Configuration Specification Generation

Douglas B. Robinson, Hewlett Packard Company

In order to obtain maximum flexibility in a ClearCase and ClearCase MultiSite environment, we have focused the complexity of the development process on the Configuration Specification (or "CS"). Flexibility is defined as the ability to encompass different branching techniques within different subtrees of the VOB hierarchy and to support different logical levels of branching. We have created a CS meta-language which describes how to transform a "Master Configuration Specification" (or "MCS") to produce CS versions for use during the different phases of development. The meta-language, the tool which implements it, and the design constraints due to the ClearCase and ClearCase MultiSite products themselves are explained.

1.0 Introduction

In the fall of 1993, we began the conversion of a large project into the ClearCase V1.1 system: 60 to 80 engineers, 100 to 200 workstations, 60,000 source files, 300Mb of derived objects per system build. As part of converting the development process, we developed a mechanism for generating all of the ClearCase configuration specification (CS) flavors that were needed by our developers. By the spring of 1994 the design and most of the implementation were done. Refinements continue to be made to this day.

Due to the size of the system, we found that some of the normal methodologies described in the ClearCase manuals would not be appropriate to use. For instance, labeling the versions used in the nightly build and test process was found to take from two to three hours during which time the network and VOB servers were under constant stress (thus builds were failing with DB time-outs). Therefore, we only label *significant* builds such as those shipped as beta, product, patch, etc.

Instead of labeling, we use the trick that:

`LATEST + time rule <=> a label`[1] **(EQ 1)**

This simple trick enables us to forego a huge amount of processing time and database space and still be just as accurate as if we had applied a label. Many of the tricks we use are beyond the scope of this paper, but we will point out as many as possible.

1. This is only true in the current timezone since the CS time rules only know about localtime. A second issue arises when using MultiSite in this manner: a forced synchronization must be done between all replicas which hold the mastership of branch types that are being used in a build in order to be assured that a future synchronization does not produce a "new" version which still qualifies as far as the time rule is concerned.

(c) 1996 Hewlett Packard Company

2.0 Guiding Principles

A number of principles guided the development of the tools in our environment:

2.1 Easy to Use

We required that the process that our engineers use to obtain appropriate CS flavors be simple and easy to follow. There should not be too many confusing options nor too many steps in the process.

On the other hand, there is a certain amount of complexity in the system, and that complexity can not be hidden from everyone. Certain engineers need to be able to tame that complexity for the vast majority.

While we strive to make it easy for the latter set of engineers, our *primary* focus was to provide the simplest process possible for the former set.

2.2 Flexibility[1]

Our product is produced from a combination of OSF's DCE source code, HP proprietary source code, 3rd party sources, etc. Each of these areas had their own branching requirements (e.g which label types and branch types are used to version elements) and therefore we found that we needed to be able to handle different branching policies at different places throughout our source base.

2.3 Lightweight

We decided early on that we did not want to be in the business of parsing the CS language in order to generate appropriate CSs. Such parsing would be complex and prone to break each time the CS language is enhanced. Whatever solution we developed would need to be lightweight enough so that we could quickly adapt it to changes in the ClearCase product.

2.4 Conform to the "Natural Order" of the ClearCase System

Given the size of our system it was clear from the outset that our solution must scale. We decided very early on that we would make use of the ClearCase system in "mainstream" ways. We would not heavily rely on features that seemed to be of high cost either in space (in the database or in the view) or time (at creation or at use).

1. Another need for flexibility was due to timing: we started this project using ClearCase V1.1. This was a time when merging directories was unsupported and therefore *difficult*. While we really wanted to have a uniform branching policy for files and directories, we also knew that we could *not* do so without better support. This led us to the policy where files were branched almost continually yet directories were branched only very rarely. With the advent of ClearCase V2.0 directory merging became supported. We immediately converted our policy so that files and directories were branched uniformly.

