About the ClearCase Type Manager

Overview, design and custom implementation

Maréchal Laurent

December 1, 2011

ABOUT THE CLEARCASE TYPE MANAGER....1

INTRODUCTION....3

OVERVIEW OF THE TYPE MANAGER....4

ClearCase element type....4
The magic file....4
ClearCase type manager....5

TYPE MANAGER COMPONENT AND DEFINITION....8

The map file....8
The type manager....11
Interface....11
Interface options....12
Operation return value....12
Operation description and arguments....13
create_element....14
create_branch....15
create_version & create_version64....16
construct_version....17
delete_branches_versions....18
compare....20
xcompare....21
merge....22
xmerge....23
context_merge....24
annotate....25
get_cont_info....26

CUSTOMIZING THE TYPE MANAGER....27

Identify the need....27
Element type creation and use....27
Change existing type manager....29
Full type manager change....31

PRACTICAL EXAMPLE....35

UTF16 handling (diff/merge tool with Beyond Compare)....35
Goal....35
Analyze....35
Procedure....35

NOTES AND WARNING....38

REFERENCES....39

Introduction

Creating a new type manager and or modifying an existing type manager is not supported by IBM Rational Technical Support. This document is provided for information only. Using this information to modify or create your own type manager is not supported by IBM Rational due to the nature of what a type manager does. The type manager acts on the container of your VOB; this means on **your** data. If a custom type manager is corrupting your data, Rational can not and will not be held liable for such corruption. To create or modify a type manager means you do so at your own risk.

There is information in the Reference Guide and Administrator's Guide on how to set up a custom type manager under UNIX® and Linux®, but there is no other information on what parameters they use or how to set them up. This leads to a few issues on what is supported or not supported. The following article will attempt to clarify the current position of IBM Rational Technical Support on the subject as well as to answer the more common questions received.

What will be primarily discussed in this article are the element types and type managers. Both will be explained, as well as instructions provided on how to set up a type manager. All of the following information is based on the ClearCase version 8.0.0.0 running on Microsoft® Windows®. Users running under a UNIX or Linux can still refer to this article; however, some minor modification will be required. This still applies to previous versions of ClearCase, however, version 8.0 specific features will, of course, not be available.

Please review carefully the "Notes and warning" section at the end of this document.

Overview of the type manager

ClearCase element type

All elements in a ClearCase VOB have an element type associated with them. This type lets ClearCase know how to handle the different operations (such as creating a new version, or reading a specific version).
The following is example output for the syntax shown in the first line below:

```
>cleartool describe -long catalog.bin
version "catalog.bin@@\main\1"
  created 08-Mar-05.16:10:47 by Marechal Laurent (HOST1)
  Element Protection:
    User : IBM \nlxxxx : r--
    Group: IBM \yyyy : r--
    Other:           : r--
  element type: compressed_file
  predecessor version: \main\0
```

Element types can be created by any user. They are primarily used to assign a customized type to specific files for example, c_source to all .c or .cpp file). This allows particular ClearCase commands to be run against these element types (such as a `cleartool find` restricted by element type). Creation of custom element types is supported and even required sometimes (for example, the no-merge type for UCM in ClearCase version before v2003.06.00).

The magic file

The magic file (that is, `default.magic`) is used to match new element creation with the desired element type. This file is used at element creation (`mkelem`) time. Several rules are analyzed one after each other in order to determine the right element type to use for the newly created file. If no rules match, then the last one will be used. Rules can be based on filename, extension, content (printable) or magic number.

An extract of the `.magic` file would be:

```
program compressed_file : -name "*.[eE][xX][eE]" | -name
"*.bin" ;
```

This would activate for a file of:

- extension .bin
- extension .EXE
- extension .exe
- extension <any uppercase or lowercase combination of exe>

This could also be simplified to:

```
program compressed_file : -name "*.exe" | -name "*.bin" ;
```

When a new file is added to source control, if the extension is equal to .exe or .bin then ClearCase will try to use the element type `program` and if it does not exist, it will default to `compressed_file`. Remember that this file is only used at creation time (`mkelem`, `add to source control`) and not at any new version or branch creation. Review the `cc.magic` page in the reference guide (cleartool man cc.magic) for more information.

ClearCase type manager

Each element type is bound to a type manager. There are several predefined element types provided by default (number depends on the ClearCase release) like `file`, `text_file`, `compressed_file`. A full list of the existing types can be retrieved with a single `cleartool` command. On ClearCase 8.0.0.0 it returns the following:

```
>cleartool lstype -s -kind eltype
binary_delta_file
compressed_file
compressed_text_file
directory
file
file_system_object
html
ms_word
rose
rosert
text_file
utf16be_file
utf16le_file
utf32be_file
utf32le_file
utf8_file
xde
xml
```

Each of these element types has an associated type manager. This can be seen when you describe any of these element types.

```
>cleartool describe -long eltype:compressed_file
element type "compressed_file"
  created 08-Mar-05.14:26:49 by Marechal Laurent (HOST1)
  "Predefined element type used to represent a file in
compressed format."
  owner: IBM \nlxxx
  group: IBM \yyyy
  scope: this VOB (ordinary type)
  type manager: z_whole_copy
  supertype: file
  meta-type of element: file element
```

The last three lines contain the type manager information.. These line define the type manager used for this element type (`z_whole_copy`) and the supertype used (`file`). For more information review the ClearCase reference guide on the topic of `mkeltype` (cleartool man mkeltype) for more information.

The following default type managers are provided with ClearCase 8.0.0.0 (refer to the IBM Rational ClearCase Information Center under the topic of type_manager for further details):

Manager	Function
whole_copy	Stores any data. Stores a whole copy of each version in a separate data container file.
z_whole_copy	Stores any data. Stores each version in a separate, compressed data container file using the

Manager	Function
	ccgzip compression program. (ccgzip is a utility that implements IETF RFC 1951 + 1952 compression using zlib source code). Note that compressed files generally take more time to check in (because they must be compressed) and to reconstruct when first accessed (first cleartext fetch).
text_whole_copy	Stores text files only. The storage mechanism is the same as that for z_whole_copy. Different versions of a file can be stored in different character encodings. The compare and merge functionality is similar to that of the text_file_delta type manager.
text_file_delta	Stores text files only (including those with multibyte text characters). Stores all versions in a single structured data container file. (On UNIX and Linux, similar to an SCCS s. file or an RCS ,v file.) Uses incremental file differences to reconstruct individual versions on the fly.
z_text_file_delta	Stores text files only. Stores all versions in a single structured data container file, in compressed format using both the ccgzip compression program and deltas.
binary_delta	Stores any data. Stores each branch's versions in a separate, structured compressed data container file using ccgzip. Uses incremental file differences to reconstruct individual versions on the fly. Version deltas are determined by comparing files on a per-byte basis.
_html	Stores HTML source. Stores information and reconstructs versions in the same way as the **text_file_delta** manager from which it is derived. Has its own **compare**, **xcompare**, **merge** and **xmerge** methods.
_html2	Same as the **_html** manager, except that **_html2** is based on the **binary_delta** manager.
_ms_word	Stores Microsoft® Word documents. Stores information and reconstructs versions in the same way as the **z_whole_copy** manager from which it is derived. On Windows, has its own **xcompare** and **xmerge** methods.
_rftdef **_rftmap** **_rftvp**	Stores XDE® Tester artifacts. Stores information and reconstructs versions in the same way as the **binary_delta** manager from which it is derived.
_rose	Stores IBM Rose artifacts. Stores information and reconstructs versions in the same way as the **text_file_delta** manager from which it is derived. On Windows, has its own**compare**, **xcompare**, **merge**, and **xmerge** methods.
_rose2	Same as the **_rose** manager, except that **_rose2** is based on the **binary_delta** manager.
_rosert	Stores IBM Rose RealTime artifacts. Stores information and reconstructs versions in the same way as the **binary_delta** manager from which it is derived. On Windows, has its own **compare**, **xcompare**, **merge**, and **xmerge** methods.
utf8_file_delta	Stores 8-bit Unicode Transformation Format source. The compare/merge functionality is similar to that of the **text_file_delta** type manager.
utf16be_file_delta	Stores 16-bit big-endian Unicode Transformation Format source. The compare/merge functionality is similar to that of the **text_file_delta** type manager.
utf16le_file_delta	Stores 16-bit little-endian Unicode Transformation Format source. The compare/merge functionality is similar to that of the **text_file_delta** type manager.
utf32be_file_delta	Stores 32-bit big-endian Unicode Transformation Format source. The compare/merge functionality is similar to that of the **text_file_delta** type manager.
utf32le_file_delta	Stores 32-bit little-endian Unicode Transformation Format source. The compare/merge functionality is similar to that of the **text_file_delta** type manager.

