ClearCase: The ten best scripts

developerWorks
In this article:
- Introduction
- General Considerations
- Cleartool find . -all script
- Scripts to recursively checkin / checkout / add to source control
- Scripts to Remove Old Files
- Other Administrative Scripts
- Copy Merge script
- Compare label script
- Conclusion:
- Downloads
- About the author
- Rate this page

Related links
- Rational technical library
- The Rational Edge

Level: Introductory

Daniel Diebolt, Rational tools consultant, IBM

18 Jun 2004
Updated 25 Jul 2006

In the first part of this two-part article, Daniel Diebolt, a Rational Senior Technical Representative, describes some useful scripts for the daily use and implementation of ClearCase.

Introduction

IBM® Rational ClearCase® is a Software Configuration Management tool that benefits both small and large software development teams. The complexity of the tool, and the fact that individual developers use it in different ways, can make the life of a Software Configuration Manager quite difficult.

Over the last few years I've performed many implementations and presentations with ClearCase. I've observed that even though every project is unique, and each organization has its own approach to solving problems, the same questions about ClearCase usage always come up. Since I didn't want to reinvent the wheel for every project, I collected some of the most useful scripts that I've used during implementations. I made these as generic as possible to increase their applicability across projects. The practice has helped me immensely: I now have a collection of scripts that I can easily use to answer recurring questions about ClearCase usage. This document describes the ten most useful scripts, which should help make the daily work of a Software Configuration Manager much less complicated.

Usage

A word of caution: All the scripts described in this document are delivered "as is." *There is no official support offered by IBM Rational Software for these scripts.*

All my scripts were developed in one of three languages: either directly as shell / batch, in Perl, or using the ClearCase API (CAL). I am not a Perl expert, and you will immediately see that my knowledge of Perl language is limited, but hopefully these scripts are easy to understand. Nor am I a Visual Basic expert, and after looking over my VB programs you may get the feeling that they could be rewritten in a more optimized way. In any case, feel free to contact me with either your comments or suggestions for improvements that you would like to see in these scripts.

The Ten Best Scripts

In this paper, we will see ten of the best scripts I have come across in my daily work as a Rational Consultant. I did not personally write all of these scripts. Some of them where written by my Rational TechRep colleagues around the world, so I want to thank all of them for their great contributions and hard work.

Special thanks to:

- Arjan Ten Hoopen / Senior Technology Consultant / The Netherlands
- Eric Ostrander / Senior Software Engineer Specialist / Arizona
- Michel Lohr / Senior Technology Consultant / The Netherlands
- Benoit Lorendeaux / Technology Representative / Switzerland

Back to top

General Considerations

Before writing a script, I always ask myself: what functionality would the user like to achieve? Do they want an improvement in their daily work processes, or just the ability to spend less time on some basic tasks? After answering these first questions, the next ones I ask myself are about which "language" to use. For this, I have three approaches:

- Simple -- I will use a **cleartool** command, with some redirection / -exec behavior, to solve the need.
- Middle -- When there is a need to perform more complex analysis and processing of ClearCase commands, I use Perl scripts.
- Complex -- When interaction with the user is necessary, and multiple pieces of information need to be collected

before the execution of the script, I often choose to write a Visual Basic application with the ClearCase API, or CAL (ClearCase Automation Layer).

While writing scripts or ClearCase commands, I constantly remind myself that the cleartool command line is my friend; it works the same on UNIX and Windows. Learning all the commands, options, and functionalities has made my life a lot easier.

My second best friend while writing scripts is the **cleartool man** command, which invokes the ClearCase reference manual. There is no better place to get a description of all the command options, and there are always some examples of how commands are used available at the end of the manual pages.

I use many cleartool commands infrequently, but there are a few of them I use over and over again, such as **cleartool find**, **cleartool describe**, and **cleartool lstype**.

What is also very useful about cleartool commands is that some of them have an -**fmt** option that formats strings for command output, which is a good way to extract the information you need without many "grep" or "sed" or "awk" statements. For more details about the -**fmt** option, run a `cleartool man fmt_ccase` command.

As you know, most Windows users do not like to use the old MSDOS command shell. They prefer to have nice GUIs, displaying their work with lots of colors and icons, and for this reason the **Clearprompt** command is another big friend of mine: it can ask for simple information via the GUI (Windows Explorer or ClearCase Details).

Back to top

Cleartool find . -all script

The simplest script that you can write with ClearCase is actually just using the cleartool **find** command. This command has an **- exec** option, giving you the ability to invoke an action based on the search results.

However, when I am writing a find request I use an iterative process, to ensure that I write the correct command. Since I know that the complexity of this command is pretty high, I prefer to execute steps sequentially, and make improvements after each step, before reaching my final command.

Let's use an example (from the cleartool manual) about the **cleartool find** command. As you can see, I modify the query slowly, to refine my search as I get more information:

- I begin by using an `-exec "cmd /c echo %CLEARCASE_PN%"` as my exec statement. Why? Simply because with this echo, I can see the results without affecting the rest of the command. Also, I use a **cmd /c** statement so that afterwards I can modify the echo to any other command.
- I then use `-exec "cmd /c echo cleartool` *command* `%CLEARCASE_PN%"`. Why not directly execute the cleartool command, after the **-exec** statement? Because if anything goes wrong, it is too late. By echoing the command, I can redirect my **cleartool find** command to a specific file, and then analyze and make any late changes to that file.

Finally, we are using the **-all** option because by default, the **find** command only considers the elements that you can actually "see" in your view. With the **-all** option, I ensure that I am considering every element in the VOB, whether I can see it or not.

Cleartool find . -all examples

Here are some examples of **cleartool find . -all** commands:

- To change the ownership of all elements, starting from the directory where you execute the command, use `cleartool find . -all -exec "cmd /c cleartool protect -chown ddiebolt %CLEARCASE_PN%"`
  - We us this instead of **cleartool protect -recurse** because the **-recurse** only considers the elements that you actually "see" in your view **config_spec**.
  - With this **cleartool find** command, you can be sure that you actually change the ownership of *all* the elements.
- To create a branch named "ddiebolt_branch" on all the elements automatically, starting the branch from the "REL1" label, use `cleartool find . -all -version "lbtype(REL1)" -exec "cmd /c cleartool mkbranch -nc ddiebolt_branch % CLEARCASE_XPN"`
  - This is superior to a **-mkbranch** in the **config_spec** because the **-mkbranch** will create the branch only for those elements that are checked out.
  - This **cleartool find** command creates a branch for all the elements, not just the ones that you are modifying.
- Once I was asked if there is a way to *find all the versions in a VOB*. After trying several approaches, I determined that the solution was actually to search all elements that have a branch **main**, which of course is always true. Therefore, the command to do this is `cleartool find . -all -version "brtype(main)" -exec "cmd /c echo % CLEARCASE_XPN%"`
- I've also been asked if there is a way to get all the latest versions, and I tried the following:
  - `cleartool find . -all -version "lbtype(LATEST)" -exec "cmd /c echo %CLEARCASE_XPN%"` Unfortunately, this does not work.

Back to top

Scripts to recursively checkin / checkout / add to source control

NOTE: This script is intended for use with base ClearCase and not UCM.

One of the first questions new ClearCase users ask is: how is it possible to check out or check in all the elements behind a directory recursively, using the ClearCase / Windows Explorer integration? I find myself asking why they would like such functionality. Typically, they have used tools like RCS or CVS, where you need to get a copy of the files to be able to access

them (that is, you need to check out all the files). However, there are times when you need to check out or check in everything because your favorite IDE does not accept files that have the "read only" attributes. Here are my solutions:

Script to recursively Check Out and Check In

Note: For the following procedures, you need to have local administration rights to be able to customize a ClearCase menu.

Checkout

For the *check out recursively* option, do the following:

1. Start the `clearmenuadmin.exe` utility (to do this, click **Start > Run**, and type `clearmenuadmin.exe`)
2. In the **Object type** tab, select the **directory** object. Choose **checked-in** as the **Object state**
3. Click the **new** button in the **Available menu Choices** pane, and then enter the following information in the appropriate fields:

- **Menu Text:** Checkout (recursively)...
- **Help Text:** Check out the selected item recursively...
- **Command Type:** Executable/Regentry
- **Command:**
  `Software\Atria\ClearCase\CurrentVersion\ContextMenus\CmdLineExe`
- **Initial Directory:**
  Arguments: `/c cleartool find $file -exec "cmd /c cleartool checkout -nc \"%CLEARCASE_PN%\""`
- **Comment:**

When this is done, you can add this new menu entry to the **This menu** contents pane by pressing the **Add** button. Position it as desired with the **Move up** button. After completing this customization, choose **Configuration>Apply** to apply the changes. Then you can test it in Windows Explorer.

Checkin

For the *check in recursively* option, do the following:

1. Start the `clearmenuadmin.exe` utility
2. In the **Object type** tab, select the **directory** object. Choose **checked-in** as the **Object state**
3. Click the **new** button in the **Available menu Choices** pane, and then enter the following information in the appropriate fields:

- **Menu Text:** Checkin (recursively)...
- **Help Text:** Check in the selected item recursively...
- **Command Type:** Executable/Regentry
- **Command:**
  `Software\Atria\ClearCase\CurrentVersion\ContextMenus\CmdLineExe`
- **Initial Directory:**
  Arguments: `/c cleartool find $file -exec "cmd /c cleartool checkin -nc -identical \"%CLEARCASE_PN%\""`
- **Comment:**

When this is done, you can add this new menu entry to the menu contents pane, position it as desired, and apply the changes as described in the previous procedure, and then test it in Windows Explorer.

Script to recursively Add to source control

These two recursive checkout or checkin script / customizations are great, but one customer gave me this additional challenge: *also do a recursive add to source control*.

For me, a recursive add to source control would mean executing **clearexport_ffile** and **clearimport** over the command line as the fastest method. Again though, this was a challenge, so I developed this small script to do the job.

