained from the current session.</td>
</tr>
<tr>
<td><code>qdefName</code></td>
<td>A String containing the name of the query.</td>
</tr>
<tr>
<td><code>qdefPath</code></td>
<td>A String containing the pathname of the folder containing the query.</td>
</tr>
</tbody>
</table>

**Return value**

None.

**Member of:** WORKSPACE object

**Remarks:**

You can use this method to ensure that the given name and path are valid in the workspace.

**See Also:**

SaveQueryDef method
Index

A
Accessing the fields of a record 38
Accessing the schema repository 37
Active property 405, 420
Add method 480
AddBcc method 558
AddCc method 559
AddFieldValue method 206
AddItem method 544
AddParamValue method 322
AddTo method 560
AddUser property 421
AdminSession object 54, 345
AppBuilder property 406
ApplyPropertyChanges method 389
Attachment object 66, 459
AttachmentField object 65, 471
AttachmentFields object 455
AttachmentFields property 202
Attachments collection object 66
Attachments object 477
Attachments property 472
AttachmentsFields collection object 65

B
BeginNewFieldUpdateGroup method 208
BuildEntity method 136
BuildField method 316
BuildFilter method 342
BuildFilterOperator method 318, 344
BuildQuery method 138
BuildResultSet method 140
BuildSQLQuery method 142

C
CHARTMGR object 74
CheckTimeoutInterval property 371
Choosing a scripting language 17
ClearAll method 561
ClearParamValues method 323
ClearQuest documentation set 14
Commit method 210
Committing entity objects to the database 40
Common API calls to get user information 52
Count Property 456
Count property 430, 434, 438, 442, 446, 450, 458, 530, 540
CreateDatabase method 355
CreateGroup method 357
CreateUser method 358
Creating a new record 34
Creating a result set 31
Creating queries 29

D
DatabaseDescription object 67, 487
DatabaseName property 374
Databases collection object 62
Databases property 347, 422
DBOLogin property 375
DBOPassword property 376
Defining your search criteria 30
Delete method 482
DeleteDatabase method 359
DeleteEntity method 144
DeleteFieldValue method 359
DeleteField method 212
Deliver method 562
Description property 377, 398, 460
DisplayName property 462
DisplayNameHeader property 474, 534
DoesTransitionExist method 283

E
EditEntity method 146
Editing an existing record 34
Email property 407
Ending a session (for external applications) 28
Ensuring that record data is current 36
Entities and Hooks 42
Entity object 38, 199
EntityDef object 44, 281
EntityDefs collection object 62
EventType property 502
Execute method 324

FieldInfo object 68, 507
FieldName property 476, 535
FileName property 464
FileSize property 466
Finding examples 14
FireNamedHook method 214
Fullname property 408