Manager	Function
_xde	Stores IBM XDE artifacts. Stores information and reconstructs versions in the same way as the **text_file_delta** manager from which it is derived. On Windows, has its own **merge**, **xmerge**, **compare**, and **xcompare** methods.
_xde2	Same as the **_xde** manager, except that **_xde2** is based on the **binary_delta** manager and it provides a **context_merge** method.
_xml	Stores XML source. Stores information and reconstructs versions in the same way as the **text_file_delta** manager from which it is derived. On Windows, has its own **compare**, **xcompare**, **merge**, and **xmerge** methods.
_xml2	Same as the **_xml** manager, except that **_xml2** is based on the **binary_delta** manager.
directory	Not involved in storing or retrieving directory versions, which reside in the VOB database, not in a source storage pool. This type manager compares and merges versions of the same directory element.

Each type manager is meant to be used with a specific kind of file 'content' (text, binary…).
Not all files can be handled the same way. File operations (such as merge and diff) as well as file storage (delta vs. whole copy) will depend on the file format. Because of this there are several type managers created by default. The `*_delta` type manager only stores the data changed between each version. The `z_*` are storing the information in a compressed mode. Other types are created to support different tools such as IBM/Rational Rose/XDE and Microsoft MS-Word.

The type manager is called for several operations (`creating element`, `creating branch`, `creating version`, `fetching specific version`...) and is spawned by a ClearCase process. The type manager process will run from the location where it is called.

- For operations that create/delete/modify source containers (`cleartool/GUI/multitool-import mkelem`, `mkbranch`, `checkin`, `checkout-w/auto-mkbranch`, `rmver`, `rmbranch`, `rmelem`), the type manager runs as a child process of `cleartool/GUI/multitool-import`.

- For operations that create cleartext containers (reading version data *via* the MVFS for a dynamic view, via cleartool update, checkout, mkbranch-w/co for a snapshot view), the type manager runs as a child process of the view server. `Multitool-export` also reads version data and the type manager runs as a child process of `multitool-export`.

This also means that if you introduce a type manager, then all your clients and servers must have it installed. If you use MultiSite, then this also applies for all other sites.

Type manager component and definition

The type manager should not be mistaken for or confused with the element type. These are two different things. The different components that make up a customized type solution include the following:

- A new element type (defines which type manager and supertype would be used for all elements of this type.)
- An entry in the magic file (to bind future element creation to the new type.)
- An entry in the map file (to match the operation [merge or diff etc…] with the type manager to use.)
- A program that will be used for handling all of the possible operations for the type manager (merge, diff etc…).

The element type and magic file will not be described in this article as they are referenced clearly in the different ClearCase manuals.

- Element Type
- Magic File

The map file

Under Windows, the map file is located in the \lib\mgrs directory of your ClearCase installation (by default C:\Program Files\IBM\RationalSDLC\ClearCase\lib\mgrs). This file contains the mapping between the different type managers, the operations and the tool to use.
The path to which the type managers reside does not matter to the map file, so long as the path resolves to the type manager application. “Paths" that are acceptable in the map file are:

- UNC paths
- Absolute paths
- Relative (to the location of the map file) paths
- Registry keys that contain the path to the executable to run

The following is an extract from the map file under Windows:

```
text_file_delta    construct_version
..\..\bin\tfdmgr.exe
text_file_delta    create_branch
..\..\bin\tfdmgr.exe
text_file_delta    create_element
..\..\bin\tfdmgr.exe
text_file_delta    create_version
..\..\bin\tfdmgr.exe
text_file_delta    delete_branches_versions
..\..\bin\tfdmgr.exe
text_file_delta    compare
..\..\bin\cleardiff.exe
text_file_delta    xcompare
..\..\bin\cleardiffmrg.exe
text_file_delta    merge
..\..\bin\cleardiff.exe
text_file_delta    xmerge
..\..\bin\cleardiffmrg.exe
```

```
text_file_delta    annotate
..\..\bin\tfdmgr.exe
text_file_delta    get_cont_info
..\..\bin\tfdmgr.exe
```

This is based on the `text_file_delta` type manager. For each operation required, a specific tool can be called. The same one can also be used, but in this case, it is up to the tool to make the difference in the operation requested (*via* the argument).

The following operations are available:

`construct_version`	Used to request the construction of a cleartext for a given version.
`create_branch`	Used to create a new branch.
`create_element`	Used to create a new element.
`create_version`	Used to create a new version <2Gb.
`create_version64`	Used to create a new version >2Gb.
`delete_branches_versions`	Used to delete a version/branch
`compare`	Used when comparing from the command line.
`xcompare`	Used when comparing with the GUI.
`merge`	Used when merging from the command line.
`xmerge`	Used when merging from the GUI.
`annotate`	Used with the annotate command.
`get_cont_info`	Used to get information on the container.

Here is an example of when a user checks in a new version of type "`text_file`" (`create_version`):

- ClearCase checks the checkin request is valid:
  - The file is actually in checkout mode
  - The user has all permission to perform the operation
- ClearCase checks the map file for the operation and the element type to determine the type manager to use.
- ClearCase gathers the information to pass to the type manager.
- ClearCase creates a new container in the VOB storage.
- ClearCase then invokes the `text_file_delta` type manager to construct and store the new version, passing all the required arguments.
- The `text_file_delta` manager saves all the data from the predecessor version's container in the new container and incorporates the data for the new version, then returns a status indicating the new container should be used and the old should be discarded.
- ClearCase removes the write permissions from the new container, updates the VOB database to record the checkin and to change all versions that used the old container to use the new container, then it deletes the old container. If the type manager or the VOB update fails, ClearCase instead deletes the new container, leaves the database unchanged, and fails the checkin.