Add-to-src-control.bat

These are the contents of the `add-to-src-control.bat` script:

```
@rem= 'PERL for Windows NT -- ccperl must be in search path
@echo off
ccperl %0 %1 %2 %3 %4 %5 %6 %7 %8 %9
goto endofperl
@rem ';

###############################################
# Begin of Perl section


$start_dir = $ARGV[0];

#
# Fixed variable
#
$S = "\\";
$list_file = "c:".$S."list_file";
$list_add = "c:".$S."list_add";
```

```
$choosed = "c:".$S."choosed";


sub clean_file
{
$status = system("del $list_file > NUL 2> NUL");
$status = system("del $list_add > NUL 2> NUL");
$status = system("del $choosed > NUL 2> NUL");
}

#
# Start of the script...
#

printf("add-to-src-control $start_dir...\n");

clean_file();
$status = system("cleartool ls -view_only -r -s $start_dir > $list_file");
open(LIST_ELEMENT,$list_file);
while ($element=<LIST_ELEMENT>)
{
chop $element;
# printf " Processing $element ";
if ($element =~ /CHECKEDOUT/)
{
# printf(" checkedout file \n");
}
else
{
# printf " view private \n";
printf " Processing $element ...\n";

#
# For files with spaces...
#
if ($element =~ / /)
{
$status = system("cmd /c echo \"$element\" >> $list_add");
}
else
{
$status = system("cmd /c echo $element >> $list_add");
}
}
}
close(LIST_ELEMENT);

if (-e $list_add)
{
$listelement = `type $list_add`;
$listelement =~ s/\n/,/g;
$status = `echo $listelement > $list_add`;

$status = system("clearprompt list -outfile $choosed -dfile $list_add -choices
-prompt \"Choose element(s) to put over version control : \" -prefer_gui");

if ($status != 0)
{
# printf("\n Aborting ...\n");
clean_file();
exit $status;
}

#
$listtoadd = `type $choosed`;
$listtoadd =~ s/\n//g;
printf("\n cleardlg /addtosrc $listtoadd");
$status = system("cleardlg /addtosrc $listtoadd");

clean_file();
exit $status;
}
else
{
```

```
# printf("\n No files founded...\n");
clean_file();
exit $status;
}

# End of Perl section

__END__
:endofperl
```

Installation

Start by putting the `add-to-src-control.bat` file in your normal script directory, then customize the ClearCase Explorer Menu integration by doing the following:

1. Start the **clearmenuadmin.exe** utility
2. On the **Object type** tab, select the **directory** object. Choose **checked-in** as the Object state.
3. Click the **New** button in the **Available menu choices** pane , and then enter the following information in the appropriate fields:

- **Menu Text:**
  Add to source control (recursively)...
- **Help Text:**
  Add to source control (recursively)...
- **Command Type:**
- **Command:**
  `<DirWhereTheScriptIs>\add-to-src-control.bat`
- **Initial Directory:**
  $dir_or_file1
- **Arguments:**
  $file
- **Comment:**

When this is done, you can add this new menu entry to the menu contents pane, position it as desired, and apply the changes as described in the previous procedure.

Also add the menu entries for:

- **Directory / Checked-out**
- **VOB / Checked-in**
- **VOB / Checked-out**

After completing this customization, choose **Configuration > Apply** to apply the changes. Then you can test it in Windows Explorer.

Usage

In the ClearCase-Windows Explorer integration, you will then have a menu entry called **Add to source control (recursively)**

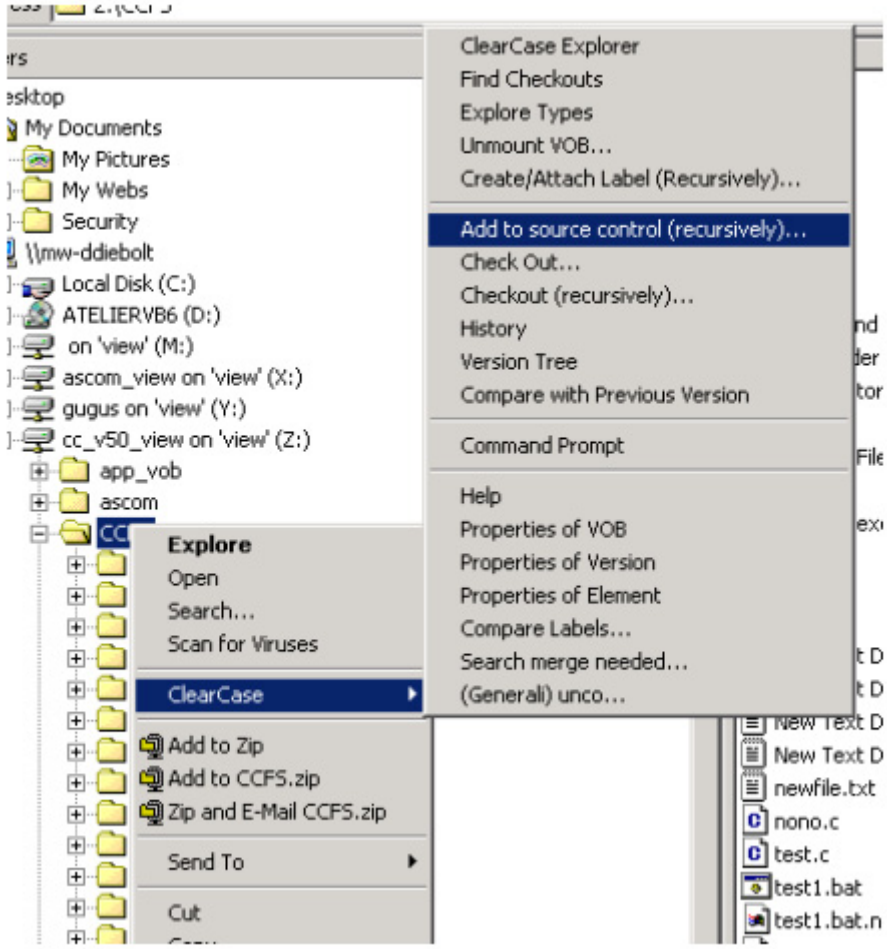


Figure 1: New menu item added by the script

When you select this item, you will see the following output window:

Figure 2: Result of choosing the **ClearCase > Add to source control (recursively)** subcommand

In this window, you can then select one or more files to put under version control:

- pressing **Yes** will put the selected item(s) under version control.
- pressing **Abort** will cancel the task.

In the background, the following window will be displayed:

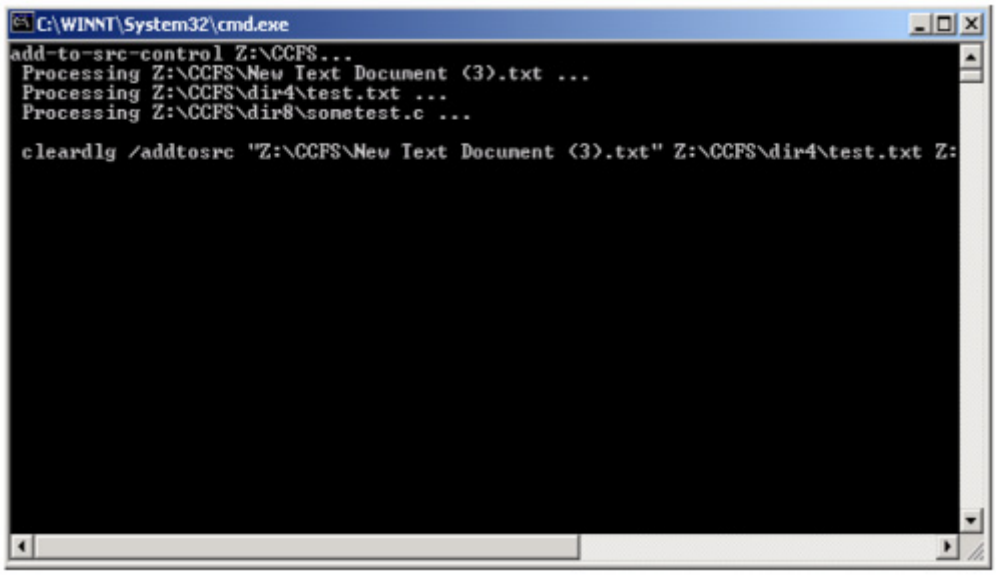


Figure 3: Background DOS display

At this point, you can insert comments about adding the files to version control:

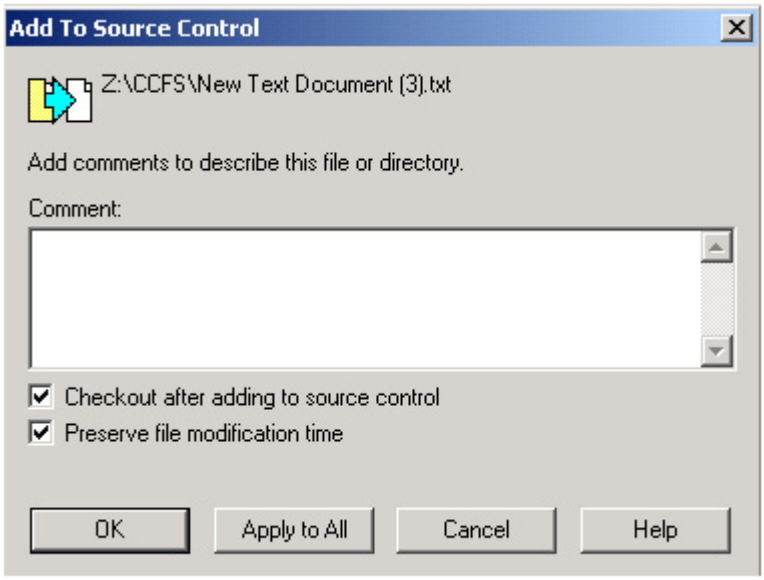


Figure 4: Comments window

Problems not solved

The trigger script does not work with empty files. If the file you are trying to put under version control is empty, then the following error message is displayed:

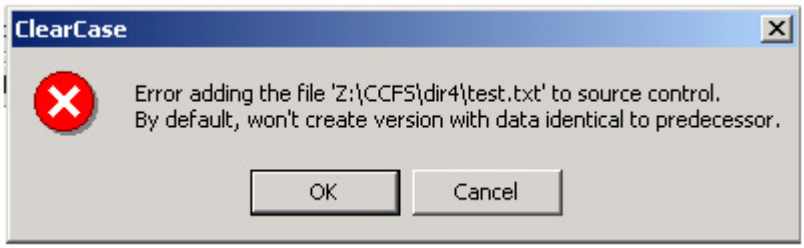


Figure 5: Empty file error message

Back to top

Scripts to Remove Old Files

Remove old view script

[Many thanks Eric J. Ostrander and Schindler Lift AG]

After using ClearCase for a few years or even months, thousands of views accumulate, and nobody has a clue what they are for. Cleaning up unused views is one of the main tasks facing the ClearCase administrator. But which views do you delete? How do you inform the user that his view will be destroyed?

