

▶ Using Perl with Rational ClearCase Automation Library (CAL)

by [Tom Milligan](#)

Software Engineering Specialist
Rational Services Organization

As a consultant for Rational, I spend a great deal of time working directly with Rational customers. Typically, I work together with Rational ClearCase® users to develop a customized solution to whatever unique challenge they are facing. Often in these encounters, I have found the sometimes overlooked ClearCase Automation Library (CAL) to be an especially handy tool for accessing and manipulating ClearCase data programmatically in a wide variety of situations.

Starting with ClearCase 4.0, both CAL and the CAL documentation are automatically installed with Rational ClearCase on Windows platforms. CAL provides a set of COM (Component Object Model) interfaces that are intended both as an integration platform and also as an API (Application Program Interface) that you can use to extend or customize Rational ClearCase. You can also use CAL to write scripts, stand-alone programs, or macros embedded in other applications. Because CAL is a COM-compliant library, you can use it from within any tool that can interface with COM, including Visual Basic, Visual C++, and Perl on Windows. Since COM is primarily a Windows-only technology, CAL is not available on UNIX platforms.

In this article, I'm going to talk about why you might want to use CAL, and some tools and resources that are useful for working with CAL in the Perl programming language. I'll also walk through a fairly simple Perl application that uses CAL as well as the Rational ClearQuest® API to generate a report combining ClearCase and ClearQuest data. An in-depth discussion of the Rational ClearQuest API is beyond the scope of this article, but it will be addressed in an upcoming edition of The Rational Edge.

A Very Quick Tech Review

- ▶ [subscribe](#)
- ▶ [contact us](#)
- ▶ [submit an article](#)
- ▶ [rational.com](#)
- ▶ [issue contents](#)
- ▶ [archives](#)
- ▶ [mission statement](#)
- ▶ [editorial staff](#)



In general, the concepts I'll discuss here are fairly straightforward and should be readily understandable by a technical audience. However, some basic familiarity -- or expertise -- in the following areas will help you get the most from this discussion:

- **Software configuration management concepts in general and Rational ClearCase in particular.** A quick review: Rational ClearCase is the market-leading tool for software asset management. Rational ClearQuest is a highly flexible defect and change tracking tool. Together, they enable Unified Change Management (UCM), Rational's out-of-the-box process for managing change at the activity level.
- **The Perl Programming Language.** Perl is a remarkably popular programming language. Relatively easy to learn and often available for free, it has been described as the duct tape of not only the Internet, but also the entire computing world. If you're looking to learn Perl or just brush up on it, I recommend *Learning Perl* (the book with the llama on the cover) and for a more advanced reference *Programming Perl* (the book with the camel on the cover), both from O'Reilly & Associates. The latter title is included in the Perl CD Bookshelf -- see [Useful Tools](#) below.
- **COM or Component Object Model.** Developed by Microsoft, COM is a standard that defines a mechanism for software components to interact with each other. COM is an object-oriented technology, and its main building blocks are objects, interfaces, methods, and properties. There are numerous excellent resources and books on COM², and even the more basic ones should give you enough information to start using CAL.

Why CAL?

You may already be familiar with another common method of accessing Rational ClearCase data from outside the Rational ClearCase user interface. The *cleartool* utility is a command-line interface that you can use to create, modify, and manage the information in Rational ClearCase VOBs (Versioned Object Bases) and views. The flexibility and ease of use of *cleartool* makes it an ideal solution for a wide range of problems. And in one sense, *cleartool* has always offered a kind of API to ClearCase, for both Windows and UNIX platforms. However, there are situations in which CAL offers a superior alternative to *cleartool* -- specifically when speed and performance are factors.

Certain languages are not well suited to invoking command-line tools and parsing the results; Visual Basic is a good example. Parsing results can be somewhat difficult in the C programming language as well. Perl, on the other hand, makes it easy to start command-line utilities and parse the results. But even when using Perl, you may want greater speed. Because CAL does not create a new process for each invocation, it can be considerably faster than a solution that uses *cleartool's* command-line interface. When a program invokes *cleartool*, the operating system must create a new process, and there is a significant amount of overhead associated with starting that new process -- allocating memory for it, creating a new entry in the process table, and so on.

