Using Perl with the Rational ClearQuest API

Tom Milligan
Jack Wilber

September 15, 2002

In the November 2001 Rational Edge, I published an article about

Using Perl with Rational ClearCase Automation Library (CAL). That article explored some of the benefits and challenges of using Perl to access information in a Rational® ClearCase® repository. You may remember that the sample application presented in that article used both CAL and the Rational® ClearQuest® API (Application Programming Interface) to create a report that combined data from both tools. However, because the article was focused on CAL, the use of the Rational ClearQuest API was not fully explored.

In this article, I will focus squarely on Rational ClearQuest and how to use the Rational ClearQuest API in Perl. In many ways, the discussion will parallel the CAL article. I'll start with a short technical review, offer some reasons for using the Rational ClearQuest API, and then walk through a simple Perl application that generates a report combining data from multiple Rational ClearQuest databases. If you haven't read the previous article, then you might want to review it, as much of the Perl-specific information is equally applicable to CAL and the Rational ClearQuest API. However, I will include a summary of all relevant issues here, in case you don't like to peruse the Rational Edge archives as much as I do.
The Traditional Quick Tech Review

If you are a software developer, there is an excellent chance that you are comfortable using APIs, and a fairly good chance that you are familiar with the other tools and technologies referenced in this discussion, namely Perl, the Component Object Model (COM) and Rational ClearQuest itself. But just in case:

- **Perl**, or Practical Extraction and Reporting Language, is an exceptionally versatile and popular programming language. Even a little knowledge of Perl goes a long way. Many software developers can read and understand Perl scripts, or write simple ones, after a quick glance at a Perl reference. Perl also has the advantage of being available for free from a number of sources. For more details on where to get Perl, and recommended Perl books, see the "Useful Tools" section of my previous Edge article "Using Perl with Rational ClearCase Automation Library (CAL)".
- **COM**, or Component Object Model, is a standard developed by Microsoft that defines a mechanism for software components to interact with one another. The Rational ClearQuest API is implemented as a COM library and as a Perl package. This article's sample application uses the COM library implementation, and accesses it from Perl using the Win32::OLE Perl package.
- **Rational ClearQuest**, by the way, is an extremely flexible defect and change tracking tool that is used by organizations to efficiently capture, track, and manage any type of change request throughout the development lifecycle on a variety of platforms.

Why Use the Rational ClearQuest API?

Rational ClearQuest is highly customizable and provides powerful querying and reporting capabilities via intuitive user interfaces on Windows and the Web as well as on UNIX and Linux. If you're now asking yourself the question, "Why would I ever need the ClearQuest API?" the answer is that most users will not need it. However, there are situations in which you might want to access Rational ClearQuest data programmatically. For example, if your organization wanted to integrate Rational ClearQuest with a custom tool that you built in-house, then you would likely use either the ClearQuest API or one of the simple import/export tools that are installed with Rational ClearQuest.

These tools are:

- ClearQuest Import Tool -- used to import data, including records, history, and attachments.
- ClearQuest Export Tool -- used to export ClearQuest data to a text file.
- ClearQuest Maintenance Tool -- used to set up and connect to a Rational ClearQuest schema repository.

However, these tools offer a much coarser granularity for manipulating the data in a ClearQuest database than does the ClearQuest client and ClearQuest Web interface. In particular they do not allow users to construct queries to filter data nor do they allow users to execute actions or change the values of individual fields. Additionally, they are not easily invoked from a script or a program. If
you are looking to do an occasional basic import or export, these tools offer ready-made solutions. However, for more complex tasks, the speed and flexibility of the ClearQuest API make it ideal for a wide variety of applications. In fact, there are some scenarios in which the ClearQuest API is the only practical solution, as we will see in the example project below. Other such scenarios include:

- Integration with a call-tracking application.
- Integration with a CM (configuration management) tool for which an integration doesn’t already exist.
- Customized integration with Rational ClearCase.
- Construction of a customized ClearQuest client (e.g., a lightweight client that exposes only partial functionality).