GetAccessibleDatabases method 149
GetActionDefNames method 285
GetActionDefType method 287
GetAllDuplicates method 218
GetAllFieldValues method 220
GetAllQueriesList method 584
GetAuxEntityDefNames method 151
GetChartDef method 585
GetChartList method 586
GetChartMgr method 588
GetChildEntity method 548
GetChildEntityDef method 549
GetChildEntityDefName method 550
GetChildEntityId method 551
GetColumnLabel method 326
GetColumnType method 327
GetColumnValue method 328
GetDatabase method 361
GetDatabaseConnectString method 488
GetDatabaseId method 490
GetDatabaseSetId method 492
GetDefaultEntityDef method 153
GetDescription method 494
GetDisplayName method 223
GetDuplicates method 225
GetEntity method 156
GetEntityByDbId method 158
GetEntityDef method 160
GetEntityDefName method 227
GetEntityDefNames method 164
GetFieldChoiceList method 229
GetFieldChoiceType method 231
GetFieldDefNames method 290
GetFieldDefType method 292
GetFieldMaxLength method 233
GetFieldNames method 234
GetFieldOriginalValue method 236
GetFieldReferenceEntityDef method 294
GetFieldRequiredness method 238
GetFieldsUpdatedThisAction method 240
GetFieldsUpdatedThisGroup method 242
GetFieldsUpdatedThisSetValue method 244
GetFieldType method 246
GetFieldValue method 248
GetGroup method 363
GetHookDefNames method 296
GetInstalledMasters method 166
GetInvalidFieldValues method 250
GetIsMaster method 496
GetLegalActionDefNames method 251
GetLocalFieldPathNames method 298
GetLogin method 498
GetMessageText method 508
GetName method 299, 509
GetNumberOfColumns method 330
GetNumberOfParams method 331
GetOriginal method 253
GetOriginalId method 255
GetParamChoiceList method 332
GetParamComparisonOperator method 333
GetParamFieldType method 334
GetParamLabel method 335
GetParamPrompt method 336
GetParentEntity method 552
GetParentEntityDef method 553
GetParentEntityDefName
method 554
GetParentEntityID method 555
GetPassword method 499
GetQueryDef method 589
GetQueryEntityDefNames method 168
GetQueryList method 590
GetReportList method 592
GetReportMgr method 594
GetReqEntityDefNames method 170
GetRequiredness method 510
GetServerInfo method 172
GetSession method 257
GetSessionDatabase method 173
GetSQL method 338
GetStateDefNames method 300
GetSubmitEntityDefNames method 174
Getting a Session Object 25
Getting entity objects 33
Getting schema repository objects 50
GetType method 259, 302, 511
GetUser method 365
GetUserEmail method 176
GetUserFullName method 178
GetUserGroups method 180
GetUserLoginName method 182
GetUserMiscInfo method 184
GetUserPhone method 186
GetValidationStatus method 512
GetValue method 513
GetValueAsList method 514
GetValueStatus method 516
GetWorkSpace method 188
Group object 60
Groups collection object 62
Groups property 349, 409

H
HasDuplicates method 261
HasValue method 189
Histories collection object 70
Histories object 539
Histories property 536
History object 69, 537
HistoryField object 69, 533
HistoryFields collection object 69
HistoryFields object 529
HistoryFields property 204
HookChoices object 71, 543

I
IsActionDefName method 304
IsAggregated property 310
IsDirty property 311
IsDuplicate method 264
IsEditable method 266
IsFieldDefName method 306
IsMetadataReadOnly method 190
IsOriginal method 268
IsStateDefName method 307
IsSystemOwnedFieldDefName method 308
Item Method 451, 457
Item method 431, 439, 443, 447, 484, 531, 541
ItemName property 503

K
Knowledge assumed 16

L
Link object 72, 547
Load method 468
Logging on to a database 26
Logging on to the schema repository 49
Logon method 367
LookupPrimaryEntityDefName method 339
LookupStateName method 270

M
MarkEntityAsDuplicate method 191
MiscInfo property 410
MoreBody method 563
MoveNext method 340
Moving through the result set 32

N
Name property 313, 378, 394, 411, 423
NameValue property 134
method 195
UnsubscribeAllDatabases method 417, 426
UnsubscribeDatabase method 418, 427
Updating user database
information 50
Upgrade method 391
UpgradeMasterUserInfo method 392
User object 59
UserLogon method 197
UserMaintainer property 415
Users property 353, 424
Using Perl 17
Using query filters 30
Using session-wide variables 27
Using the AdminSession object 49
Using this reference 15
Using this reference manual 14
Using VBScript 19

V
Validate method 279
ValidateQueryDefName method 599
ValidityChangedThisAction method 517
ValidityChangedThisGroup method 519
ValidityChangedThisSetValue method 521
Value property 538
ValueChangedThisAction method 522
ValueChangedThisGroup method 524
ValueChangedThisSetValue method 526
Vendor property 388
Viewing the contents of a record 35
Viewing the metadata of a record 36

W
Ways to use the ClearQuest API 16
Working with a result set 31
Working with duplicates 41
Working with multiple sessions 28
Working with queries 29