Under UNIX or Linux, this map file does not exist. It is replaced by a directory (one for each type manager) containing a set of symlink (one for each operation) pointing to the corresponding program acting as the type manager program. The main directory is /usr/atria/lib/mgrs and will contain an entry for each of the defined type managers, as an example below:

```
drwxr-xr-x     512  binary_delta
drwxr-xr-x     512  directory
```

```
drwxr-xr-x    512  _html
drwxr-xr-x    512  _html2
-r--r--r--  41277  mgr_info.h
-r-xr-xr-x   3484  mgr_info.sh
drwxr-xr-x    512  _ms_word
drwxr-xr-x    512  _rftdef
lrwxrwxrwx      7  _rftmap -> _rftdef
lrwxrwxrwx      7  _rftvp -> _rftdef
drwxr-xr-x    512  _rose
drwxr-xr-x    512  _rose2
lrwxrwxrwx      6  _rosert -> _rose2
drwxr-xr-x    512  text_file_delta
drwxr-xr-x    512  whole_copy
drwxr-xr-x    512  _xde
drwxr-xr-x    512  _xde2
drwxr-xr-x    512  _xml
drwxr-xr-x    512  _xml2
drwxr-xr-x    512  _xtools
drwxr-xr-x    512  _xtools2
drwxr-xr-x    512  z_text_file_delta
drwxr-xr-x    512  z_whole_copy
```

Some of the directory entries are, in fact, linked to other ones. This is because they share the same program to handle the type manager calls.

For each one, there will be an entry matching the type manager call and will point to a program.

```
lrwxrwxrwx   6  annotate -> tfdmgr
lrwxrwxrwx  22  compare -> ../../../bin/cleardiff
lrwxrwxrwx   6  construct_all_versions -> tfdmgr
lrwxrwxrwx   6  construct_version -> tfdmgr
lrwxrwxrwx   6  create_branch -> tfdmgr
lrwxrwxrwx   6  create_element -> tfdmgr
lrwxrwxrwx   6  create_version -> tfdmgr
lrwxrwxrwx   6  delete_branches_versions -> tfdmgr
lrwxrwxrwx   6  get_cont_info -> tfdmgr
lrwxrwxrwx  22  merge -> ../../../bin/cleardiff
-r-xr-xr-x 33K  tfdmgr
-r-xr-xr-x 33K  tfdmgr.old
lrwxrwxrwx  23  xcompare -> ../../../bin/xcleardiff
lrwxrwxrwx  23  xmerge -> ../../../bin/xcleardiff
```

Additionally, under Microsoft Windows, we also use the registry. For every operation that involves map file access, the Windows registry is tested first. Specific keys are required to exist and these follow the same naming and order as the physical files.

The root for the registry definition is:

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Atria\ClearCase\CurrentVersion\Type Managers\<NAME>]
```

If we find the key in the registry, then the map file is not used. This is important to remember as putting different definitions for the same type manager in both will result in the map file being ignored.

Direct modifications to this registry entry are not supported as it may be removed from the product in a future update and could cause unexpected results. The normal supported way is to use the map file.

The type manager

As seen before, the type manager is responsible for handling the request and performs several operations.. A single executable or script can implement all or only part of the possible operation. The map file will be used to determine which executable or script will be called. The parameter provided to the type manager will include the operation so it can handle more than one in the same code. The term ”type manager” refers to the set of executables, scripts... that will allow the full set of operations.

The type manager manipulates cleartext container but also source container. Only the type manager knows the format of these files and is responsible for handling the versions. ClearCase relies on the type manager for access to the repository of the code. This also means that if there is any issue in the type manager, there are no workarounds or rollbacks to fix the data (except the last backup you have done).

Custom type managers are not supported by IBM Rational as they are responsible for all the data handling in your VOB. In case of defect or malfunction of your type manager, you are exposed to irreversible data loss and do so at your own risk.

A primary reference moving forward will be the file \lib\mgrs\mgr_info.h from your ClearCase installation. This file contains all information on a type manager implementation.

Interface

The type manager program can be used either to perform one specific operation or several different operations. When ClearCase calls this type manager, it includes the operation in the argument list and then must check the argument and execute the corresponding call. As long as the interface is respected, the type manager can be implemented in any language possible. Under Windows, only the .exe and .com file can be used. It is not possible to use any non-executable file on Windows at this time (bat, pl, wsh, vbs…). It is also not possible to use parameters or switches to the executable to run commands at this time. A change request has been submitted to allow any kind of executable to be used under Windows; the reference number for tracking purposes is RATLC01005985.

Arguments are passed as pure ASCII string and only an exit value is returned.

Operations that can be included in the type manager are the following (as seen in the argument list provided when the type manager is run):

create_element	compare
create_branch	xcompare
create_version	merge
construct_version	xmerge
delete_branches_versions	context_merge
get_cont_info	annotate
create_version64	

These operations are matching the ones from the map file as seen earlier, in Section 3.1.

Interface options

A few options can also be passed to the type manager. These options are coming from ClearCase and not from the users. The type manager must know how to handle them. Here are the actual possible options that the type manager could receive:

-abort	-fname	-serial
-all	-hstack	-tiny
-base	-blank_ignore	-versions
-branches	-out	-vstack
-columns	-parallele	-windows
-diff	-qall	-qntrivial
-directory	-q	

Most of them are identical to the standard cleartool command and an explanation can be found in the reference manual for the corresponding equivalent.

Operation return value

After execution of an operation, an exit status must be returned. The status value will depend on the result and on the operation. The status will be returned to ClearCase which will then accept or cancel the execution of the command. The following is based on the \lib\mgrs\mgr_info.h file.

Ret.Val.	Constant from mgr_info.h	Description
0	MGR_OK	Value indicating operation succeeded.
1	MGR_FAILED	Value indicating operation failed. Any unexpected return value is also treated as a failure.
2	MGR_FAILED_NO_PRED_CONT	Return value for create operation, indicating failure as previous container does not exist. Common case if container was renamed before manager could open it.
101	MGR_STORE_KEEP_NEITHER	Return value for delete operation, indicating that both the old container and the new container should be discarded.
102	MGR_STORE_KEEP_JUST_OLD	Return value for delete operation, indicating that the old container should be used and the new container not needed.
103	MGR_STORE_KEEP_JUST_NEW	Return value for delete operation, indicating that the old container should be discarded and the new container should be used.
104	MGR_STORE_KEEP_BOTH	Return value for create operation, indicating that both the old and new containers should be kept. A new container has been created to store the requested update. Any versions that used the old container should continue to use the old container. The new version should use the new container.
(1<<7){?}	MGR_STORE_FILES_ARE_IDENT	Special value to OR into a MGR_STORE_* status to indicate that the version stored has the same data as its predecessor.
101	MGR_CONSTRUCT_USE_SRC_CONTAINER	Return value for construct operation, indicating that the source container can be used as the cleartext for the version.
102	MGR_CONSTRUCT_USE_NEW_FILE	Return value for construct operation, indicating that cleartext has been constructed at the supplied pathname.
101	MGR_DELETE_KEEP_NEITHER	Return value for create operation, indicating that the old container, if any, can be discarded and the new container is not needed.

102	MGR_DELETE_KEEP_JUST_OLD	Return value for create operation, indicating that the old container should be saved, and the new container is not needed. Any versions that used the old container should continue to use the old container.
103	MGR_DELETE_KEEP_JUST_NEW	Return value for create operation, indicating that the old container should be discarded and the new container should be used. The new container has all the information in the old container plus the requested update. All versions that used the old container should now use the new container. The old container can then be discarded.
101	MGR_INFO_NO_INFO	Return value for 'get cont info' operation, indicating that the manager does not maintain version identification information. Such a manager's containers can only hold one version data image (though multiple versions that have identical data; for example, version 0 of a branch and the branch's branch point- can share the container.
0	MGR_COMPARE_NODIFFS	Return value for compare operation, indicating that the comparison was successful and there were no differences.
1	MGR_COMPARE_DIFF_OR_ERROR	Return value for compare operation, indicating that the comparison was not successful or that the comparison was successful but there were differences.
0	MGR_MERGE_OK	Return value for merge/xmerge operation, indicating that the merger was completed successfully.
1	MGR_MERGE_ABORT	Return value for merge/xmerge operation, indicating that the merger was aborted due to conflicts.

From the table, some return values are identical for several constants. This is because you cannot execute multiple operations at the same time and so the returned value can be reused. As an example both MGR_COMPARE_NODIFFS, MGR_MERGE_OK and MGR_OK have a return value of 0. The two last one are identical (returned when the operation is successful for a **merge**) the first one is the return value of the compare operation. As you cannot perform both a merge and compare at the same time, the value is identical but does not mean the same thing (It is reused.).

Operation description and arguments

When the type manager is called, it receives a set of arguments from ClearCase that respect the following rules:

- The first one will be the operation type.
- All following ones will be operation arguments.
- Number of argument can be fixed or variable depending on the operation requested.
- All arguments are represented as ASCII strings.

The following is a description of all operations and arguments. This is the same list as found in the \lib\mgrs\mgr_info.h files.

- The description explains what the operation should do, this is what has to be written and handled by the type manager.
- The command line is listing what information the type manager can expect from ClearCase.
- The argument list explains the command line options.
- The result is what the type manager should return as exit value.

create_element

Description

- Record the creation of a new element, its main branch and initial version.
- Store data for an empty initial version of the element (that is, /main/0).

Command line