An Example Project Using the ClearQuest API

Not long ago, I came across a real-world problem that was just begging to be solved by a Perl script using the ClearQuest API. I was working with a Rational customer who wanted to generate a report that combined data from multiple databases.

But before I go into further detail, it will help to review some Rational ClearQuest terminology.

- In ClearQuest, a user database stores, in the form of records, all the data you want to collect, including enhancement requests and defects. It is called the user database because it contains the data entered and accessed by end users.
- The term schema refers to all the attributes associated with a database, including field definitions, field behaviors, the state transition table, actions, report formats, and forms.
- The schema repository is a master database that contains all the data associated with existing schemas.
- Finally, a database set, dbset, or connection consists of a schema repository and all of the databases associated with that repository.

Rational ClearQuest has always allowed more than one database containing user data to be associated with a single schema repository. The query and reporting features in ClearQuest, however, do not allow the user to create a report that spans multiple user databases. In addition, database sets -- which are relatively recent additions to Rational ClearQuest Maintenance Tool GUI -- increase the likelihood that an organization will have multiple user databases organized into multiple database sets. But again, there is no out-of-the-box facility in Rational ClearQuest to allow queries or reports that span these database sets and their associated databases.

As you might expect, however, the Rational ClearQuest API provides all the functionality needed to generate such a report, and Perl offers a way to leverage the API simply and efficiently. The sample Perl application for this article, called dbspan_report.pl, allows the user to select a set of database sets or user databases and construct a query by specifying record types (e.g., Defect, EnhancementRequest) and States (e.g., Submitted, Open). The application then generates a report that spans all selected databases for records that match the specified criteria and optionally creates an Excel spreadsheet with the results. (See Figure 1.)
Figure 1: The Perl Application `dbspan_report.pl` Uses the Rational ClearQuest API to Access Multiple User Databases and Multiple Database Sets

If you compare this application with the sample Perl application described in my ClearCase Automation Library article, you'll see similarities in architecture as well as implementation. Both sample applications allow options to be set using command line switches or via a Tk-based graphical user interface, or GUI (see Figure 2). Similarly, both applications allow the user to send the results either to standard output (i.e., the terminal) or to a newly created Excel spreadsheet. And finally, both applications generate reports by combining data from multiple sources. In the CAL sample application, the data sources were a Rational ClearCase database and a Rational ClearQuest database. In `dbspan_report.pl`, the data sources are two or more Rational ClearQuest databases.
What Does the Sample Application Look Like?

Before delving into the source code, let's take a quick look at the sample application from the user's perspective. Figure 3 shows the sample application, `dbspan_report.pl`, with some information filled in. I have selected two databases (CLSIC and SAMPL) for my query, entered my Login ID and Password (note that the password text is masked), specified ClearQuest States for the query (Opened and Submitted), and opted to have the output sent to an Excel spreadsheet. The ClearQuest Login ID that I provide must be valid in all the ClearQuest databases specified for the report; otherwise, the report will be incomplete because the sample application will not be able to log into some databases via the ClearQuest API to extract the desired information.
Because I didn't specify and record types on the command line, `dbspan_report.pl` uses the ClearQuest API to access the definition of each database and determine what Record Types and Record Type Families are available. Figure 4 shows the GUI that displays the available Record Types and Record Type Families and allows the user to select which ones to include in the report. Because the DBSet/Database/Record Type specification resembles a treelike structure, `dbspan_report.pl` uses the TK::Tree widget as the display and selection mechanism. This widget allows multiple Record Types and Record Type Families to be selected using the standard Windows CTRL and SHIFT key multiple select conventions. Note that the item selected for the CLSIC database is not a Record Type; it is a Record Type Family: "All_UCM_Activities." The item selected for the SAMPL Database is a "Defect" Record Type.
After I click "OK," the sample Perl application queries all of the selected databases for records that have the specified Record Types and States. The results are reported in Excel, as shown in Figure 5. The report includes the Record Type of each record found. As a result, even though I specified a Record Type Family as one of my criteria, I can see what the actual Record Type of each item is.
Wading Through the Code

Now let's take a look at the `dbspan_report.pl` source code to see how all of that is accomplished. Like the CAL sample application, the first important lines of `dbspan_report.pl` appear right after the initial comments:

```perl
use Tk;
use Tk::Tree;
use Getopt::Long;
use Win32::OLE;
require "dbset_api.pl";
```

These lines essentially *include* the following modules so they can be used in the program:

- Tk modules for the GUI.
- `Getopt::Long` module for parsing command line options.
- The `Win32::OLE` module for accessing COM objects such as the ClearQuest API, Excel, and a custom-engineered DLL (see Appendix), which is accessed by the *required* script `dbset_api.pl`.