I recently did some very basic performance measurements by timing a task that I completed with *cleartool* and then timing the same task using a CAL

implementation instead. For this specific operation, CAL was about 30 percent faster. While these results are empirical, and fairly rough, they show that CAL can offer a significant advantage when speed is important.

Useful Tools

Of course, CAL itself and the CAL documentation are essential for developing Perl scripts that access ClearCase data (without using *cleartool*). If you've installed Rational ClearCase, then you already have CAL and the documentation on your system.

Documentation for CAL includes conceptual material, diagrams, several examples, and reference pages for each interface in CAL. There are several other tools -- some essential, some merely handy -- that I find valuable in Perl/CAL development.

As I mentioned earlier, CAL is a standard COM interface, which means that it will work with any tool that supports COM. This includes some, but not all, versions of Perl. The version of Perl that comes with ClearCase, *ccperl*, does not support COM. If you want to use CAL with Perl, you will need to use another Perl distribution. I've found ActivePerl from ActiveState³ to be a great tool. It is a nice, robust implementation, and best of all it is free. Plus, the ActivePerl distribution includes a Perl package manager, which automatically finds and installs Perl packages. This is helpful for COM developers because you have to install a COM package in order to use COM in Perl.

The inexpensive Perl Development Kit, also available from ActiveState, is another extremely useful tool. If you plan to do Perl development in any serious sense, it is probably one of the best investments you can make. The two components I have used most are the Perl debugger -- which allows you to step through code, set breakpoints and watch variables -- and PerlApp, which allows you to convert a Perl application into a completely self-contained Windows program that can be run on any system, even if the system does not have a Perl interpreter.

The second edition of the *Perl CD Bookshelf* from O'Reilly includes searchable online copies of the following Perl references: *Perl in a Nutshell*, *Programming Perl*, 3rd Ed., *Perl Cookbook*, *Advanced Perl Programming*, and *Perl for System Administration*. This resource includes source code that you can cut-and-paste, which is an exceptionally common (and convenient) practice in Perl Programming. Legend has it that only one Perl program was ever written -- all the rest were derived from it. I take a copy of this CD everywhere I go, because it is just too useful to be without.

Some other tools and resources that are useful (but not required) in this context include:

- **Rational SoDA.** Rational SoDA is an automated report generator that has the ability to pull together static data from a variety of Rational tools. For example, it can reach into Rational ClearCase and Rational ClearQuest repositories, and build a report based on a template that you define. However, Rational SoDA cannot perform interactive queries and then take action based on the results -- something a Perl script can do

with ease.

- **Rational ClearQuest API and Documentation.** Like Rational ClearCase, Rational ClearQuest also offers a COM library that you can use to access its data. The Rational ClearQuest API is used by external programs to view or modify the data that Rational ClearQuest stores in the user database and schema repository. The sample Perl application we'll look at in this article, `changeset_report.pl`, uses the Rational ClearQuest API to query the Rational ClearQuest database.
- **Visual Basic.** I use Visual Basic to quickly develop prototypes and test programs. Visual Basic has extensive COM support and a built-in IDE (Integrated Development Environment), which makes it handy for troubleshooting any problems you might have with COM interfaces, including CAL. Visual Basic's object browser is also helpful for examining COM interfaces in an easy-to-understand format.

A Sample Project

Recently, while working with a customer, I came across a situation ideally suited to a CAL-based solution written in Perl. To understand this customer's particular needs, a little background will be helpful. Rational ClearCase and Rational ClearQuest store data in different data domains. Each tool provides extensive reporting capabilities within its respective data domain. Rational ClearCase uses *cleartool* subcommands such as `cleartool find` and `cleartool describe` for locating and reporting on ClearCase-controlled data, and more recent versions of ClearCase include ClearCase Report Builder, which uses a library of report programs to support dozens of standard report options. Rational ClearQuest uses Crystal Reports. There is, of course, integration between the two tools that allows them to share data. For example, if you enable UCM with Rational ClearQuest and Rational ClearCase, a link is made between change requests and activities, so that when you access an activity in Rational ClearCase, Rational ClearQuest gives you information about the change request or defect associated with that activity.