```
create_element  create_time  element_oid  initial_branch_oid
zero_ver_oid  new_container_pname
```

Arguments

create_time	Date/time when element created, in seconds, since January 1, 1970, formatted as hex string.
element_oid	Object id of future element
initial_branch_oid	Object id of future initial branch of element
zero_ver_oid	Object id of future version 0 on initial branch.
new_container_pname	Pathname of container in which to store initial empty version

Result

MGR_STORE_KEEP_JUST_NEW	Version was successfully stored in new container
MGR_FAILED	Manager operation failed

create_branch

Description

- Record the creation of a new branch and version zero on that branch.
- The data for version zero is identical to the data for the predecessor version.
- Add the new branch and version zero to source container, or reuse the predecessor's container.
- See the construct_version operation description for requirements regarding exact preservation of version data.

Command line

```
create_branch  create_time  new_branch_oid  zero_ver_oid
new_container_pname  pred_branch_oid  pred_ver_oid pred_ver_num
pred_container_pname
```

Arguments

create_time	Date/time when branch created, in seconds, since January 1, 1970, formatted as hex string
new_branch_oid	Object id of future branch
zero_ver_oid	Object id of future version 0 on new branch
new_container_pname	Pathname of container in which to add new branch and version 0
pred_branch_oid	Object id of branch of predecessor version
pred_ver_oid	Object id of predecessor version from which new branch sprouts
pred_ver_num	Version number of predecessor version, formatted as decimal string
pred_container_pname	Pathname of predecessor version's storage container

Result

MGR_STORE_KEEP_BOTH	Both the old and new containers should be kept. The new version should use the new container. Old versions should continue to use the old container.
MGR_STORE_KEEP_JUST_NEW	The old container should be discarded and the new container should be used for this version and any other versions that used the old container.
MGR_STORE_KEEP_JUST_OLD	The old container should be saved but the new container is not needed. The new version should use the old container.
MGR_FAILED	Manager operation failed.

create_version & create_version64

Description

- Store the data for a new version.
- Store the version's data in the supplied new container, combining it with the predecessor's data if desired (such as for incremental deltas).

Warning: In some situations `create_version` is called with a `branch_oid` of a branch of which the manager has not yet been informed. In this case the manager should be prepared to create that branch as needed. Furthermore, when this occurs it is legal for the new version to be version 0 on the branch (new_ver_num == 0).

See the `construct_version` operation description for requirements regarding exact preservation of version data.

Note: The `create_version64` must be able to deal with files larger than 2GB.

Command line

```
create_version create_time branch_oid new_ver_oid
new_ver_num new_container_pname pred_branch_oid pred_ver_oid
pred_ver_num pred_container_pname data_pname
```

Arguments

`create_time`	Date/time when version created, in seconds, since January 1, 1970, formatted as hex string
`branch_oid`	Object id of branch where new version will be added
`new_ver_oid`	Object id of future version
`new_ver_num`	Version number of new version, formatted as decimal string
`new_container_pname`	Pathname of container in which to store new version's data
`pred_branch_oid`	Object id of branch of predecessor version
`pred_ver_oid`	Object id of predecessor version
`pred_ver_num`	Version number of predecessor version, formatted as decimal string
`pred_container_pname`	Pathname of predecessor version's storage container
`data_pname`	Pathname of data for new version

Result

`MGR_STORE_KEEP_BOTH`	Both the old and new containers should be kept. The new version should use the new container. Old versions should continue to use the old container.
`MGR_STORE_KEEP_JUST_NEW`	The old container should be discarded and the new container should be used for this version and any other versions that used the old container
`MGR_STORE_FILES_ARE_IDENT`	Value to OR into above result if data for version stored is identical to data of predecessor version (optional).
`MGR_FAILED`	Manager operation failed.

construct_version

Description

- Fetch the data for a version.
- Extract the data for the requested version into the supplied pathname, or return a value indicating that the source container can be used as the cleartext data for the version.

Note: The type manager must preserve the version data provided to it on the `create_version` operation (and the implicit version data for a create_branch operation) such that the `construct_version` operation yields exactly the same version data that was originally provided.

Command line

```
construct_version source_container_pname data_pname version_oid
```

Arguments

`source_container_pname`	Pathname of source container for the version.
`data_pname`	Pathname in which to store extracted data for version.
`version_oid`	Object id of version to fetch.

Result

`MGR_CONSTRUCT_USE_SRC_CONTAINER`	Use source container as data instead of data_pname.
`MGR_CONSTRUCT_USE_NEW_FILE`	Data was successfully fetched to data_pname.
`MGR_FAILED`	Manager operation failed.

delete_branches_versions

Description

- Delete one or more branches and/or versions all from the same container.
- Delete each branch and its corresponding version zero, and delete each version.
- Store the resulting data in the supplied source container, or if no data remains after the deletions, return a result indicating that the old container can be discarded and the new container is not needed.

Managers are REQUIRED to actually delete versions that have no living descendents (that is, to prune the version tree). This is to ensure that the manager will behave correctly following certain recovery actions. Specifically, under those conditions, the manager can later be presented with a `create_version` operation for a version number that the manager had previously been told to create and delete. The version's oid and data may be different than in the original instance.

This apparent ”reuse” of the version number can only occur for deleted versions with no living descendents. Managers are allowed to merely tag delete versions that do not meet the ”no living descendents” criteria. Such managers are REQUIRED to meet the initial requirement following *any* `delete_branches_versions` operation. That is, previously tag-deleted versions that no longer have any living descendents must be deleted (regardless of whether or not the current `delete_branches_versions` operation affects any of those versions).

The manager must be prepared to be asked to delete branch and/or version oids that it may not know about. It must silently accept these (the manager will meet the requirement that the branch / version does not exist in the container).

Command line