(c) 1996 Hewlett Packard Company

2.4.1 General Examples:

Frequent labeling proved to be expensive in both space and time (to apply). Attributes proved to be expensive in database space and time (to apply *and* to use). Queries in CSs also proved to be expensive in time (to use).

2.4.2 Branching:

Branching technology was scrutinized in terms of overall impact on the system and on the way that it affected our ability to query the system. We believe that our engineers (possibly even managers) need to be able to ask questions of the system such as “what changed” and we wanted to be able to answer such questions easily and quickly.

We identified two major branching techniques: physically hierarchical branching (PHB) and logically hierarchical branching (LHB). PHB is characterized by the requirement that an instance of branch type C, called the child, is always rooted in an instance of branch type P, called the parent. LHB is characterized by the requirement that child branch instances always merge into parent branch instances. In LHB, child branch instances can either be rooted in the parent branch instance into which they will be merged (identical to PHB above) or can be rooted at the place where the parent itself *will be* rooted when it is eventually created. Normally the parent branch instance will only be created during the merge of a child branch instance.

FIGURE 1. Branching Techniques:

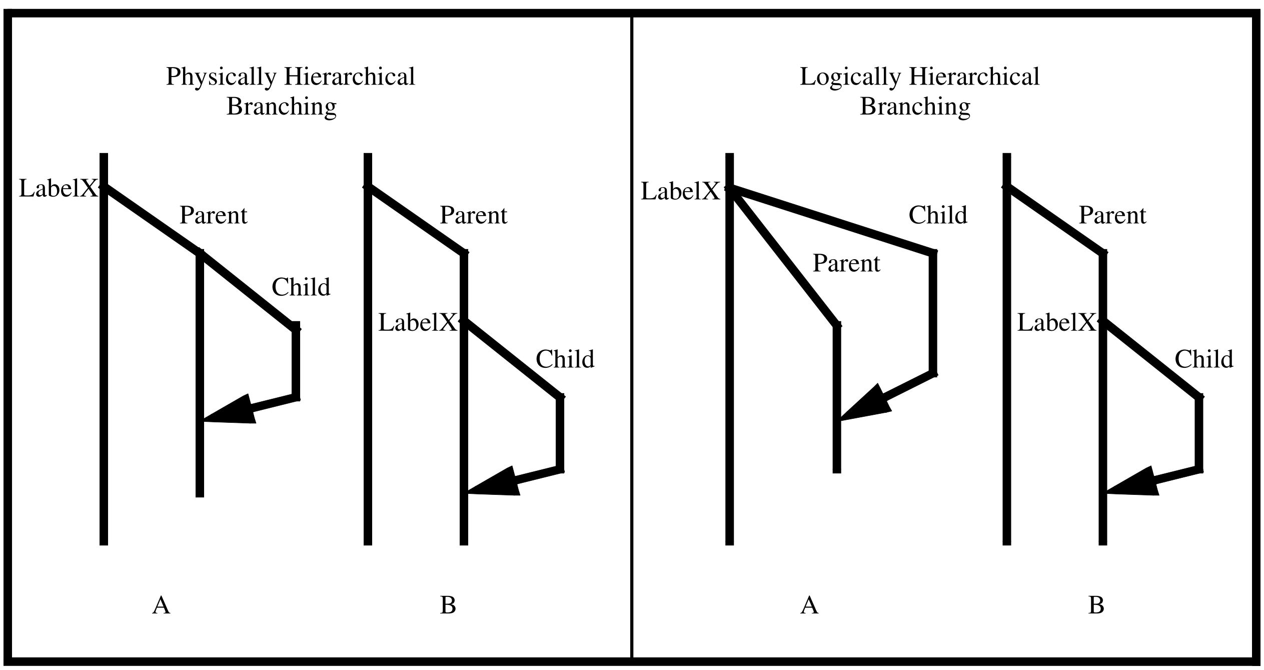

Figure 1 above shows graphically the difference between PHB and LHB.