Rational ClearQuest maintains information about the process, the state of the activity, and so on; Rational ClearCase maintains the activity objects -- the actual change set information. In UCM, that information always stays in the Rational ClearCase database to avoid replicating information unnecessarily⁴. Now, let's say you were doing a code review, and you wanted a report that showed all the open and active defects in Rational ClearQuest, including the state of the defect, the owner, etc. -- and you also wanted to include the change set to show what code needed to be reviewed.

In fact, this is exactly what the customer wanted to do. The solution, as you have probably guessed, was to write a simple Perl script that uses CAL and the Rational ClearQuest API to create a report with just the change set information that they needed.

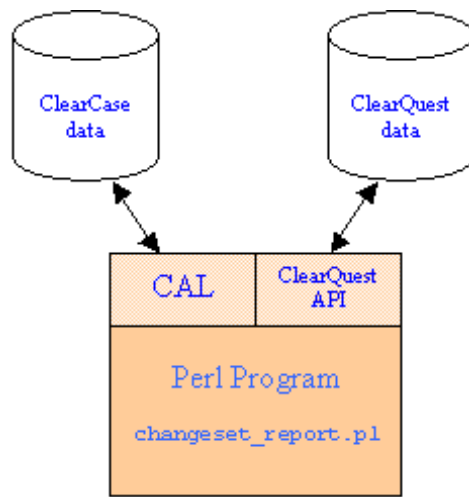


Figure 1: The Perl Application `changeset_report.pl` Uses CAL and the Rational ClearQuest API

The Perl application, `changeset_report.pl`, queries the Rational ClearQuest database via the Rational ClearQuest API to obtain a list of ClearQuest records. For each record, the program gets the Rational ClearCase UCM activity, and uses CAL to extract the change set information for that activity. The program then prints a report based on the data it finds.

Now let's look at the commented [source code](#) for `changeset_report.pl`. As we discuss the highlights and most important CAL-related aspects, you'll find it helpful to refer to the full code to better see how the pieces fit together. Because the application was actually developed for use in the real world, there are some added features that make it easier to use, but that you would not necessarily find in an example application focused solely on using CAL. For example, although all of the `changeset_report.pl` options and parameters can be set via the command line switches, there is also a Tk⁵-based graphical user interface (GUI) that prompts the user for any missing information (see Figure 2).

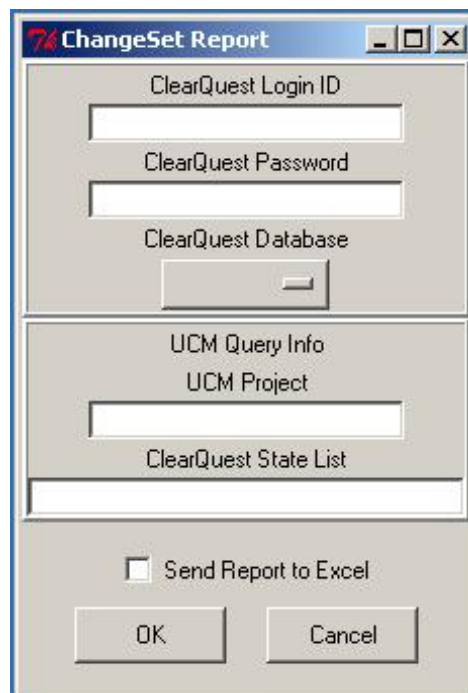


Figure 2: The Tk-Based Front End of `changeset_report.pl`

The sample program also includes an option for sending the output to an Excel spreadsheet instead of printing it out. Finally, although use of the ClearQuest API is an integral part of this sample application, we'll focus primarily on the sections related to CAL. In particular, the steps used to access Rational ClearQuest -- including creating a ClearQuest session, building the query, and getting results -- are not explained in detail here.