```
delete_branches_versions old_container_pname
new_container_pname [-branches branch_oid ...] [-versions
ver_oid ...]
```

Arguments

`old_container_pname`	Pathname of source container that currently contains the data for the branches/versions
`new_container_pname`	Pathname of new source container in which to store data without the deleted branches/versions
`-branches branch_oid ...`	List of object ids of branches to delete
`-versions ver_oid ...`	List of object ids of versions (including zero) to delete

Result

`MGR_DELETE_KEEP_JUST_NEW`	The old container should be discarded and the new container should be used for all remaining versions that used the old container before the deletion
`MGR_DELETE_KEEP_NEITHER`	The old container can be discarded and the new container is not needed

MGR_DELETE_KEEP_JUST_OLD	The old container should be used for all remaining versions, and the new container is not needed. (This result is not recommended. It should be used only for managers who cannot delete the data from the container. The branches/versions will be deleted from the database without ever reclaiming the space in the source container.)
MGR_FAILED	Manager operation failed

compare

Description

- Compare the data for two or more versions.
- For more information, see cleartool diff.

Command line

```
compare [-tiny | -window] [-serial | -diff | -parallel] [-columns n]
      [-directory] [-blank_ignore] [pass-through-options]
      [-fname string ...] pname pname ...
```

Arguments

`[-tiny | -window] [-columns n]`	See corresponding cleartool diff options
`-serial`	See cleartool diff -serial_format
`-diff`	See cleartool diff -diff_format
`-parallel`	Requests side-by-side display of differences (such as cleartool diff default).
`-directory`	Indicates that contributors are VOB directory versions represented by encoded files in format described under "Directory Diff/Merge" above
`-blank_ignore`	See cleartool diff -blank_ignore
`pass-through-options`	Options specific to this type manager
`-fname string ...`	String to display for an input pathname; this option can be repeated up to the number of input pathnames supplied the compare program should use string[n] as a label for pname[n], if string[n] was supplied
`pname pname ...`	List of pathnames of cleartext data of versions to compare

Result

`MGR_COMPARE_NODIFFS`	The comparison was successful and there were no differences
`MGR_COMPARE_DIFF_OR_ERROR`	The comparison was not successful, or the comparison was successful but there were differences

xcompare

Description

- Compare the data for two or more versions, using a graphical display.
- For more information, see `cleartool xdiff`.

Command line

```
xcompare [-hstack | -vstack] [-tiny] [-directory] [-blank_ignore]
      [pass-through-options] [-fname string ...] pname pname ...
```

Arguments

`[-hstack | -vstack] [-tiny]`	See corresponding xcleartool xdiff option
`-directory`	Indicates that contributors are VOB directory versions represented by encoded files in format described under "Directory Diff/Merge" above
`-blank_ignore`	See cleartool diff -blank_ignore
`pass-through-options`	Options specific to this type manager
`-fname string ...`	String to display for an input pathname; this option can be repeated up to the number of input pathnames supplied; the xcompare program should use string[n] as a label for pname[n], if string[n] was supplied
`pname pname ...`	List of pathnames of cleartext data of versions to compare

Result

`MGR_COMPARE_NODIFFS`	The comparison was successful and there were no differences.
`MGR_COMPARE_DIFF_OR_ERROR`	The comparison was not successful, or the comparison was successful but there were differences.

merge

Description

- Merge the data for two or more versions.
- For more information, see `cleartool merge`.

When selective query is used (`-query switch`) it is assumed that the last pname listed is the to-version, unless -to is used, in which case the pname given with -to is the to-version.

Command line

```
merge [-qall | -abort] [-base base_pname] -out output_pname
      [-tiny | -window] [-serial | -diff | -parallel] [-columns
      n]
      [-directory] [pass-through-options]
      [-fname string ...] pname pname ...
```

Arguments

`[-qall	-abort] [-tiny	-window] [-columns n]`	See corresponding cleartool merge options
`-serial`	See cleartool merge -serial_format
`-diff`	See cleartool merge -diff_format
`-parallel`	Requests side-by-side display of differences (e.g. cleartool merge default)
`-base base_pname`	Pathname of cleartext data for base version, if any
`-out output_pname`	Pathname where cleartext data for result of merge should be stored
`-directory`	Indicates that contributors are VOB directory versions represented by encoded files in format described under "Directory Diff/Merge" above
`pass-through-options`	Options specific to this type manager
`-fname string ...`	String to display for an input pathname; this option can be repeated up to the number of input pathnames supplied, including the base_pname (which is considered to be the first pathname); the merge program should use string[n] as a label for pname[n], if string[n] was supplied
`pname pname ...`	List of pathnames of cleartext data of versions contributing to merge

Result

`MGR_MERGE_OK`	The merger was completed successfully.
`MGR_MERGE_ABORT`	The merger was aborted due to conflicts.
`MGR_FAILED`	Manager operation failed.

xmerge

Description

- Merge the data for two or more versions, using a graphical display.
- For more information, see `cleartool xmerge`.

Command line

```
merge [-qall | -abort] [-base base_pname] -out output_pname
      [-hstack | -vstack] [-tiny] [-directory] [pass-through-
      options]
      [-fname string ...] pname pname ...
```

Arguments

`[-qall	-abort] [-hstack	-vstack] [-tiny]`	See corresponding cleartool xmerge options
`-base base_pname`	Pathname of cleartext data for base version, if any
`-out output_pname`	Pathname where cleartext data for result of merge should be stored
`-directory`	Indicates that contributors are VOB directory versions represented by encoded files in format described under "Directory Diff/Merge" above
`pass-through-options`	Options specific to this type manager
`-fname string ...`	String to display for an input pathname; this option can be repeated up to the number of input pathnames supplied, including the base_pname (which is considered to be the first pathname); the xmerge program should use string[n] as a label for pname[n], if string[n] was supplied
`pname pname ...`	List of pathnames of cleartext data for versions contributing to merge

Result

`MGR_MERGE_OK`	The merger was completed successfully.
`MGR_MERGE_ABORT`	The merger was aborted due to conflicts.
`MGR_FAILED`	Manager operation failed.

context_merge

Description

- Post-process the group of files that were merged or xmerged by this type manager in the current "merge session".
- This method is optional, and only a subset of ClearCase merge operations will invoke it. Currently this is limited to UCM deliver and rebase.

If ClearCase intends to invoke this method following a merge or xmerge, the environment variable `CCASE_WILL_CONTEXT_MERGE` will be set during those methods.

Command line

```
context_merge [-abort] pname
```

Arguments

`-abort`	Cancels the command instead of engaging in a user interaction; a merge takes place only if it is completely automatic.
`pname`	Pathname of a temporary file that itself contains a list of pathnames, each of which was successfully processed by the [x]merge method of this type manager (or would have been had it not been a trivial merge), and remains checked out. One filename is listed per line. Since each pathname will refer to a checked-out file in the view's file area (in the MVFS for dynamic views and in the copy area for snapshot views), ClearCase commands such as properties or history may be invoked using that pathname.

Result

`MGR_MERGE_OK`	The merger was completed successfully.
`MGR_MERGE_ABORT`	The merger was aborted due to conflicts.
`MGR_FAILED`	Manager operation failed.

annotate

Description

- Fetch the annotated data for a version.
- Extract the annotated data for the requested version from the indicated source container into the supplied output pathname.
- For more information, see `cleartool annotate`.

Default: Fetch annotated data for requested version only and indicate that all lines are INCLUDED.

Command line

```
annotate [-all] source_container_pname output_pname version_oid
```

Arguments

`-all`	Fetch annotated data for all versions. Indicate for each line whether it is still INCLUDED in the requested version, was included in an ancestor of the version but was DELETED, or is UNRELATED to the version.
`source_container_pname`	Pathname of source container for the version.
`output_pname`	Pathname in which to store annotated data for version.
`version_oid`	Object id of version. If not -all, fetch and annotate data for this version. If -all, fetch data for all versions of element and annotate them with respect to this version.

Result

`MGR_OK`	Data was successfully fetched to output_pname.
`MGR_FAILED`	Manager operation failed.

Output file format:

Each line of the output file has the format:

code version-oid[,deleted-version-oid] line-of-data

where:

`code`	Indicates the relationship of this line to the version fetched (a mgr_annotate_code_t value)
`version-oid`	Object id of the version that first included the line of data
`deleted-version-oid`	If line has been deleted, object id of the version that deleted the data
`line-of-data`	Line of data

get_cont_info

Description

- Fetch the information describing the contents of a container.
- Extract the version, etc information from the source container into the supplied output pathname.

Command line