Case A represents the branch structure that occurs when a child branch instance is created when the parent branch instance had yet to exist. For PHB, the act of creating the child branch instance requires that a parent instance be created.

Case B represents the branch structure that occurs when a child branch instance is created when the parent branch instance already exists. Case B is identical for both PHB and LHB.

PHB, while seductively alluring, was found to have far too many rough edges[1]. Thus we decided to use LHB. Immediately we benefited: the existence of a branch instance now signifies work done. One system level query is simpler to do.

Ultimately, we were very happy with choosing LHB: when MultiSite appeared it became quite clear that PHB is unsuitable for even moderately close development between sites because branch type mastership problems appear very early in the development cycle: during task branch creation. LHB enables close development between sites since the issue of branch mastership then only appears at the very end of the development cycle: at the time the task branch needs to be merged into its logical parent - exactly when they should.

3.0 A Bit More Terminology

3.1 Collection Branch

“Collection branch” refers to any branch in the hierarchy which is modified primarily by being merged to from other branches.

3.2 Task Branch

“Task branch” refers to any branch in the hierarchy which is modified primarily by editing by an engineer.

3.3 Project Branch

A “project branch” is a collection branch where work for a particular release is collected via merges. If there are different branching policies in different areas of the source base, there may be more than one project branch per project or product.

3.4 Sub-project Branch

A “sub-project branch” is a collection branch shared by two or more engineers in order to integrate their work before merging it into the project branch.

1. This topic is worth a small paper on its own: perhaps next year...

 (c) 1996 Hewlett Packard Company

4.0 Requirements

The following are the requirements for the CS generation system:

4.1 Handle All Normal Development Phases

We work in a pretty standard model of short, repetitive cycles of Isolation and Integration.

A known "good" system is used to begin the isolation phase in which an engineer makes a change to the system so that the engineer can be confident that any problems introduced into the product by their changes are due to those changes. During the isolation phase of development the engineer is said to work using a "task oriented" CS.

When an engineer is confident that their changes are correct, they switch into the integration phase. During the integration phase of development the engineer is said to work using a "merge oriented" CS.

4.2 Must Not Modify the ClearCase CS Language

We wanted the Master Configuration Specification files (MCSs) to be "settable" on their own. We chose that the MCSs be written in such a way that they provide a "read-only" CS if set. In this way we provided one of the normal CS types that need to be used during development with no additional processing.

In the future there will be one exception made to that requirement: we will be adding a timezone to all time rules used so that we can replicate the MCSs around the world via ClearCase MultiSite as part of the normal process. The CS generation tool, `createcs`, will then be made to adjust the specified time with the offset of the current timezone from the specified timezone.

5.0 The Meta Language

The language described in this section grew up over time. As such, the language has some areas that should be revisited. The intent of this document is to describe the actual language currently in use in order to provide a basis for further work.

5.1 Global Rules

The CS meta language statements are embedded in CS comment lines in the MCS.

Nearly all CS version rules ("`element...`") in the MCS are normally modified with the "`-nocheckout`" clause. This is important for two reasons. First, the MCS itself is meant to be settable and is used for *read-only* access to the source base: `createcs` knows this and will therefore copy the MCS if asked for neither a task oriented CS nor a merge oriented CS. Second, when asked for a task oriented CS, `createcs` will replace "`-nocheckout`" clauses with "`-mkbranch task_branch`" clauses.

In order to emphasize that an MCS is *read-only*, the MCSs always have the "`element * CHECKEDOUT`" rule commented out. `createcs` will uncomment it when it generates either a task oriented or merge oriented CS.

5.2 Language Constructs

The CS meta language is broken up into three types of constructs: the first type defines certain characteristics global to the MCS, the second type get turned into explicit version rules and the third type modify the actions that will be performed on a range of CS rules.