After parsing the command line and doing some initialization, the program creates a ClearQuest session object, which will be used as the entry point to the ClearQuest API:
The two root objects in the ClearQuest API are the Session object and the AdminSession object. The Session object is used to create or access many of the other objects in the system, create new records or modify existing records, and create the query objects that enable searching of the database for a particular record (or set of records). In contrast, the AdminSession object provides access to the schema repository information. Because the sample application only deals with the contents of ClearQuest databases and not their schemas, it does not make use of the AdminSession object.

The code above instantiates a ClearQuest Session object named $CQSession. If the Session object cannot be created, then the script terminates or "dies" with an appropriate error message. The print ... $DEBUG line in the code above and throughout the example scripts can be used for rudimentary debugging. You can enable this verbose output by simply un-commenting the $DEBUG = 1 line located near the start of the script.

After the Session object is created, if the user has not specified which databases to query via the command line, the program uses the Session object to build a list of databases to populate the Tk Tree element of the GUI. As shown in the following code segment, this is done via the custom DLL described in Appendix:

```perl
#-----------------------------------------------------
# Create ClearQuest session object
#-----------------------------------------------------
print "Creating ClearQuest Session
" if $DEBUG;
$CQsession = Win32::OLE->new("CLEARQUEST.SESSION")
or die "Can't create ClearQuest session object via call to Win32::OLE->new(): $!";

@dbsetlist = getDbSetList($CQsession);
foreach $dbset (@dbsetlist) {
    $tree->add( $dbset, -text => $dbset );
    $MDB = getMasterDb($dbset,$CQsession);
    #
    # Now add the user databases as the second level in
    # the hierarchical Tree
    #
    @userdatabases = getUserDbList($dbset,$CQsession);
    foreach $userDb (@userdatabases) {
        $dbpath = "$dbset\/$userDb";
        $tree->add( $dbpath, -text => $userDb );
    }
}
```
At this point, if the user has not specified any Record Types, the script will login to each selected database individually, find all available Record Types, and display them so the user can select the ones he or she is interested in. The login is accomplished with a call to the Session object's `UserLogon()` method, and the list of Record Types is acquired through calls to the `GetEntityDefNames()` and `GetEntityDefFamilyNames()` methods.

```perl
# Now for each of these databases, find the record types
# defined in that database and add those record types to
# the tree list as the third level of hierarchy
#
foreach $db (@userdblist) {
    # log on to DB
    #-----------------------------------------------------
    # Log into ClearQuest
    #-----------------------------------------------------
    print "Specified DBSet/Database is $db\n" if $DEBUG;
    my @dbspecs = split ( ///, $db );
    
    # To get what record types are defined in each
    # database, we need to log in to each database
    #
    print "Logging onto ClearQuest Session\n" if $DEBUG;
    $CQsession->UserLogon("$opt_cqlogin", "$opt_cqpasswd",
        "$dbspecs[1]\", $AD_PRIVATE_SESSION,
        "$dbspecs[0]\" );
    
    # Get the names of the record types and add them
    # as the third level of the hierarchical tree.
    #
    $recordtypes = $CQsession->GetEntityDefNames();
    
    $typecount = @$recordtypes;
    for ( $rix = 0 ; $rix < $typecount ; $rix++ ) {
        $RecType = $recordtypes->[ $rix ];
        $rectree->add("$db/$RecType", -text => "$RecType" );
    }
    
    # Also get the names of record family types and add
    # them to the hierarchical tree
    #
    $familytypes = $CQsession->GetEntityDefFamilyNames();
    $familycount = @$familytypes;
    for ( $rix = 0 ; $rix < $familycount ; $rix++ ) {
        $FamType = $familytypes->[ $rix ];
        $rectree->add("$db/$FamType", -text => "$FamType" );
    }
}
MainLoop;
```