A Walk Through the Code

The first important line to note in the script is the first non-comment line:

```
use Win32::OLE;
```

Essentially, this line includes the `Win32::OLE` package so that it can be used in our script.

After the program has processed the command-line switches (or the input from the GUI), and builds a list of ClearQuest records, it enters a loop that is repeated for every ClearQuest record returned. In that loop, which appears considerably later in the script, CAL is instantiated for any ClearQuest record that contains a `ucm_vob_object` field. In UCM, the VOB object identifies the change set associated with a particular activity. If the ClearQuest results include an identifier for this object, then the script uses `Win32::OLE` to create a new ClearCase Application object, as you can see below.

```
#-----  
# If the CC activity ID is not null, then we need to use CAL  
#   to extract the changeset info from the CC activity object  
#-----  
  
print "Instantiating CAL\n" if $DEBUG;  
my ($CCApp) = Win32::OLE->new("ClearCase.Application") or  
    die "Can't create ClearCase application object via  
        call to Win32::OLE->new(): $!";
```

The two entry points for interacting with CAL are the *Application object* and the *ClearTool object*. Most applications -- including this sample application -- use the Application object, as it exposes most of the CAL API. The ClearTool object contains a single method, `CmdExec`, which can be used to execute a *cleartool* subcommand string. This ensures that any capabilities in *cleartool* not covered by the Application object are accessible programmatically. The code above instantiates a ClearCase Application object named `$CCApp`. The `my` operator simply limits the scope of the `$CCApp` variable. If the Application object cannot be created, then the script terminates or "dies" with an appropriate error message. The "`print...$DEBUG`" line in the code above and throughout the script serves as a rudimentary -- but very common -- debugging mechanism. If you don't want to use the Perl debugger, you can enable this verbose output by simply un-commenting the `$DEBUG = 1` line located near the start of the script.

Once we have an Application object, we use the `Activity()` method to obtain an activity object from the value of the `ucm_vob_object` field that we obtained from

Rational ClearQuest, as shown below.

```
#-----  
# Get an activity object from CAL  
#-----  
  
$myactivity = $CCApp->Activity($activity_id);
```

The result is an activity object, which I assign to `$myactivity`. To resolve the names of objects in an activity's change set, Rational ClearCase requires a view context. You can have ClearCase identify the best view heuristically, using the `NameResolverView` property of the activity. This view can, in turn, be used to resolve names of objects in the activity change set:

```
$view = $myactivity->NameResolverView;
```

To get the actual change set, we call the `ChangeSet()` method of the activity object, which returns a Collection object containing change set entries or `CCVersions`:

```
#-----  
# Get the activity's change set, which is a CCVersions collection  
#   Use the activity's "nameresolver view" for name resolution.  
#-----  
  
$ChangeSet = $myactivity->ChangeSet($view, "False");  
$CS_Entries = $ChangeSet->Count;
```

At this point, the script enumerates the change set entries and accesses the path name of each changed source file via the `ExtendedPath` property.

```
#-----  
# Loop through the CCVersions collection, collecting the names of  
#   the versions for printing.  
#-----  
  
print "Getting ChangeSet info\n" if $DEBUG;  
  $CS_Index = 1;  
  while ($CS_Index <= $CS_Entries) {  
    .  
    .  
    .  
    $Version = $ChangeSet->Item($CS_Index);  
    $VersionPN = $Version->ExtendedPath;  
    .  
    .  
    .  
    $CS_Index++;  
  }
```