There are currently two flavors of MCSs: project and sub-project. We have not yet found a need for any finer grain breakdown. Further, we have found that more than 80% of the work done on any product can be done simply by using a project level MCS, although these results are most likely process/organization specific.

5.2.1 Master Definitions

Found near the top of the MCS before any actual CS rules, these statements define the master, or principle, collection branches to be used in the project.

5.2.1.1 The `MPB` Statement

The `MPB` statement defines the "master project branch" for the MCS. This is the branch that is normally used to collect the project changes for 80% to 100% of the product for this project. The statement looks like:

```
# MPB: master_project_branch_name
```
(EQ 2)

5.2.1.2 The `MCB` Statement

The `MCB` statement defines the "master collection branch" for the MCS. This is the branch that will be merged to in the majority of cases when a merge oriented CS is called for by the engineer. This statement looks like:

```
# MCB: master_collection_branch_name
```
(EQ 3)

5.2.1.3 `MPB` versus `MCB`

If the MCB is the same as the MPB then the MCS is a project MCS otherwise it is a sub-project MCS. We believe the declaration of the MPB in a sub-project MCS is good documentation to which project the sub-project is derived[1].

5.2.2 The CB Statement

In general, labels are used to identify a version per element for a set of elements that can be looked on as a self-consistent whole. "Backing labels" are labels which identify a version per element for a set of elements that will be used to root collection branch instances (if the collection branch instance needs to be created due to a change in the system). The `CB` statement(s) defines the relationship between backing labels and collection branches. As we noted before, we have different branching policies in different

1. It is also deemed "good practice" to include the version ID of the Project MCS that was used to form the Sub-project MCS in a comment in the Sub-project MCS.

 (c) 1996 Hewlett Packard Company

parts of our source base. We distinguish between those policy areas by having a {backing label, collection branch} pair for each.

```
# CB: <label_type> <branch_type>
```
(EQ 4)

This information is used by `createcs` when generating a merge oriented CS in order to produce the correct "`–mkbranch`" clauses.

5.2.3 CS Explicit Version Statements

There are currently two of these: one that gets expanded into the version rule calling for checked out elements and one that gets expanded into the version rule calling for the engineer's task branch.

5.2.3.1 The CHECKEDOUT Statement

This statement is any commented out version rule calling for checked out versions. A CHECKEDOUT statement can be included anywhere in the MCS and it can be included more than once (hopefully in more than one form). As noted before, statements of this type will be uncommented by `createcs` if the desired CS is other than read-only. For example:

```
# element /vob/krb5/... CHECKEDOUT
```
(EQ 5)

and somewhere further down in the file:

```
# element * CHECKEDOUT
```
(EQ 6)

5.2.3.2 The Task Branch Statement

This statement is any commented out version rule with "`@PRIVATE@`" embedded in it. The rule will be uncommented and the "`@PRIVATE@`" token replaced with the engineer's task branch name if `createcs` is asked to produce a task oriented CS. This normally is used once in the general form near the top of the MCS. For example:

element * .../@PRIVATE@/LATEST **(EQ 7)**

5.2.4 CS Rule Modifier Statements

The CS rule modifier statements all are constructed in such a way as to modify a *set* of lines. Thus, since they are embedded in CS comments, they each have a "`BEGIN`" half and an "`END`" half. While these statements are not normally intended to be nested, they are presented in priority order from highest to lowest: if more than one statement is applicable to a rule then the highest priority statement will control the behavior of `createcs`.

The first three modify CS version rules, the next three modify CS time rules.

5.2.4.1 NOCHECKOUT – – – – – – – – – – – – – – task and merge

This statement causes the enclosed version rules to always be modified by the "`–nocheckout`" option.

```
# NOCHECKOUT-BEGIN
# NOCHECKOUT-END
```
(EQ 8)

5.2.4.2 NOMERGE - - - - - - - - - - - - - - - - - - - merge

This statement causes the enclosed version rules to be modified by the "`–nocheckout`" option when `createcs` is generating a merge oriented CS. When `createcs` is generating a task oriented CS, the global rule of replace "`–nocheckout`" with "`–mkbranch task_branch`" is followed.