Eric Ostrander once wrote some scripts to delete any view that was not used for 30 days. I delivered them to my customer Schindler Lift AG, but they felt that just "removing old views" without any notice was not correct; they wanted to send an e-mail to the view's owner first. So I rewrote the script to send an e-mail to the view owner, via the **notify.exe** utility.

Search_view.bat

Here is the script:

```
@rem= 'PERL for Windows NT -- ccperl must be in search path
@echo off
ccperl %0 %1 %2 %3 %4 %5 %6 %7 %8 %9
goto endofperl
@rem ';


#
# Views last accessed since 30 days...
#

$days = 30;

#
# ----------------------------
#

if ($ENV{TEMP})
{
$TDIR=$ENV{TEMP};
}
else
{
$TDIR="c:${S}temp";
}

$S = "\\";
$JUNK = $TDIR . $S . "junk";
$NULL = $TDIR . $S . "null";


# Begin of Perl section

printf("Start of the script...\n");


if ($ARGV[0] eq "")
{
printf "\n No user specify, please give one !!! \n";
exit 1;
}

$search_user = $ARGV[0];


printf("Searching for all views that hasn't been accessed since $days days for
$search_user \n");
chomp(@views = `cleartool lsview -s | findstr $search_user`);



foreach $viewtag (@views)
{
printf("Processing $viewtag: ");

# Determine the last accessed date and owner for the view.
```

```
chomp(@properties = `cleartool lsview -properties -full $viewtag`);
foreach $property (@properties)
{
$accessed_line = $property if ( $property =~ /^Last accessed / );
$owner_line = $property if ( $property =~ /^Owner: / );
}

$last_accessed = (split(/\./,(split(/ +/,$accessed_line))[2]))[0];
($view_day,$view_month,$view_year) = split(/-/,$last_accessed);
$owner = (split(/\\/,(split(/ +/,$owner_line))[1]))[1];

# Build the view's date string for use with the since.pl script.
if ( $view_year > 80 )
{
$view_year = "19$view_year";
}
else
{
$view_year = "20$view_year";
}

$view_date = "$view_month $view_day $view_year";

# Find out how long ago the view was last accessed. If greater
# than the specified number of days, print out the data.
$diff_days = `ccperl .\\since.pl $view_date $now_date`;
if ( $diff_days > $days )
{
printf("$owner $last_accessed\n");
printf("\t Sending notification for view $viewtag to $owner \n");
system("cmd /c del $JUNK 2> $NULL");
system("echo .
> $JUNK");
system("echo Hello, $owner...
>> $JUNK");
system("echo .
>> $JUNK");
system("echo Your view $viewtag hasnt been used since $days days !!!
>> $JUNK");
system("echo .
>> $JUNK");
system("echo Please check if you still need this view... >>
$JUNK");
system("echo .
>> $JUNK");
system("echo If not, please delete it !
>> $JUNK");
system("echo .
>> $JUNK");
system("echo If not done/used in the next 30 days, it will be
>> $JUNK");
system("echo automatically deleted without notice >>
$JUNK");
system("echo .
>> $JUNK");
system("echo S. Aeschbacher and D. Diebolt
>> $JUNK");
system("echo .
>> $JUNK");

system("notify -s \"View $viewtag...\" -f $JUNK $owner\@rational.com");
system("notify -s \"View $viewtag...\" -f $JUNK ddiebolt\@rational.com");

push(@old_views,"$viewtag");
}
else
{
printf("\n");
}
}

printf("End of the script...\n");
```

```
# End of Perl section

__END__
:endofperl
```

This script uses the **since.pl** module (following).

Since.pl

```
#!/usr/local/bin/perl

# Determine the number of days since a certain date
# or between two dates.
# It will return a -1 if there is something wrong with
# your input values.

# 1.0 11/23/1998 Eric J. Ostrander Created.
# 1.1 09/19/1999 Eric J. Ostrander Modified to calculate days between any two
# arbitrary dates.

# Predefinitions.
@months = split(/ +/,"Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec");
@mlength = split(/ +/,"31 28 31 30 31 30 31 31 30 31 30 31 ");

# Are there the correct number of arguments?
if ( scalar(@ARGV) != 3 && scalar(@ARGV) != 6 ) {
print "Usage: since month day year [ month day year ]\n";
print " since from-date to-date\n";
print " ex: since Apr 15 1962 Sep 30 1963\n";
print "If to-date is omitted, today's date is assumed.\n";
exit;
}

# What is the from-date?
$req_month = $ARGV[0];
$req_day = $ARGV[1];
$req_year = $ARGV[2];

# What is the to-date?
if ( scalar(@ARGV) == 6 ) {
$month = $ARGV[3];
$day = $ARGV[4];
$year = $ARGV[5];
} else {
($x,$month,$day,$x,$year) = split(/ /,localtime(time));
}

# Is the requested date valid?
$bad = 1;
$i = 0;
foreach $mon (<@months> ) {
if ( "$req_month" eq "$mon" ) { $req_ind = $i; $bad = 0 }
if ( "$month" eq "$mon" ) { $mon_ind = $i }
$i = $i + 1;
}
if ( $req_day > $mlength[$req_ind] || $req_year > $year ) { $bad = 1; }
if ( $req_year == $year && $req_ind >= $mon_ind && $req_day > $day ) { $bad = 1; }
if ( $bad == 1 ) { print "-1\n"; exit; }

# Calculate days from Jan. 1, $req_year to today's date.
CALC_DAYS();
$to_days = $days;

# Calculate days from Jan. 1, $req_year to requested date.
$month = $req_month;
$day = $req_day;
$year = $req_year;
CALC_DAYS();
$from_days = $days;

# Subtract the two.
print $to_days - $from_days . "\n";

# The iteration subroutine.
```

```
sub CALC_DAYS {
$yr = $req_year;
$days = 0;
ITERATION: while (1) {
$i = 0;
$lpyr = 0;
if ( $yr/4 == int($yr/4) && $yr != 2000 ) { $lpyr = 1; }
foreach $mon (<@months>) {
if ( "$mon" eq "$month" && $yr == $year ) {
$days = $days + $day;
last ITERATION;
} else {
$days = $days + @mlength[$i];
if ( "$mon" eq "Feb" ) { $days = $days + $lpyr; }
}
$i = $i + 1;
}
$yr = $yr + 1;
}
}

exit;
```

Usage

Simply start the `search_view.bat` file with a username as a parameter.

Example:

Enter `Z:\cc_tools\batch\RmOldView>search_views.bat ddiebolt` and the script starts:

```
Searching for all views that haven't been accessed for 30 days
for ddiebolt

Processing ddiebolt_UCM_Release_3.0_integration:

Processing ddiebolt_test: ddiebolt 12-Jun-02

Sending notification for view ddiebolt_test to ddiebolt
```

Remove view private file script

Users create view *private* files over and over again, filling up the view-storage directory with useless files that you (the administrator) must now clean up. You may also simply want to clean up your views, removing old objects, executables, and so on. Removing the whole view and recreating a new one is just not the solution. Saving the view private files that you want to keep to another location, and then copying them back after creating the new view, is also not the answer. In place of these cumbersome solutions, I've created an elegant script, `rmviewprivate.bat`, which selects the files to delete and performs the task of cleaning up your views.