```
get_cont_info source_container_pname output_pname
```

Arguments

source_container_pname	Pathname of source container for the version
output_pname	Pathname in which to store container description information

Result

MGR_OK	The manager maintains version identification information and that data was successfully fetched to output_pname. The output file must have at least one MGR_INFO_VERSION record in it.
MGR_FAILED	Manager operation failed.
MGR_INFO_NO_INFO	The manager does not maintain version identify information. This response is only legal for container-per-version style managers (also known as 'whole copy' style managers). Such managers can still legally share a container between a branch's predecessor version (its sprout point) and the branch's branch-version-0 version. (Note: a result of MGR_OK with an empty output file is considered an error).

Output file format:

- An unordered set of version, branch and element identification records, with a single information record per line.
- Only the version info records are mandatory. See the previous descriptions for the information record definitions:

```
MGR_INFO_VERSION
MGR_INFO_BRANCH
MGR_INFO_ELEMENT
```

Customizing the Type Manager

As seen until now, the type manger is more than just a few modifications or settings. A full type manager involves coding and lot of testing. The type manager is a critical part of the ClearCase application as it is the one responsible for your data. It is always a better idea to use the provided type manager instead of creating a new one. The ones provided with ClearCase are supported and tested.

Identify the need

The first question would be, "What is really needed?"

- Do you just want to identify and handle a set of identical elements?
- Do you need to replace or change only one specific part or one single operation?
- Do you need to handle a very specific file format that none of the existing type managers can?

Element type creation and use

This is the more simple solution. This can be used if you want to "see" a set of file with a different type and if the format can be handled by an existing provided type manager.
An example would be a set of files in a VOB, some are relating to C source code, others are documentation or design notes. What if one day you need to find all the files that are related to design and they are not all present in a single place? One solution is to assign them an attribute or to assign them one element type. Just define a new element type and assign it to the desired element.

If you want to automate the type assignment to a newly created element, you will need to modify the magic file. Remember that the magic file is local to each client. This means all modification must be done on ALL clients of your site, and if you use MultiSite this must also be done on all replicated sites.

The following example defines an element type named '`design_file`' and assigns it to all desired files. The only requirement is that all files must use the same supertype (so all will be text files or all binary files, but not a mix of them).

First a new element type needs to be created:

```
cleartool> mkeltype -supertype text_file –nc design_file
Created element type "design_file".

cleartool> describe -long eltype:design_file
element type "design_file"
  created 11-Mar-05.13:48:50 by Marechal Laurent (HOST1)
  owner: IBM \nlxxxx
  group: IBM \yyyy
  scope: this VOB (ordinary type)
  type manager: text_file_delta (inherited from type
"text_file")
  supertype: text_file
  meta-type of element: file element
```

This creates a new type called '`design_file`', it will use the same type manager as the `text_file` one and will have the same limitation (no binary char, no line greater than 8000 char).

For existing elements, the current element type needs to be changed in this manner:

```
>cleartool chtype design_file catalog.dsg
```

```
Change version manager and reconstruct all versions for
"catalog.dsg"?  [no] yes
Changed type of element "catalog.dsg" to "design_file".

>cleartool describe -long catalog.dsg
version "catalog.dsg@@\main\1"
  created 08-Mar-05.16:10:47 by Marechal Laurent (HOST1)
  Element Protection:
    User : IBM \nlxxxx : r--
    Group: IBM \yyyy : r--
    Other:           : r--
  element type: design_file
predecessor version: \main\0
```

For new element, you can either create the element as usual and then assign the new type, or use the magic file from ClearCase in order to use your type at each element creation.

To use the magic file, you will need to edit and change the file on ALL clients and the server. If this is not done, then the element will be added to source control but not with your customized type.
The following line needs to be added at the beginning of the file (after the '`core`' line):

```
design_file text_file : -name "*.[dD][sS][gG]";
```

After adding this line, element creation will get the desired type:

```
>cleartool mkelem -nc reviewed.dsg
Created element "reviewed.dsg" (type "design_file").
Checked out "reviewed.dsg" from version "\main\0".
```

When done, you can use the new type in all ClearCase commands where query can be used:

```
>cleartool find . -all -elem (eltype(design_file)) -print
\TypeManagerWP\catalog.dsg@@
\TypeManagerWP\reviewed.dsg@@

>cleartool find . -elem (eltype(design_file)) -ver
attype(REVIEW)
  -exec "cleartool describe -fmt \"%n   %[REVIEW]a \n \"
%CLEARCASE_XPN%"
.\catalog.dsg@@\main\1   (REVIEW="NO")
.\publish.dsg@@\main\1   (REVIEW="YES")
.\reviewed.dsg@@\main\GRAM\1   (REVIEW="IN PROGRESS")

>cleartool> find . -elem (eltype(design_file)) -ver
'{REVIEW=="YES"}'
  -print
.\publish.dsg@@\main\1

>cleartool> find . -elem (eltype(design_file)) -ver
'{REVIEW=="YES"}'
  -exec 'cleartool mklabel -nc REVIEWED "%CLEARCASE_XPN%"'
Created label "REVIEWED" on ".\publish.dsg" version "\main\1".
```

If the VOB holding this new element is replicated using MultiSite, you must perform the same step on all sites for all replica of this VOB.

Change existing type manager

You would change your existing type manager if you want to keep most of the normal operations (checkin/checkout) but want to change only one (such as merge) with another executable. This can be done by modifying the map file and by adding a wrapper to the tool to present the argument in the right order. Note that the executable that will be used to replace the provided ClearCase type manager MUST comply with the ClearCase type manager argument list. If not, do not expect it to work. You will have to order the argument and result to let ClearCase think it is a valid type manager. You will also have to update the map file and copy the executable on ALL machines (client and server) and on all sites where the VOB is replicated. This, of course, is **not** supported. Do **not** change the mapping of an existing element type. Always create a new element type if you need to change an operation. If not, some ClearCase operations relying on the provided official type manager will fail.

The following is a quick example of how to change a type manager in order to diff a Microsoft Excel file. Note that this example is not complete and was not tested in all situations. It also only performs the **DIFF** operation and not the merge one.

The following example uses WDD.exe. This diff tool is part of the Softinterface, Inc, product (for more information see http://www.softinterface.com). It is possible to use WDD.exe as compare/diff tools for Excel files.

The following versions were used:
WDD.exe version 2.36
ClearCase version 2003.06.00 (Fri Apr 18 13:06:18 2003)

1. Install WDD and copy the WDD.exe application to the ClearCase \bin directory.

2. Create an Excel file in a dynamic view (for testing purpose)

3. Create several different versions of this file

4. Test the WDD compare. A manual compare with WDD is working; however the parameter must be passed in a certain order that will NOT be the one of the type manager by default. The following command works for comparing two versions:

   ```
   wdd.exe /M spread.xls /S spread.xls@@\main\1 /L
   compare.log
   ```

   Note that you have to specify both versions. The type manager used is the one responsible for handling the correct set of parameters (mainly the file/path name as well as against which one you compare as specified in this document). The IBM Rational compare tool understands the switch `-previous` and so is able to get the previous version alone. WDD does not accomplish this and you are required to pass the other version yourself. The version must be passed in an extended pathname syntax. This means it will only work in dynamic views. Another way would be to add a wrapper that would handle the different case and get from the VOB the two versions in a temp directory and run the diff on these two files.

5. When the manual compare is working and you have identified and fixed the parameters, you will have to update the type manager tools. The following has been added to the map file:

   ```
   _ms_excel_file construct_version ..\..\bin\zmgr.exe
   _ms_excel_file create_branch ..\..\bin\zmgr.exe
   ```