The script then does some validation of the input data, and ensures that the various lists of databases, States, and Record Types are formatted properly. If the user has chosen to send output to Excel, then the script instantiates an Excel Application object using the Win32::OLE object, just as it did for the ClearQuest session object:
$Excel  = Win32::OLE->new("Excel.Application")
    or die
    "Can't create Excel application object via call to Win32::OLE->new(): $!"

Now the script is ready to do its real work, which is done in three nested loops.

The basic algorithm for this section follows in bold, along with the associated ClearQuest API calls for each step.

For each database selected
$CQsession->UserLogon(
    "$loginid",         "$password",
    "$dbcomponents\[1\]", $AD_PRIVATE_SESSION,
    "$dbcomponents\[0\]"
);

For each Record Type specified
my ($QueryDef) = $CQsession->BuildQuery($recordtype);

Specify the fields to be returned by the query
$QueryDef->BuildField("id");
$QueryDef->BuildField("State");
$QueryDef->BuildField("headline");
$QueryDef->BuildField("Owner");
$QueryDef->BuildField("record_type");

Create a filter for the query using a Boolean "OR" operator
$QueryDef->BuildFilterOperator($AD_BOOL_OP_OR);

For each State specified
Add the state as an OR condition to the query filter
$FilterNode1->BuildFilter( "State", $AD_COMP_OP_EQ, "$st" );

Create a result set
my ($ResultSet) = $CQsession->BuildResultSet($QueryDef);
Execute the query
$ResultSet->Execute();

For each record in the result set
$status = $ResultSet->MoveNext();
Get the ID, State, headline, owner and Record Type
$id          = $ResultSet->GetColumnValue(1);
$state       = $ResultSet->GetColumnValue(2);
$headline    = $ResultSet->GetColumnValue(3);
$owner       = $ResultSet->GetColumnValue(4);
$record_type = $ResultSet->GetColumnValue(5);
Output these fields to Excel or standard output

If you have experience in writing code to access database information programmatically, the
structure, order, and general functionality of the above ClearQuest API calls will likely seem
familiar to you. For further details, the Rational ClearQuest API Reference, an invaluable resource,
is installed by default with Rational ClearQuest.

Similarly, for more details on Perl data structures and the data manipulations that take place
between ClearQuest API calls, see the second edition of the Perl CD Bookshelf from O'Reilly. Any
good Perl reference should suffice, as there is nothing particularly tricky about the code.
Whither Now?

Like the CAL sample application (changeset_report.pl), the dbspan_report.pl application is a fairly simple example of ClearQuest API use from Perl. It by no means demonstrates all of the functionality available in the ClearQuest API. If you have Rational ClearQuest installed, I encourage you to take a look at the online Rational ClearQuest API Reference to get a feel for what is possible through the API.

I also encourage you to try out dbspan_report.pl for yourself. Just remember: You need to use regsvr32 to register the DLL before you can use it. For example:

regsvr32 DRIVE:\PATH\TO\THE\LIBRARY\CQDBsets.dll cqperl -I"C:\program files\rational\clearquest\lib" dbspan_report.pl

If you are considering using Perl to access ClearQuest data via the ClearQuest API, I think you'll find both changeset_report.pl and dbspan_report.pl instructive and helpful. And even if you are not planning on any development yourself, you might find these utilities useful in their own right, just as they are.

Rational ClearQuest is an exceptionally flexible tool. And when combined with an extensive API and the power and versatility of Perl, the possibilities are practically limitless.

Appendix

A Visual Basic DLL

Notes

1 For further reading on COM, I recommend Essential COM by Don Box (Addison-Wesley, 1998).

2 If a ClearQuest administrator is customizing ClearQuest through the use of hooks, he or she is probably familiar with some aspects of the ClearQuest API, but end-users will likely never know it is there.