Rmviewprivate.bat

Here is the script:

```
@rem= 'PERL for Windows NT -- ccperl must be in search path
@echo off
ccperl %0 %1 %2 %3 %4 %5 %6 %7 %8 %9
goto endofperl
@rem ';

if ($ENV{TEMP})
{
$TDIR=$ENV{TEMP};
}
else
{
$TDIR="c:${S}temp";
}

$S = "\\";
$LIST_VIEW_ONLY = $TDIR . $S . "list_view_only";
$LIST_SELECTED = $TDIR . $S . "list_selected";



sub do_del_file
```

```
{
local($file_to_delete) = @_[0];
if (-e $file_to_delete)
{
printf("del \"$file_to_delete\" \n");
$return = system("cmd /c del \"$file_to_delete\"");
}
}

sub do_del_directory
{
local($dir_to_delete) = @_[0];
printf("rmdir \/q \/s \"$dir_to_delete\" \n");
$return = system("cmd /c rmdir \/q \/s \"$dir_to_delete\"");
}

# Begin of Perl section


printf("\n Start of the script \n\n");

if ($ARGV[0] eq "")
{
printf("\n No directory specified, will use .\n");
$start_dir = ".";
}
else
{
$start_dir = $ARGV[0];
}

if (!(-d $start_dir))
{
printf("\n $start_dir is not a directory !!! \n");
exit 1;
}


system("cmd /c del $LIST_VIEW_ONLY 2> NUL");
printf("\n cleartool ls -recurse -view_only $start_dir \n");
chomp(@list_files = `cleartool ls -recurse -view_only $start_dir`);
foreach $file (@list_files)
{
if ( $file =~ /CHECKEDOUT/ )
{
$is_checkedout = $file;
}
else
{
$is_checkedout = "";
}
# printf("\n Processing $file : $is_checkedout \n");
if ("$is_checkedout" eq "")
{
system("cmd /c echo $file, >> $LIST_VIEW_ONLY");
}
}

if (-e $LIST_VIEW_ONLY)
{
$return = system("clearprompt list -outfile $LIST_SELECTED -dfile
$LIST_VIEW_ONLY -choices -prompt \"Please choose files \/ directory to delete\"");
}
else
{
$return = 512;
}

if ("$return" eq "0")
{
open(LIST_ELEMENT,$LIST_SELECTED);
while ($element=<LIST_ELEMENT>)
{
chop $element;
```

```
if (!("$element" eq ""))
{
if (-d $element)
{
#printf(" $element is a directory\n");
do_del_directory($element);
}
else
{
#printf(" $element is a file\n");
do_del_file($element);
}
}
}
close(LIST_ELEMENT);
}

system("cmd /c del $LIST_VIEW_ONLY 2> NUL");
system("cmd /c del $LIST_SELECTED 2> NUL");


printf("\n End of the script \n");


# End of Perl section

__END__
:endofperl
```

Usage

Here is an example of the script in action:

```
Z:\rdn>rmviewprivate.bat
Z:\cc_tools\batch\RmViewPrivate>rmviewprivate.bat

Start of the script

No directory specified, will use .

cleartool ls -recurse -view_only .
del "rmviewprivate.zip"

End of the script

Z:\cc_tools\batch\RmViewPrivate>
```

You will then see the following window:

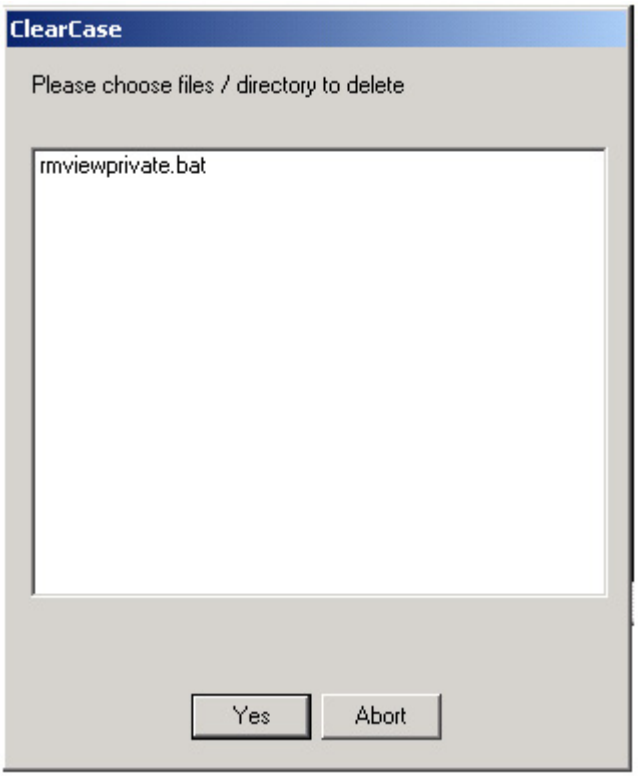


Figure 6: Remove private view file

Back to top

Other Administrative Scripts

The **Send To** menu entry is available in many interfaces: Windows Explorer, ClearCase Explorer, the ClearCase VersionTree Browser, and so on. When using the Version Tree Browser, it is sometimes quite useful to be able to edit an older version (that is, select a version and be able to see its contents). Since we can customize the Windows **Send To** menu by adding a shortcut, we can easily get this functionality.

Adding a Shortcut

Use the following steps to add a shortcut:

1. Navigate to the `C:\Documents and Settings\username\SendTo` folder
2. From the **File** menu, select **New>Shortcut**
3. This will launch the **Create Shortcut** Wizard. The first prompt asks for the location of the item. Type **notepad**, and click **Next**
4. The next prompt asks for a shortcut name. Name the shortcut and click **Finish**

Now the shortcut for Notepad is available through the **Send To** menu. To use it, right-click the version in the Version Tree Browser, select **Send To**, and then click the newly created shortcut to Notepad. Of course, you can also add several other shortcuts: WordPad, Word, etc.

Send to mkbranch script

A ClearCase UNIX user was once complaining to me that the ClearCase GUI utility in UNIX, **xclearcase**, was "twenty times better" than the Windows one, with more functionality, options, and possibilities available. I asked for an example: what is missing in Windows, but available in Unix? The answer was that you could do an "mkbranch" in the UNIX Version Tree Browser.

Then, a few months ago, a new colleague (Benoit Lorendeaux) started at Rational Software Switzerland. He also asked for this functionality. As a challenge, I asked him to create this functionality in Windows. Here is his solution:

Installation

We can reuse the concept of customizing the "Send to" menu. However, instead of invoking Notepad, we will start the following script: Example: execute this command: `ccperl script %1`

Script

```
$version = $ARGV[0];

@VOB_list=`cleartool lsVOB -s`;
$index = 0;
$choix = "";
while ($VOB_list[$index] ne "") {
# printf($VOB_list[$index]);
chomp($choix);
if ($index == 0){
$choix .= $VOB_list[$index];
} else {
$choix .= "," . $VOB_list[$index];
}
$index++;
}
system("clearprompt list -outfile C:\\choice.txt -items $choix -prompt \"Choose VOB\"");
$data_file="C:\\choice.txt";
open(DAT, $data_file) || die("Could not open file!");
@raw_data=<DAT>;
close(DAT);
$choix = pop(@raw_data);
chomp($choix);

@branch_list=`cleartool lstype -s -kind brtype -inVOB VOB:$choix`;
$index = 0;
$choix = "";
while ($branch_list[$index] ne "") {
# printf($branch_list[$index]);
chomp($choix);
if ($index == 0){
$choix .= $branch_list[$index];
} else {
$choix .= "," . $branch_list[$index];
}
$index++;
}
system("clearprompt list -outfile C:\\choice.txt -items $choix -prompt \"Choose branch to
implement\"");
$data_file="C:\\choice.txt";
open(DAT, $data_file) || die("Could not open file!");
@raw_data=<DAT>;
close(DAT);
$choix = pop(@raw_data);
chomp($choix);
$cmd = "cleartool mkbranch -nc -nco -nwarn $choix $version";
system($cmd);
```

VOB backup script

VOB backup has always been a nightmare for ClearCase administrators: you need to safely backup all the VOBs at the same time (which is a time-consuming process), but you also want to keep the VOB lock-time to a minimum (so that normal development, night builds, and MultiSite synchronization can take place). Rational Software has a solution to this nightmare: the VOB_snapshot / VOB_restore functionality. Since I am an old administrator hacker who always wants to have all the solutions on hand, however, I prefer to do it all myself.

Introduction

The purpose of this script is to copy the complete VOB-storage directory (for all the local VOBs) to another directory, where it can be easily backed-up. The script should be run automatically via the ClearCase scheduler.

Note: the script should cooperate with your backup utility.

The Script

This script has been written in Perl so that it provides more output and a better selection of options. Please edit the file, and change the parameter: `$targetPATH = "d:\\nono";` to the appropriate target directory for the copy.

Warning: the target directory should already exist!