```
_ms_excel_file create_element ..\..\bin\zmgr.exe
_ms_excel_file create_version ..\..\bin\zmgr.exe
_ms_excel_file delete_branches_versions
..\..\bin\zmgr.exe
_ms_excel_file compare ..\..\bin\wdd.exe
_ms_excel_file xcompare ..\..\bin\wdd.exe
_ms_excel_file merge ..\..\bin\zmgr.exe
_ms_excel_file xmerge ..\..\bin\zmgr.exe
_ms_excel_file get_cont_info ..\..\bin\zmgr.exe
```

This instructs the file type `ms_excel` to use wdd.exe for comparing file for both GUI and command line compare.

6. Create a new element type for use with it:

```
cleartool mkeltype -nc -supertype file -manager
_ms_excel_file _ms_excel_file
Created element type "_ms_excel_file".

cleartool describe -long eltype:_ms_excel_file
element type "_ms_excel_file"
created 11-Aug-03.13:02:38 by lmarecha@host1
owner: RATIONAL\lmarecha
group: RATIONAL\ccuser
scope: this VOB (ordinary type)
type manager: _ms_excel_file
supertype: file
meta-type of element: file element
```

7. Change the existing file type to the new element:

```
cleartool chtype _ms_excel_file spread.xls
Change version manager and reconstruct all versions for
"spread.xls"? [no] yes
Changed type of element "spread.xls" to "_ms_excel_file".
```

8. Comparing the file using '`compare with previous version`' while the file is checked-in result in an error. WDD is started but fails as the file is read-only. This is not an issue with ClearCase but with WDD.exe that tries to open the file in write when it just has to read it. Just ignore the error and the compare will work.

9. Comparing files using '`compare with previous version`' while the file is checked-out will succeed.

10. Comparing from the version tree between two different versions (even not the one selected by the view) is also working.
    Example: View is selecting version 4. Click on version 3, right-click to get the context menu, select compare/with another version. Click on version 2. WDD is started with the correct version to compare, and no error is reported.

11. In order to use that on different machines you need to perform the following steps on each machine:
    - Install WDD
    - Update the map file
    - Change the element type to use the new type

- Set the new element to use the new type by changing the magic file, as desired

Note 1: The log file is not generated. You cannot pass personal parameters to WDD. The two filenames are passed from ClearCase to the type manger. If you want to pass any personal parameters you will need to create a wrapper around WDD that will add them.

Note 2: The parameters are passed to the type manager; you cannot change them or personalize them if you use the default type manager. The only way would be to create a brand new type manager, or to create a wrapper to reorder them.

Full type manager change

This is the same as changing an existing type manager, but in this case you are changing the whole type manager to handle a specific file format. It will be necessary to be able to handle all of the common type manager operations. This is the more difficult implementation. You will have to create an element type, update the map file and write the whole type manager that must comply with all operations, arguments, exit values, calls.

It also will be necessary to update the map file and copy the executable on ALL machines (client and server) and on all sites where the VOB is replicated.

This dummy type manager below is just to show a simple example. It is not supported. It does not perform any action , just outputs to the screen the arguments it got when called; it should be safe to use on a TEST machine.

The full type manager is set up the same way as when you change an existing one. The additional step is that YOU have to write the full handling of all operations. This can be done at one time (all operation in one single file) or split across several files. You can also only implement a specific operation and keep using the other one from the element type supertype. This is common if you just want to replace one operation.

Here is the dummy type manager code written in C (Microsoft Visual Studio® .NET with wizard for console project).

```
// TM_Dump.cpp : Defines the entry point for the console
application.
// Microsoft Visual C++ 2005 (Express) Marechal Laurent
IBM/Rational

#include <stdafx.h>
#include <string.h>
#include <stdlib.h>
#include "C:\Program
Files\Rational\ClearCase\lib\mgrs\mgr_info.h"

#define SHOWARG {for(i=0;i<argc;i++){printf("argc %i:
%s\n",i,argv[i]);}}

int _tmain(int argc, char* argv[])
{
int i;
```

```
if (!strcmp(argv[1],MGR_OP_COMPARE))
{
      SHOWARG
      exit(MGR_FAILED);
}
if (!strcmp(argv[1],MGR_OP_CREATE_ELEMENT))
{
      SHOWARG
      exit(MGR_FAILED);
}
if (!strcmp(argv[1],MGR_OP_CREATE_VERSION))
{
      SHOWARG
      exit(MGR_FAILED);
}
      if (!strcmp(argv[1],MGR_OP_CREATE_VERSION64))
{
      SHOWARG
      exit(MGR_FAILED);
}
if (!strcmp(argv[1],MGR_OP_CREATE_BRANCH))
{
      SHOWARG
      exit(MGR_FAILED);
}
if (!strcmp(argv[1],MGR_OP_CONSTRUCT_VERSION))
{
      SHOWARG
      exit(MGR_FAILED);
}
if (!strcmp(argv[1],MGR_OP_XMERGE))
{
      SHOWARG
      exit(MGR_FAILED);
}
if (!strcmp(argv[1],MGR_OP_DELETE_BRANCHES_VERSIONS))
{
      SHOWARG
      exit(MGR_FAILED);
}
if (!strcmp(argv[1],MGR_OP_CONTEXT_MERGE))
{
      SHOWARG
      exit(MGR_FAILED);
}
if (!strcmp(argv[1],MGR_OP_GET_CONT_INFO))
{
      SHOWARG
      exit(MGR_FAILED);
}
if (!strcmp(argv[1],MGR_OP_ANNOTATE))
{
      SHOWARG
      exit(MGR_FAILED);
}
if (!strcmp(argv[1],MGR_OP_XCOMPARE))
```

```
{
        SHOWARG
        exit(MGR_FAILED);
}
if (!strcmp(argv[1],MGR_OP_MERGE))
{
        SHOWARG
        exit(MGR_FAILED);
}

return MGR_FAILED;
}
```

Note: The code includes the skeleton to all operations that could be handled. Refer to the previous section or the `mgr_info.h` for more information. For each ”operation”, you must show the current arguments received when called, and exit with FAILED status.

Compile it to an executable.
Then copy the executable to the lib\mgrs directory of your ClearCase installation.

Edit the map file to add:

```
_custom          construct_version       ..\..\bin\zmgr.exe
_custom          create_branch           ..\..\bin\zmgr.exe
_custom          create_element          ..\..\bin\zmgr.exe
_custom          create_version          ..\..\bin\zmgr.exe
_custom          delete_branches_versions
..\..\bin\zmgr.exe
_custom          compare
..\..\bin\cleardiff.exe
_custom          xcompare
..\..\bin\worddiffmrg.exe
_custom          merge
..\..\bin\cleardiff.exe
_custom          xmerge
..\..\bin\worddiffmrg.exe
_custom          get_cont_info           ..\..\bin\zmgr.exe
```

Create the element type in the VOB

```
cleartool mkeltype -nc -supertype file -manager _custom _custom
```

Replace the compare line in the map file with the custom one for the desired operation