This allows engineers to do "what if" experiments on their task branches in such areas as the build environment and yet does not allow them to automatically merge those changes back onto the active source branches. Instead they must manually edit the CS: a procedure that normally requires agreement from some project level manager.

```
# NOMERGE-BEGIN
# NOMERGE-END
```
(EQ 9)

5.2.4.3 MCB-CTRL - - - - - - - - - - - - - - - - - - - merge

This statement causes the enclosed version rules to be modified by the "`–mkbranch MCB`" option when `createcs` is generating a merge oriented CS. When `createcs` is generating a task oriented CS, the global rule of replace "`–nocheckout`" with "`–mkbranch task_branch`" is followed.

The intent is that this statement enclose sets of version rules that cover elements that do not have the normal project backing label because they were created since the backing label was applied (and therefore the `CB` rules above do not cover them).

```
# MCB-CTRL-BEGIN
# MCB-CTRL-END
```
(EQ 10)

This is probably an area that could use improvement. Rather than only allowing merges to either the MCB or to /main (the only other current alternative), a new statement should be created that specifies the collection branch into which to merge.

5.2.4.4 MERGE-TIME - - - - - - - - - non-sub-project merge

This statement causes the enclosed CS time rules to have their times replaced with a time far into the future when `createcs` is generating a merge oriented CS. This effectively unlocks the meaning of `LATEST` so that true integration will take place.

```
# MERGE-TIME-BEGIN
# MERGE-TIME-END
```
(EQ 11)

5.2.4.5 source time: - non-sub-project merge

Identical in effect to `MERGE-TIME` above, except that there is no `BEGIN / END` pair: rather this statement modifies all subsequent time rules. This statement is historical: `MERGE-TIME` is now the preferred usage.

```
# source time:
```
(EQ 12)

5.2.4.6 SP-MERGE-TIME - - - - - - - - - - - sub-project merge

This statement is identical in effect to `MERGE-TIME` above, except that it is only honored if the MCS is for a sub-project (it is an error to use it in a project level MCS).

```
# SP-MERGE-TIME-BEGIN
# SP-MERGE-TIME-END
```
(EQ 13)

 (c) 1996 Hewlett Packard Company

5.2.5 Build Environment Statements

Some of the CS meta language is inextricably linked with our build environment. To make them understandable we will briefly describe the overall build process.

We build our product in phases: first we build a set of tools that will be used in subsequent phases to build the product and *check them in*, second and subsequent phases build the product. Checking in the tools is an optimization to the build process and is beyond the scope of this paper but it is necessary to understand that we do this in order to understand Section 5.2.5.1 on page 9 below.

As noted earlier, we generally do not label our nightly builds. Instead we capture the time when we start the process. This time is used to version all of the actual sources.

5.2.5.1 Tool Time

This is the time at which we freeze the meaning of `LATEST` for the checked in tools (N.B. tool time is normally *later* than source time - see below). This time is captured at the end of the tool generation / checkin phase of the build process. Normally it only modifies the CS version rules that explicitly cover the VOB subtree where the tools are checked in.

```
# tool time:
time 09-Aug-1996.17:43:59
```
(EQ 14)

5.2.5.2 Source Time

This is the time at which we lock the meaning of `LATEST` for all product source code, including the source code used to generate the tools. Historically, `createcs` knows about this rule so that it can unlock the meaning of `LATEST` during a merge (see Section 5.2.4.5 on page 8).

```
# source time:
time 09-Aug-1996.17:00:05
```
(EQ 15)

5.2.5.3 Environment Variables

In order to build our product an engineer must enter the "build environment". This is done by executing the command:

```
cworkon <view_tag>
```
(EQ 16)

The `cworkon` command does a number of special things, but the one that is germane here is that it looks in the CS in the specified view, finds all comment lines of the form:

```
# %ENV% EV_NAME=EV_VALUE
```
(EQ 17)

and sets each environment variable "`EV_NAME`" to have the value "`EV_VALUE`".