The `backupVOB.bat` script

```
@rem = ' PERL for Windows NT -- ccperl must be in search path
@echo off
ccperl %0 %1 %2 %3 %4 %5 %6 %7 %8 %9
goto endofperl
@rem ';
#
#
# Variable to be modified
#
$targetPATH = "d:\\nono";
#
# End Variable to be modified
#
##############################################################################
#
#
# Global variable definition
#
if ($ENV{TEMP}) {
$TDIR=$ENV{TEMP}; }
else {
$TDIR="c:${S}temp";}
$COMPUTERNAME = $ENV{COMPUTERNAME};
$S = "\\";
$PERL = "ccperl";
$NULL = $TDIR . $S . "null";
$JUNK = $TDIR . $S . "junk";
$TMPFX = $TDIR . $S . "." . $$;
$TMP_VOBINFO = $TMPFX . ".VOBstuff";
$COPYCMD = "ccopy ";
$MDEP_TIME_STAMP = 'cmd /c echo y | cmd /c time';
$MDEP_DATE_STAMP = 'cmd /c echo y | cmd /c date';
$MDEP_RMDIR_COMMAND = 'cmd /c rmdir /s ';
$MDEP_RMFILE_COMMAND = 'cmd /c del';
$MDEP_ECHO = 'cmd /c echo';
#
#
# End of global variable definition
#
##############################################################################
#
#
# Function definition
#
sub system_redirect {
local($command) = @_[0];
local($ret_status);
local($tmpname);
$tmpname = &mdep_get_tmpname();
$ret_status = system("$command > $tmpname 2>&1");
#
# Don't call mdep_rmfile - it calls system_redirect.
#
system("$MDEP_RMFILE_COMMAND $tmpname");
return $ret_status;
}
sub mdep_rmfile {
local($file) = @_[0];
local($status);
if($file eq "") {
return $RTN_PATH_NULL;
}
$status = &system_redirect("$MDEP_RMFILE_COMMAND \"$file\"");
if($status) {
return $RTN_NOT_OK;
}

return $RTN_OK;
}
sub mdep_rmdir {
local($dirpath) = @_[0];
local($status);
if($dirpath eq "") {
print "null directory\n";
```

```
return(-1);
}
if(! -d $dirpath) {
print "directory $dirpath doesn't exist\n";
return(-1);
}
$status = &system_redirect("$MDEP_ECHO Y | $MDEP_RMDIR_COMMAND \"$dirpath\"");
return($status);
}
sub mdep_get_tmpname {
$VOB_recover_tmpseq++;
return "${TDIR}\\backupVOB.$$.${VOB_recover_tmpseq}";
}
sub mdep_get_time {
local($tmpname);
local($line);
local($date);
local($time);
local($r_string) = "";
$tmpname = &mdep_get_tmpname();
system("$MDEP_DATE_STAMP > $tmpname");
open(GETDATE,$tmpname);
foreach $line (<GETDATE>) {
chop $line;
if($line =~ /The current date is:\s+[a-zA-Z]+\s+(.*)/) {
$date = $1; last;
}
}
close(GETDATE);
system("$MDEP_TIME_STAMP > $tmpname");
open(GETTIME,$tmpname);
foreach $line (<GETTIME>) {
chop $line;
if($line =~ /The current time is:\s+(.*)/) {$time = $1; last}
}
close(GETTIME);
&mdep_rmfile($tmpname);
$r_string = "$date $time";
return($r_string);
}
#
#
# End of function definition
#
################################################################################
$tstamp = &mdep_get_time();
printf "Backup script started : $tstamp \n";
system("cleartool lsVOB -host ${COMPUTERNAME} > $TMP_VOBINFO 2> $JUNK");
open(VOBS,$TMP_VOBINFO);
while ($VOB=<VOBS>) {
($active, $VOBtag, $VOBstrg) = split(/\s+/,$VOB);
print "\nCopying up : $VOBtag to $targetPATH$VOBtag \n";
system("cleartool lock VOB:$VOBtag");
if( -d $targetPATH.$VOBtag) {
mdep_rmdir($targetPATH.$VOBtag); }
system("mkdir $targetPATH$VOBtag");
print ".";
system("$COPYCMD $VOBstrg\\groups.sd $targetPATH$VOBtag\\groups.sd");
print ".";
system("$COPYCMD $VOBstrg\\identity.sd $targetPATH$VOBtag\\identity.sd");
print ".";
system("$COPYCMD $VOBstrg\\replica_uuid $targetPATH$VOBtag\\replica_uuid");
print ".";
system("$COPYCMD $VOBstrg\\VOB_oid $targetPATH$VOBtag\\VOB_oid");
print ".";
system("$COPYCMD $VOBstrg\\VOB_server.conf $targetPATH$VOBtag\\VOB_server.conf");
print ".";
system("$COPYCMD $VOBstrg\\admin $targetPATH$VOBtag\\admin");
print ".";
system("$COPYCMD $VOBstrg\\c $targetPATH$VOBtag\\c");
print ".";
system("$COPYCMD $VOBstrg\\d $targetPATH$VOBtag\\d");
print ".";
system("$COPYCMD $VOBstrg\\db $targetPATH$VOBtag\\db");
```

```
print ".";
system("$COPYCMD $VOBstrg\\s $targetPATH$VOBtag\\s");
print "\n";
system("cleartool unlock VOB:$VOBtag");
print "\n";
}
close(VOBS);
&mdep_rmfile($TMP_VOBINFO);
&mdep_rmfile($JUNK);
$tstamp = &mdep_get_time();
printf "Backup script stopped : $tstamp \n";
__END__
:endofperl
```

Ccopy.exe

The Perl script uses ClearCase's **ccopy.exe** to copy the files to the target directory. It has been written so that the access control list (ACL) is also copied over. The `ccopy.exe` utility is usually in `%ATRIAHOME%\etc\utils`

Warning: ccopy.exe should be in your search path!

Other Process Steps

- BackupVOB.bat

1. Copy `backupVOB.bat` to the following directory:
   `%ATRIAHOME%\var\scheduler\tasks`
2. Copy `ccopy.exe` to the following directory:
   `%SYSTEMROOT%\system32`

- Set up the Scheduler: Change the Scheduler permission

By default, the ClearCase scheduler has an ACL that grants everyone only "Read" permission. This should be modified to be able to schedule new jobs. To do that, logon as a member of the "clearcase" group (for example, as *ccadmin*), and create a small file with the following content: Everyone: Full. Then, run the following:

```
cleartool schedule -set -acl filename
```

- Create a new task:

1. Go to the following directory:
   `%ATRIAHOME%\var\schedule\tasks`
2. And edit the following file:
   `task_registry`
3. Add at the end, and put the following:

```
Task.Begin
Task.Id: 102
Task.Name: "Backup/Copy VOB"
Task.Pathname: backupVOB.bat
Task.End
```

- Schedule the task

1. Start the ClearCase Administration Console: **Start > Programs > ClearCase Administration > ClearCase Administration Console**
2. Go to the scheduled jobs entry, right click and select **New > job**
3. In the **Job Properties** dialog box, select the **Task** tab, and choose **Backup/Copy VOB**
4. Select the remaining tabs to specify the rest of the scheduling options, such as time, date, frequency, and so on

Once scheduled, it should work!

License usage script

[Many thanks to Arjan Ten Hoopen]

ClearCase license usage has always been a big question: since more and more companies have deployed ClearCase company-wide, it is sometimes difficult to keep track of how many licenses are being used across the enterprise. This can cause conflicts when you want to do a build, or simply when the number of concurrent users has exceeded the number of licenses. Knowing the extent of ClearCase usage is a big feature to have.

Arjan Ten Hoopen developed a full solution for this, based on a scheduled job that displays the results with nice .jpg files.

Enclosed is his solution: ClearLicense.zip

Installation

Follow the instructions in the readme.doc of Arjan's zip file.

Usage

When everything is set up, you can run the script and get a clear picture of the statistical usage of ClearCase licenses.

Example:

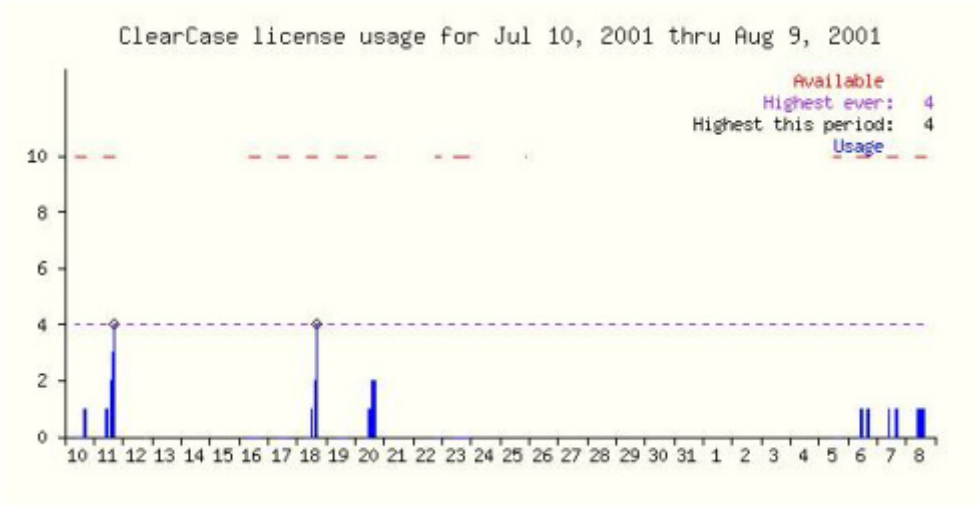


Figure 7: Graphic display of license usage

MultiSite with ftp script

Introduction

ClearCase MultiSite is an add-on product. The main purpose of MultiSite is the ability to create and synchronize replicas of VOBs from several sites world-wide. To do this, ClearCase MultiSite uses a mechanism that consists of two major steps:

1. **Create a replica** -- usually an administrator does this task manually
2. **Synchronize the replicas** -- usually this task is automatically performed using scheduled jobs

Both of these steps are divided into 3 tasks:

1. On the original site, *export* the replica / synchronization packet
2. *Ship / deliver* the packet to the other site
3. On the target site, *import* the replica / synchronization packet.

Note that tasks 1 and 3 are done via ClearCase MultiSite, that is, with MultiSite's own functionality. But note also that task 2 can be accomplished either with the embedded mechanism of MultiSite, called the **shipping_server**, or externally, by another procedure (like delivering over ftp, e-mail, and so on). In the case of a disconnected network, ClearCase provides the capability to package up the CCMS information packets and get them to the other site using whatever medium or mechanism you want (shipping server, ftp, email, and so on).

Here is a solution that does not use MultiSite's shipping_server, but an *ftp server* to transfer the packets.

Concept

To simplify the MultiSite mechanism, a completely new task script (which has the three embedded synchronization steps) has been developed. The script is called `multisite.bat`, and resides in: `%ATRIAHOME%\var\scheduler\tasks`.

This script has 3 main steps:

1. For all VOBs, see if a replica exists, and then perform a `multitool syncreplica -export`
2. Send the generated packets using ftp on an ftp server.
3. Perform a `multitool syncreplica -import`, in other words import all the synchronization packets available.

The script has been developed in Perl, and is not difficult to understand, or to change. Debug output can be switched on/off, and it can be run directly from the command line to be tested.