After iterating over each element of the change set collection, the script prints the results (or sends them to Microsoft Excel), and then repeats the process on the next ClearQuest record until there are no more. Sample output from `changeset_report.pl` is shown in Figures 3A and 3B.

```

Result set of query for record type All_UCM_Activities
ID           State      Owner      Headline
-----
RUC00000042  Opened    tomm       Fix Order Stuff
Change Set Info:
M:\tomm_Rel13\Classics\Ops\OPSInterface.cpp@@\main\tomm_Rel13\2
M:\tomm_Rel13\Classics\Ops\OPSInterface.h@@\main\tomm_Rel13\1
M:\tomm_Rel13\Classics\Ops\OPSInterface.cpp@@\main\tomm_Rel13\1
RUC00000043  Opened    tomm       Fix Demo Notation
Change Set Info:
M:\tomm_Rel13\Classics\Ops\OPSDemo.cpp@@\main\tomm_Rel13\1

2 activities were found
Press any key to exit

```

Figure 3A: Sample Standard Output from changeset_report.pl

	A	B	C	D	E
1	ID	State	Owner	Headline	Change Set
2	RUC00000042	Opened	tomm	Fix Order Stuff	M:\tomm_Rel13\Classics\Ops\OPSInterface.cpp@@\main\tomm_Rel13\2 M:\tomm_Rel13\Classics\Ops\OPSInterface.h@@\main\tomm_Rel13\1 M:\tomm_Rel13\Classics\Ops\OPSInterface.cpp@@\main\tomm_Rel13\1
3	RUC00000043	Opened	tomm	Fix Demo Notation	M:\tomm_Rel13\Classics\Ops\OPSDemo.cpp@@\main\tomm_Rel13\1

Figure 3B: Sample Microsoft Excel Output from changeset_report.pl

What's Next?

The `changeset_report.pl` script is not exceptionally complicated. In fact, it uses only a small subset of the capabilities offered by CAL. The CAL documentation includes a comprehensive list of all the CAL interfaces, as well as additional sample applications (including a couple more written in Perl). So, while `changeset_report.pl` illustrates only a fraction of what you can do with CAL, it does show how useful and practical CAL can be with a minimum of effort. With this small Perl program, we can see how easy it is to use CAL and the Rational ClearQuest API to access Rational ClearCase and ClearQuest data programmatically.

In fact, there are a myriad of different uses for CAL; and you've probably already thought of a few yourself. One possible use that I have started to play with is a kind of "release manager" application -- a utility that would present the user with a choice of baselines from which to construct a particular configuration and generate release notes based on the selected baseline. Although I have not finished it yet, working on it has reminded me how valuable CAL is, and how many possible applications it has. Once you start using it, I'm sure you'll find even more.

Notes

¹ Larry Wall, Tom Christiansen, and Jon Orwant, *Programming Perl*, 3rd Edition. O'Reilly & Associates, 2000.

² For example, *Essential COM* by Don Box (Addison-Wesley, 1998).

³ The author has no relationship with Active State. ActivePerl is available at www.activestate.com.

⁴ Note: In the non-UCM integration between Rational ClearCase and Rational ClearQuest, change set data is replicated into the ClearQuest database, so the sample application is not needed to generate such a report.

⁵ Tk is an exceptionally helpful package used to create graphical user interfaces. For more information on using Tk in Perl, see *Learning Perl/Tk* by Nancy Walsh, O'Reilly & Associates, 1999, or the *Perl/Tk Pocket Reference* by Stephen Lidie, also from O'Reilly & Associates (1998).

References

The Perl CD Bookshelf, 2nd Edition. O'Reilly & Associates, 2001.

Randal L. Schwartz and Tom Phoenix, *Learning Perl*, 3rd Edition. O'Reilly & Associates, 2001.

Dave Roth, *Win32 Perl Programming: The Standard Extensions*. Macmillan Technical Publishing, 1998.

Larry Wall, Tom Christiansen, and Jon Orwant, *Programming Perl*, 3rd Edition. O'Reilly & Associates, 2000.

Don Box, *Essential COM*. Addison-Wesley, 1998.

Nancy Walsh, *Learning Perl/Tk*. O'Reilly & Associates, 1999.

Stephen Lidie, *Perl/Tk Pocket Reference*. O'Reilly & Associates, 1998.



For more information on the products or services discussed in this article, please click [here](#) and follow the instructions provided. Thank you!