Most of the `%ENV%` statements are written into the MCSs by the engineer responsible for maintaining them. One special case is created by `createcs` only during the generation of a task oriented CS:

```
# %ENV% WORK_BRANCH=<specified_task_branch>
```
(EQ 18)

This environment variable is then available for shell scripts to use. For example, one of the shell scripts available uses this environment variable to generate context differences for all elements with instances of the task branch type to show all work done so far on this task.

6.0 The CS Generation Tool: "`createcs`"

The `createcs` tool is currently written as a combination of Ksh, awk, etc. and processes MCSs into different flavors of CS.

Project MCSs are versioned as part of the normal nightly process. All project MCSs are found in a specific directory in a particular VOB. We have hardwired the path to that directory into `createcs` in order to make it easy for our engineers to specify the project MCS (see below). A project MCS is a file element in that directory.

Since the MCS directory is in a VOB, it must be accessed through a view. If one of the options that specifies a view is provided by the engineer, then the specified view will be used. Otherwise there is a globally available view that is used. In either case, the path to the MCS directory is always found by using the standard default CS (i.e. /main/LATEST). If the engineer's specified view is used then the first step in the generation process is to set the CS in that view to the default. This is not necessary if the globally available view is used since it always has the default CS set.

6.1 Required Argument: The "Project"

We normally have many projects underway concurrently. Therefore the engineer must specify the project for which they want a CS generated. Because `createcs` knows where the project MCSs are found, the engineer must provide only the project leafname. If `createcs` is not given a project, then it will display a list of all projects to the engineer as part of its usage error message.

Normally the engineer uses the latest MCS for a given project, but in rare cases they might need prior versions. Therefore, the specified project can include a version specifier, e.g. "`project@@/main/35`".

6.2 Options

6.2.1 No Options

If `createcs` is given no options then a "read-only" or "perusal" CS is generated and output to standard out.

6.2.2 `-b <task_branch>`

Produce a CS appropriate for an engineering task. All files checked out will be on the specified task branch.

6.2.3 `-merge`

Produce a CS appropriate for use for merging a task to the appropriate collection branch(s) as specified in the MCS.

 (c) 1996 Hewlett Packard Company

6.2.4 -cview or -ctag

Automatically set the generated CS into the current view.

6.2.5 -o <outfile>

Place the generated CS in the specified file. Does not conflict with "-cview" or "-ctag".

6.3 Additional Processing

When asked to produce a task oriented CS via the "-b <task_branch>" option, createcs checks to see if the intended task branch exists or not. If it does not, createcs prints to stderr the command necessary to create the task branch.

7.0 Sample Project MCS

The following is a simplified version of one of our current project level MCSs:

7.1 The `RELX` MCS itself:

```
# MPB:relx
# MCB:relx
# CB: RELX_START  relx
# CB: PKG_Y_DROP_6pkgy_bugfix
# element * CHECKEDOUT
# element * .../@PRIVATE@/LATEST
element /vob/src/ccase/... /main/LATEST
# NOCHECKOUT-BEGIN:
element /vob/notouch/... SPCL_LBL         -nocheckout
# NOCHECKOUT-END.
# NOMERGE-BEGIN:
element /vob/tools/... BE_10.20_FCS       -nocheckout
# NOMERGE-END.
# MERGE-TIME-BEGIN:
time 13-Jun-1995.19:00
# MERGE-TIME-END.
element /vob/src/HP/... /main/LATEST      -nocheckout
element /vob/pkgy/... .../pkgy_bugfix/LATEST -nocheckout
element /vob/pkgy/... PKG_Y_DROP_6        -nocheckout
element * .../relx/LATEST                 -nocheckout
element * RELX_START                      -nocheckout
# MCB-CTRL-BEGIN:
element /vob/src/... /main/LATEST         -nocheckout
# MCB-CTRL-END.
element * /main/LATEST                    -nocheckout
time 01-Jan-2001
# MCB-CTRL-BEGIN:
element /vob/src/... /main/0              -nocheckout
# MCB-CTRL-END.
element * /main/0                         -nocheckout
```