A script, called `newshipping.bat`, has been developed, and resides in `%ATRIAHOME%\var\scheduler\tasks`. This script does the following:

1. Launches an ftp client, with has as a parameter a file containing the list of instructions to be executed
2. Moves the shipping packets and the shipping order to an archive directory, so that they are available if necessary

Here are the scripts:

Multisite.bat

```
@rem = ' PERL for Windows NT -- ccperl must be in search path
@echo off
set CLEARCASE_VOBLOCKWAIT=120
ccperl %0 %1 %2 %3 %4 %5 %6 %7 %8 %9
goto endofperl
@rem ';
$CFG::verbose = 1;
##############################################################################
#
#
# Global variable definition
#
$VOBLOCKWAIT = $ENV{CLEARCASE_VOBLOCKWAIT};

if ($ENV{TEMP}) {
$TDIR=$ENV{TEMP}; }
else {
$TDIR="c:${S}temp";}

$COMPUTERNAME = $ENV{COMPUTERNAME};
$S = "\\";
if ($ENV{TEMP}) {
$TDIR=$ENV{TEMP}; }
else {
$TDIR="c:${S}temp";}

$PERL = "ccperl";
$NULL = $TDIR . $S . "null";
$JUNK = $TDIR . $S . "junk";
$JUNK1 = $TDIR . $S . "junk1";
$JUNK2 = $TDIR . $S . "junk2";
$JUNK3 = $TDIR . $S . "junk3";
$JUNK4 = $TDIR . $S . "junk4";
$JUNK5 = $TDIR . $S . "junk5";

$TMPFX = $TDIR . $S . "." . $$;
$TMP_VOBINFO = $TMPFX . ".VOBstuff";
$TMP_REPLICAINFO = $TMPFX . ".replicastuff";
$TMP_PACKETINFO = $TMPFX . ".packetstuff";
$TMP_SHIPPINGINFO = $TMPFX . ".shippingstuff";
$TMP_RECEIVEINFO = $TMPFX . ".receivestuff";
$TMP_PINGINFO = $TMPFX . ".pingstuff";

$COPYCMD = "ccopy ";
$MDEP_TIME_STAMP = "cmd /c echo y | cmd /c time";
$MDEP_DATE_STAMP = "cmd /c echo y | cmd /c date";
$MDEP_RMDIR_COMMAND = "cmd /c rmdir /s ";
$MDEP_RMFILE_COMMAND = "cmd /c del";
$MDEP_ECHO = "cmd /c echo";

#
#
# End of global variable definition
#

##############################################################################
#
#
# Function definition
#

sub system_redirect {

local($command) = @_[0];
local($ret_status);
local($tmpname);

$tmpname = &mdep_get_tmpname();
$ret_status = system("$command > $tmpname 2>&1");

#
# Don't call mdep_rmfile - it calls system_redirect.
#
system("$MDEP_RMFILE_COMMAND $tmpname");
```

```
return $ret_status;


}

sub mdep_rmfile {

local($file) = @_[0];
local($status);

if($file eq "") {
return $RTN_PATH_NULL;
}

$status = &system_redirect("$MDEP_RMFILE_COMMAND \"$file\"");

if($status) {

return $RTN_NOT_OK;
}

return $RTN_OK;
}

sub mdep_rmdir {

local($dirpath) = @_[0];
local($status);
if($dirpath eq "") {
print "null directory\n";
return(-1);
}

if(! -d $dirpath) {
print "directory $dirpath doesn't exist\n";
return(-1);
}

$status = &system_redirect("$MDEP_ECHO Y | $MDEP_RMDIR_COMMAND \"$dirpath\"");

return($status);
}

sub mdep_get_tmpname {

$VOB_recover_tmpseq++;
return "${TDIR}\\backupVOB.$$.${VOB_recover_tmpseq}";

}

sub mdep_get_time {

local($tmpname);
local($line);
local($date);
local($time);
local($r_string) = "";
$tmpname = &mdep_get_tmpname();
system("$MDEP_DATE_STAMP > $tmpname");
open(GETDATE,$tmpname);
foreach $line (<GETDATE>) {
chop $line;
if($line =~ /The current date is:\s+[a-zA-Z]+\s+(.*)/) {
$date = $1; last;
}
}
close(GETDATE);
system("$MDEP_TIME_STAMP > $tmpname");
open(GETTIME,$tmpname);
foreach $line (<GETTIME>) {
chop $line;
if($line =~ /The current time is:\s+(.*)/) {$time = $1; last}
}
```

```
close(GETTIME);

&mdep_rmfile($tmpname);
$r_string = "$date $time";
return($r_string);

}


sub dbgprint {
return if (! $CFG::verbose);
system "cmd /c echo @_";
# print @_;
# print STDERR @_ if $CFG::stderr_msgs;
}

#
#
# End of function definition
#

################################################################################
$tstamp = &mdep_get_time();
printf("Hello Dany...");
dbgprint "MultiSite script started : $tstamp \n";
dbgprint "Maximum Wait time for Lock : $VOBLOCKWAIT minutes";
dbgprint "Phase 1 : syncreplica -export ";
system("cleartool.exe lsVOB -host ${COMPUTERNAME} > $TMP_VOBINFO 2> $JUNK");
open(VOBS,$TMP_VOBINFO);
while ($VOB=<VOBS>) {
($active, $VOBtag, $VOBstrg) = split(/\s+/,$VOB);
system("multitool lsreplica -s -siblings -inVOB $VOBtag > $TMP_REPLICAINFO 2>
$JUNK1");
open(REPLICAS,$TMP_REPLICAINFO);
while ($replica=<REPLICAS>) {
chop $replica;
system("multitool syncreplica -export -ship $replica\@$VOBtag >
$TMP_PACKETINFO 2> $JUNK2");
dbgprint " $VOBtag : $replica \n";
# dbgprint " Standard output : \n";
# system("cmd /c type $TMP_PACKETINFO");
# dbgprint " Standard Error : \n";
# system("cmd /c type $JUNK2");
&mdep_rmfile($TMP_PACKETINFO);
&mdep_rmfile($JUNK2);

}
close(REPLICAS);
&mdep_rmfile($TMP_REPLICAINFO);
&mdep_rmfile($JUNK1);
}
close(VOBS);
&mdep_rmfile($TMP_VOBINFO);
&mdep_rmfile($JUNK);

dbgprint "MyPhase 2 : Shipping_server";
system("ping -n 10 -w 2000 -l 10000 psdev-ftp > $TMP_PINGINFO 2> $JUNK5");

# This is the normal command for the shipping server :
#system("shipping_server -poll > $TMP_SHIPPINGINFO 2> $JUNK3");
# For E+H, another shipping server has been used :
system("cmd /c \"c:\\Program Files\\ClearCase\\var\\scheduler\\tasks\\newshipping.bat\"
> $TMP_SHIPPINGINFO 2> $JUNK3");
#dbgprint " Standard output : \n";
#system("cmd /c type $TMP_SHIPPINGINFO");
#dbgprint " Standard Error : \n";
#system("cmd /c type $JUNK3");
dbgprint "Phase 3 : syncreplica -import ";
system("multitool syncreplica -import -receive > $TMP_RECEIVEINFO 2> $JUNK4");
#dbgprint " Standard output : \n";
#system("cmd /c type $TMP_RECEIVEINFO");
#dbgprint " Standard Error : \n";
#system("cmd /c type $JUNK4");
$tstamp = &mdep_get_time();
```

```
dbgprint "MultiSite script stopped : $tstamp";
exit(0);
__END__
:endofperl
```

newshipping.bat

```
echo "Newshipping started..."
attrib -r "C:\Program Files\ClearCase\var\shipping\ms_ship\outgoing\sync*"
ftp -v -s:"C:\Program Files\ClearCase\var\scheduler\tasks\sendinput.txt"
copy "C:\Program Files\ClearCase\var\shipping\ms_ship\outgoing\sync*" "C:\Program
Files\ClearCase\var\shipping\ms_ship\outgoing\archive"
del "C:\Program Files\ClearCase\var\shipping\ms_ship\outgoing\sync*"
copy "C:\Program Files\ClearCase\var\shipping\ms_ship\outgoing\sh_*" "C:\Program
Files\ClearCase\var\shipping\ms_ship\outgoing\archive"
del "C:\Program Files\ClearCase\var\shipping\ms_ship\outgoing\sh_*"
ftp -v -s:"C:\Program Files\ClearCase\var\scheduler\tasks\getinput.txt"
```

sendinput.txt

```
open ftp-server
ftpuser
ftpuser
cd "/multisite/ftp/outgoing"
lcd "C:\Program Files\ClearCase\var\shipping\ms_ship\outgoing"
hash
binary
prompt
mput sync*
quit
```

getinput.txt

```
open ftp-server
ftpuser
ftpuser
cd "/multisite/ftp/outgoing"
lcd "C:\Program Files\ClearCase\var\shipping\ms_ship\incoming"
hash
binary
prompt
mget sync*
quit
```

Scheduler

A new task needs to be added to the ClearCase scheduler to automatically run the `multisite.bat` script. See the ClearCase scheduler documentation to learn how best to set it up.

Limitations:

The workaround presented in the preceding paragraphs has the following limitations:

- The output that can generate the scripts is limited to 511 bytes, as described in the help pages:
  - **Show Completion Details** -- Displays information about the last time the job was run: the process ID, the start and end times, the exit status, and the first 511 bytes of any output or error messages from the job.
- The script has been developed for only two replicas, that is, it cannot handle multiple replicas in several sites.
- The ClearCase AdminConsole uses the following icon to display warnings after running the script:

This *doesn't necessarily mean that there is a problem* with the script, but but rather displays a completion message.

Back to top

Copy Merge script

Sometimes elements in ClearCase VOBs cannot be merged; they may be binary files, objects, or executables. But if you want to do parallel development, or you are using UCM, you need to perform merge operations. Having a way to specify that you don't want to merge a file, but just "copy over" the version, either from the source branch, or the target branch, is a welcome feature.

Michel Lohr of Rational Software Netherlands, developed a solution.