```
_custom          compare                 .\TM_dump.exe
```

Change the element type to the new one

```
cleartool chtype _custom <file>
```

Test it

```
cleartool diff -pre <file>
```

If installed smoothly, it will only output the parameters and do nothing to the data (the last two errors are normal):

```
>cleartool diff -pred toto.txt
argc 0 : compare
argc 1 : M:\General_view\TypeManagerWP\toto.txt@@\main\1
argc 2 : toto.txt

>cleartool mkelem -nc -eltype _custom newfile.dta
argc 0 : create_element
argc 1 : 42382a31
argc 2 : d88ee7aa.7f414a6b.9d52.86:3b:bc:02:46:d0
argc 3 : 6b5b1b94.addd4824.a689.e3:5d:b9:17:37:60
argc 4 : e836f039.987344bc.8fa1.bb:c7:e8:dd:c2:2a
argc 5 :
C:\CCSHARE\VOB\TypeManagerWP.vbs\s\sdft\1a\34\tmp_4724.1
cleartool: Error: Unexpected value (0) returned by type manager
"_custom"
cleartool: Error: Unable to create element "newfile.dta".
```

Practical example

UTF16 handling (diff/merge tool with Beyond Compare)

Goal

UTF16 files (Unicode) actually are not supported by ClearCase 7.0.0.0 out-of-the-box for diff/merge. This kind of file can be put under source control and used, but there are no ways to diff/merge them with the provided tool. The graphical diff and merge functions need to be replaced as do the command line ones.

Analyze

We will only see how to replace the GUI tools, replacing the command line ones would be equivalent as long as you find a command line tool to perform the same operations (merge, compare operations). As we only need to replace the graphical functions, we only need to redirect the xcompare and xmerge to the new tool. All others can use the ClearCase originals ones.

Procedure

This will be very near the Excel example. This is done with the ClearCase 7.0.0.0-IFIX02, Beyond Compare V2.2.7 and Windows XP SP2.

1- Install the Beyond Compare tools.

2- Create a file with several versions (or use a test file) to test.

3- Test the compare operation manually.

```
BC2.exe /noedit /readonly /fv /silent <file1> <file2>
```

Note that you have to specify both versions. The type manager used is the one responsible for handling the correct set of parameters (mainly the file/path name as well as against which one you compare as specified in this document).

4- When the manual compare is working and you have identified and fixed the parameters, you will have to update the type manager tools. The following as been added to the map file in order to create a new element type for this purpose:

```
_bc_file construct_version ..\..\bin\zmgr.exe
_bc_file create_branch ..\..\bin\zmgr.exe
_bc_file create_element ..\..\bin\zmgr.exe
_bc_file create_version ..\..\bin\zmgr.exe
_bc_file delete_branches_versions ..\..\bin\zmgr.exe
_bc_file compare ..\..\bin\cleardiff.exe
_bc_file xcompare ..\..\..\..\Beyond Compare 2\BC2.exe
_bc_file merge ..\..\bin\cleardiff.exe
_bc_file xmerge ..\..\..\..\Beyond Compare 2\BC2.exe
_bc_file get_cont_info ..\..\bin\zmgr.exe
```

This instructs the file type _bc_file to use BC2.exe for comparing and merging file for GUI only. Command line operation will still be handled by ClearCase.

5- Create a new element type for use with it:

```
>cleartool mkeltype -nc -supertype file -manager _bc_file
_bc_file
Created element type "_bc_file".

>cleartool describe -long eltype:_bc_file
element type "_bc_file"
  created 2006-12-05T17:14:06+01 by jdoe.grp@host2
  owner: DOM1\jdoe
  group: DOM1\grp
  scope: this VOB (ordinary type)
  type manager: _bc_file
  supertype: file
  meta-type of element: file element
```

6- Change the existing file type to the new element:

```
>cleartool chtype _bc_file toto.txt
Change version manager and reconstruct all versions for
"toto.txt"?  [no] y
Changed type of element "toto.txt" to "_bc_file".
```

7- Try the compare operation from the command line first:

```
cleartool diff -graphical toto.txt@@\main\3 toto.txt@@\main\2
```

This should start-up the Beyond Compare GUI with your two files.

8- Try the compare operation from the version tree, by using ‘compare with other…’; it should launch the GUI as well.

If nothing happens, make sure to try from the command line; more information is returned and can help to find the root cause (as these errors are not logged).

Example:

```
>cleartool diff -graphical toto.txt@@\main\3 toto.txt@@\main\2
cleartool: Error: Operation "xcompare" unavailable for manager
"_bc_file"
        (Operation pathname was: "C:\Program
Files\Rational\ClearCase\lib\mgrs\"C:\Program Files\Beyond
Compare 2\BC2.exe"")
```

9- If you need some specific options or need to reorder the parameters, then you have an issue. The map file cannot handle options, only a pathname. This means that you cannot define a call like:

```
_bc_file xcompare ..\..\..\..\Beyond Compare 2\BC2.exe
/readonly /fv /noedit /silent
```

There are two solutions for this:

- Write your own type manager that will handle the options and arguments
- Use a wrapper to add what is needed

In this example, the wrapper is chosen. Because the Windows edition of ClearCase can only use

executables (.exe, .com) as target for the type manager, you would need a wrapper in .exe form to reorder and handle the parameters. You can either write your own, or use the third-party tool from Scooter Software (the owner of Beyond Compare).

You can get it from their website http://www.scootersoftware.com/support.php?c=kb_vcs.php

- Download their `cleardiffbc.zip` archive and extract it. Make sure to read the `readme.txt` file.
- Copy `ClearDiffBc.exe` into the bin directory of the ClearCase installation.
- Update your map file with the `cleardeiffbc.exe` target for the desired operations
  `_bc_file xcompare ..\..\bin\cleardiffBC.exe`

10- In order to use the new diff/merge feature on different machines you need to perform the following steps on each of them:

- Install Beyond Compare and the cleardiffbc.exe
- Update the map file and magic file

If you have an existing element, the element type must be changed to use the new type.

Notes and warning

- All examples provided here are for instruction only and should not be used in production unless they are well tested. Any modification to the product is subject to the support license.
- Custom written scripts and type managers are NOT supported by IBM Rational.
- It is not possible on Windows platforms to use anything other than .exe or .com as a target for the type manager. Change request (RFE) RATLC01005985 has been submitted to allow the use of another type.
- When using external tools, you are subject to their limitations (that is, some tool are not able to diff/merge more than two files at the same time.)

References

The following were used in references or as other sources of information:

- IBM Rational ClearCase Information Center

- IBM Rational Support Knowledge Base (technotes)
  - **About type managers and size limitations**
  - **About _xml2, _html2, and _rose2 type managers**
  - **About the ClearCase Magic file**
  - **Changing the XML Diff/Merge Type Manager**
  - **Changing unused element types in a replicated VOB**
  - **Changing used element types in a replicated VOB**
  - **Installed type manager does not support operation "get_cont_info"**
  - **Error comparing files containing lines over 8000 bytes**
  - **Operation construct_version unavailable for type manager**
  - **File too large error checking in a text file**
  - **About XML and the Windows 1252 encoding scheme**
  - **How file types are determined when creating a new element**
  - **New Merge Type Copy feature with ClearCase version 7**
  - **Handling binary files in ClearCase**
- <ATRIAHOME>\lib\mgrs\map
- <ATRIAHOME>\lib\mgrs\mgr_info.h
- http://www.scootersoftware.com/support.php?c=kb_vcs.php