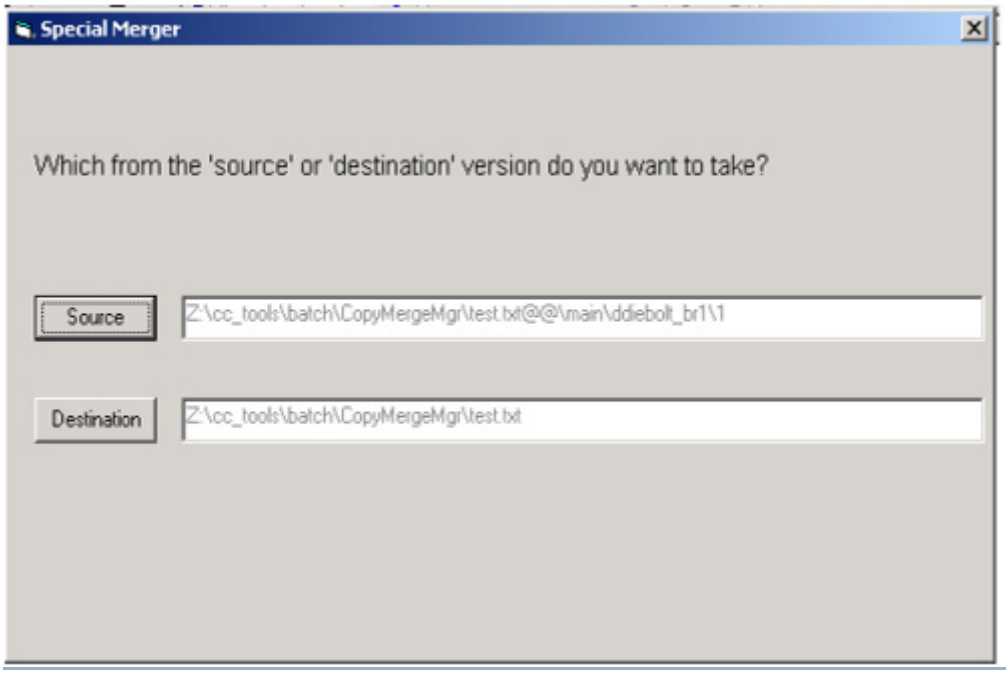
Here is his solution: CopyMerge.zip

Installation

Follow the instructions of "Forced copy merger.doc"

Usage

Once you perform a merge of a file having as type manager the "special_merge," you will get the following dialog box:



Back to top

Compare label script

ClearCase, by default, does not have any GUI functionality to compare two labels, i.e., to get the list of all versions that differ between two labels sets. However, this functionality exists over the command line with the cleartool find command, so let's make a compare function over the GUI!

Installation

Caution: You MUST have local administration rights to be able to make the customization: the customization is done by user profile.

1. Start the clearmenuadmin.exe utility
2. For the "compare label" option, select the **Object type** tab, the "VOB" object
3. Choose as "Object state," the checked-in state

- Menu Text: Compare labels...
- Help Text: Compare two labels...
- Command Type: Executable
- Command:
  ">Place Where the tool is>\CompareLabel.exe"

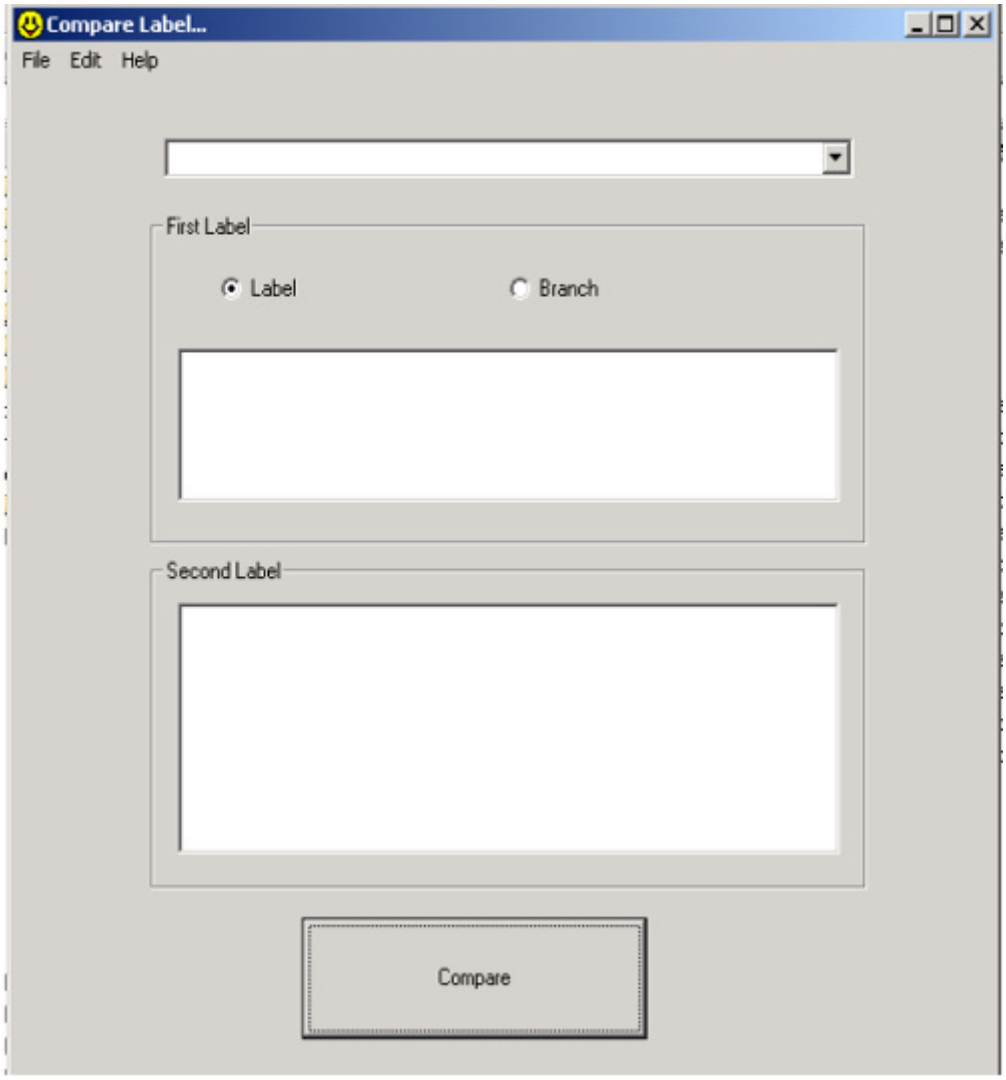
```
Initial Directory: $dir_or_file1
Arguments: "$dir_or_file1"
```

Comment:

You can put this new menu entry to the menu contents pane by pressing the **<- add** button, and move it with the **Move up** button.

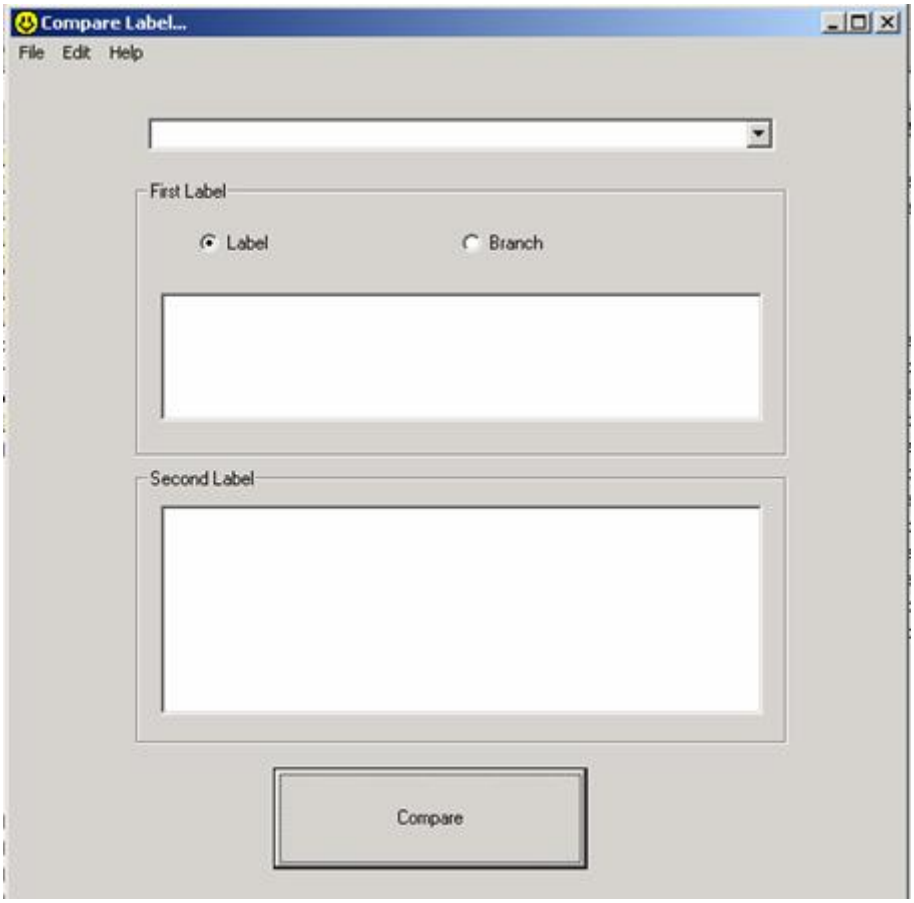
Usage

Using Windows Explorer, select the VOB, click the right mouse button, and the new menu: **Compare labels...** should appear:

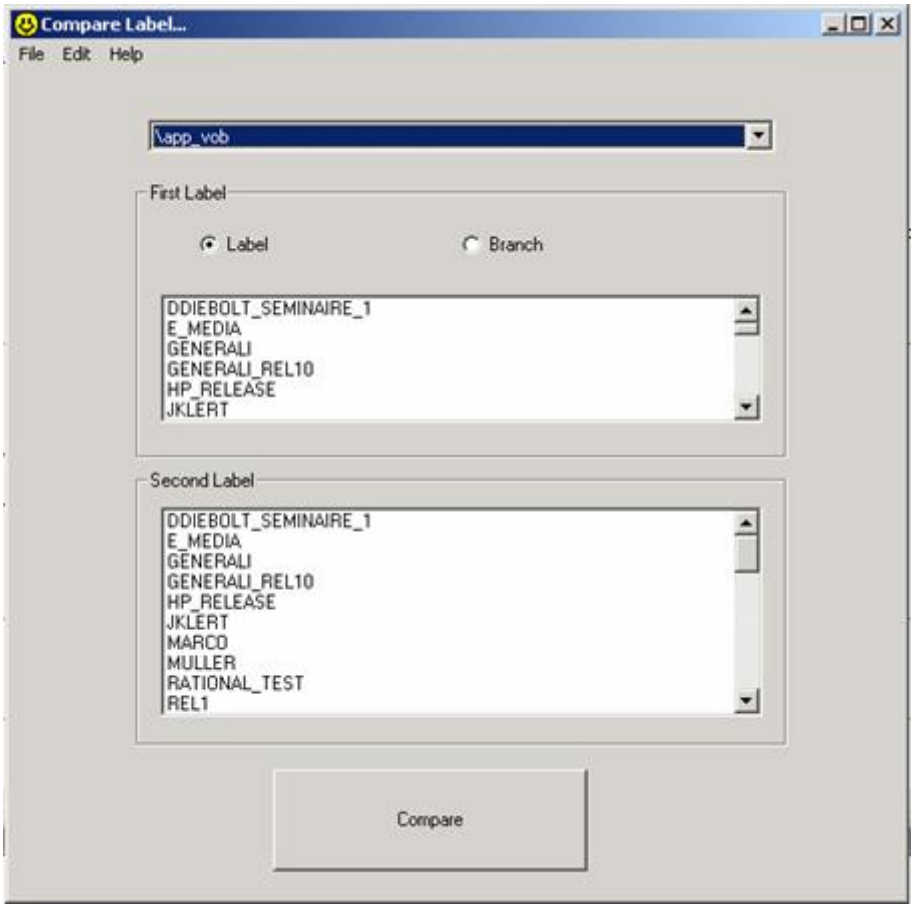


If you don't have a view called "Administration_Views," then the following dialog box will appear first:

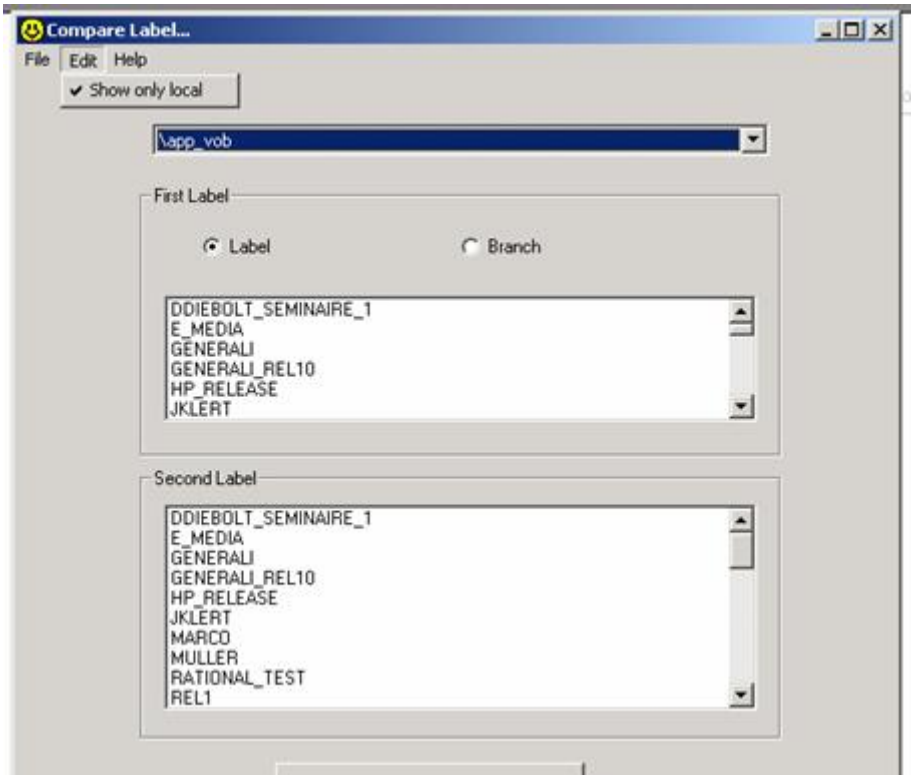
You then need to choose one of the views to have the tool work properly, and then to choose a VOB first to drop down the combo box. You can then either select to compare any two labels, or compare the latest from a branch and a label.



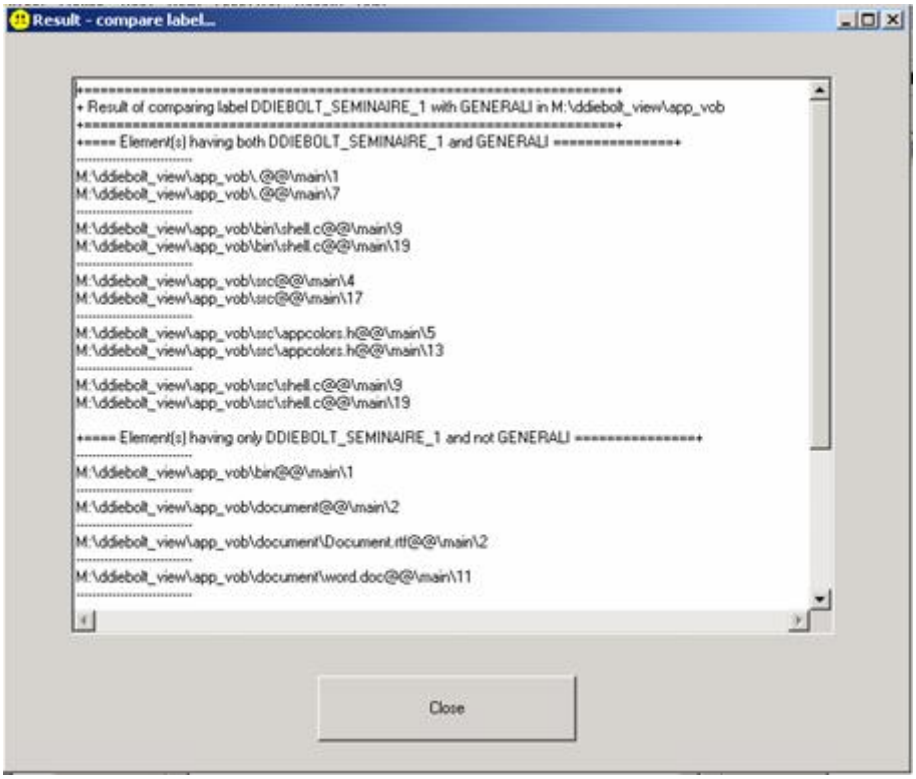
1. Choose the first label / branch
2. Choose a second label
3. Click the **Compare** button to start the compare process



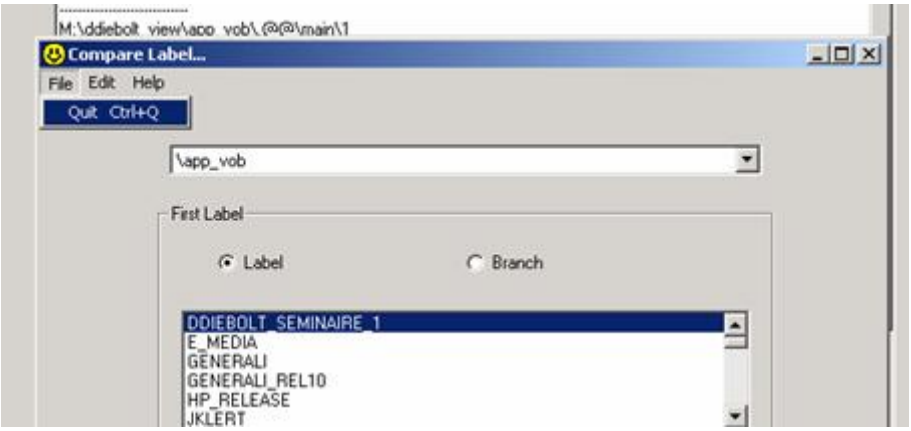
The menu item **Edit > Show only local** is for displaying either the list of local labels / branches, or the labels / branches coming from the AdminVOB.



After comparing, the results will be displayed in a **Result -- compare label** dialog box:



You can then select the text, and copy/paste it to your favorite application. With the menu **File > Quit** (or Ctrl-Q), you can close the whole application.



The source of CompareLabel.exe

The application `CompareLabel.exe` has been developed with Visual Basic v6.0 and the ClearCase Automation Library (COM) called "CAL."

The Visual Basic project is called `CompareLabel.vbp`, and has the following forms:

- `main.frm`: contains the main dialog box description and code
- `frmResult.frm`: contains all the features of the **Results** dialog box
- `frmAbout.frm`: contains all the features of the About dialog box
- `frmChooseView.frm`: contains all the work for the **Choose view** dialog box, if an "Administrations_View" does not already exist
- `frmProgressDialog.frm`: contains all the logic of the **Progress** dialog box

Also, a Visual Basic module is used called `General Definition.bas`. See the source files for the logic of the application.

Back to top

Conclusion:

With these ten scripts, we've covered several ways to extend ClearCase to meet our needs. We've seen simple examples; some with just a command, others with Perl scripts, and even full integration written in Visual Basic and CAL. After working for more than seven years with ClearCase, sometimes quite intensively, each day I discover new features and new ideas to make life better for both developers and ClearCase administrators.

Back to top

Downloads

Name	Size	Download method
comparelabel.zip	281 KB	HTTP
CopyMerge.zip	47 KB	HTTP

→ Information about download methods

About the author

Daniel Diebolt is a Senior Technical Representative with IBM in Switzerland, specializing in Rational software.

Rate this page

Please take a moment to complete this form to help us better serve you.

Did the information help you to achieve your goal? ○ Yes ○ No ○ Don't know

Please provide us with comments to help improve this page:

How useful is the information? ○ 1 Not useful ○ 2 ○ 3 ○ 4 ○ 5 Extremely useful

Submit

Back to top