 (c) 1996 Hewlett Packard Company

7.2 Output of "`createcs -b dbr_task1 RELX`"

```
# %ENV% WORK_BRANCH=dbr_task1
# MPB:relx
# MCB:relx
# CB: RELX_START  relx
# CB: PKG_Y_DROP_6pkgy_bugfix
element * CHECKEDOUT
element * .../dbr_task1/LATEST
element /vob/src/ccase/... /main/LATEST
# NOCHECKOUT-BEGIN:
element /vob/notouch/... SPCL_LBL   -nocheckout
# NOCHECKOUT-END.
# NOMERGE-BEGIN:
element /vob/tools/... BE_10.20_FCS -mkbranch dbr_task1
# NOMERGE-END.
# MERGE-TIME-BEGIN:
time 13-Jun-1995.19:00
# MERGE-TIME-END.
element /vob/src/HP/... /main/LATEST-mkbranch dbr_task1
element /vob/pkgy/... .../pkgy_bugfix/LATEST \
                                    -mkbranch dbr_task1
element /vob/pkgy/... PKG_Y_DROP_6  -mkbranch dbr_task1
element * .../relx/LATEST           -mkbranch dbr_task1
element * RELX_START                -mkbranch dbr_task1
# MCB-CTRL-BEGIN:
element /vob/src/... /main/LATEST   -mkbranch dbr_task1
# MCB-CTRL-END.
element * /main/LATEST              -mkbranch dbr_task1
time 01-Jan-2001
# MCB-CTRL-BEGIN:
element /vob/src/... /main/0        -mkbranch dbr_task1
# MCB-CTRL-END.
element * /main/0                   -mkbranch dbr_task1
```

7.2.1 Text from stderr:

```
NOTE: Branch type dbr_task1 doesn't exist yet.
NOTE: You will need to do a 'allvobs mkbrtype dbr_task1' to create it.
```
(EQ 19)

7.3 Output of "`createcs -merge RELX`"

```
# MPB:relx
# MCB:relx
# CB: RELX_START  relx
# CB: PKG_Y_DROP_6pkgy_bugfix
element * CHECKEDOUT
# element * .../@PRIVATE@/LATEST
element /vob/src/ccase/... /main/LATEST
# NOCHECKOUT-BEGIN:
element /vob/notouch/... SPCL_LBL   -nocheckout
# NOCHECKOUT-END.
# NOMERGE-BEGIN:
element /vob/tools/... BE_10.20_FCS -nocheckout
# NOMERGE-END.
# MERGE-TIME-BEGIN:
time 01-Jan-2010
# MERGE-TIME-END.
element /vob/src/HP/... /main/LATEST
element /vob/pkgy/... .../pkgy_bugfix/LATEST
element /vob/pkgy/... PKG_Y_DROP_6 -mkbranch pkgy_bugfix
element * .../relx/LATEST
element * RELX_START                -mkbranch relx
# MCB-CTRL-BEGIN:
element /vob/src/... /main/LATEST   -mkbranch relx
# MCB-CTRL-END.
element * /main/LATEST
time 01-Jan-2001
# MCB-CTRL-BEGIN:
element /vob/src/... /main/0
# MCB-CTRL-END.
element * /main/0
```

 (c) 1996 Hewlett Packard Company

8.0 Sample Sub-project MCS

The following is an example of a sub-project MCS:

8.1 The **SPY** MCS itself:

```
# (RELX@@/main/42)
# MPB:relx
# MCB:spy
# CB: RELX_START  relx
# CB: PKG_Y_DROP_6pkgy_bugfix
# element * CHECKEDOUT
# element * .../@PRIVATE@/LATEST
element /vob/src/ccase/... /main/LATEST
# SP-MERGE-TIME-BEGIN:
time 04-Jul-1995.13:13
element * .../spy/LATEST            -nocheckout
# SP-MERGE-TIME-END.
# NOCHECKOUT-BEGIN:
element /vob/notouch/... SPCL_LBL   -nocheckout
# NOCHECKOUT-END.
# NOMERGE-BEGIN:
element /vob/tools/... BE_10.20_FCS -nocheckout
# NOMERGE-END.
# MERGE-TIME-BEGIN:
time 13-Jun-1995.19:00
# MERGE-TIME-END.
element /vob/src/HP/... /main/LATEST-nocheckout
element /vob/pkgy/... .../pkgy_bugfix/LATEST -nocheckout
element /vob/pkgy/... PKG_Y_DROP_6  -nocheckout
element * .../relx/LATEST           -nocheckout
element * RELX_START                -nocheckout
# MCB-CTRL-BEGIN:
element /vob/src/... /main/LATEST   -nocheckout
# MCB-CTRL-END.
element * /main/LATEST              -nocheckout
time 01-Jan-2001
# MCB-CTRL-BEGIN:
element /vob/src/... /main/0        -nocheckout
# MCB-CTRL-END.
element * /main/0                   -nocheckout
```

8.2 Output of "**createcs -b dbr_task2 SPY**"

```
# %ENV% WORK_BRANCH=dbr_task2
# (RELX@@/main/42)
# MPB:relx
# MCB:spy
# CB: RELX_START  relx
# CB: PKG_Y_DROP_6pkgy_bugfix
element * CHECKEDOUT
element * .../dbr_task2/LATEST
element /vob/src/ccase/... /main/LATEST
# SP-MERGE-TIME-BEGIN:
time 04-Jul-1995.13:13
element * .../spy/LATEST            -mkbranch dbr_task2
# SP-MERGE-TIME-END.
# NOCHECKOUT-BEGIN:
element /vob/notouch/... SPCL_LBL   -nocheckout
# NOCHECKOUT-END.
# NOMERGE-BEGIN:
element /vob/tools/... BE_10.20_FCS -mkbranch dbr_task2
# NOMERGE-END.
# MERGE-TIME-BEGIN:
time 13-Jun-1995.19:00
# MERGE-TIME-END.
element /vob/src/HP/... /main/LATEST-mkbranch dbr_task2
element /vob/pkgy/... .../pkgy_bugfix/LATEST \
                                    -mkbranch dbr_task2
element /vob/pkgy/... PKG_Y_DROP_6  -mkbranch dbr_task2
element * .../relx/LATEST           -mkbranch dbr_task2
element * RELX_START                -mkbranch dbr_task2
# MCB-CTRL-BEGIN:
element /vob/src/... /main/LATEST   -mkbranch dbr_task2
# MCB-CTRL-END.
element * /main/LATEST              -mkbranch dbr_task2
time 01-Jan-2001
# MCB-CTRL-BEGIN:
element /vob/src/... /main/0        -mkbranch dbr_task2
# MCB-CTRL-END.
element * /main/0                   -mkbranch dbr_task2
```

8.2.1 On stderr:

```
NOTE: Branch type dbr_task2 doesn't exist yet.
NOTE: You will need to do a 'allvobs mkbrtype dbr_task2' to create it.
```
(EQ 20)

 (c) 1996 Hewlett Packard Company

8.3 Output of “`createcs -merge MCS`”

```
# (RELX@@/main/42)
# MPB:relx
# MCB:relx
# CB: RELX_START  relx
# CB: PKG_Y_DROP_6pkgy_bugfix
element * CHECKEDOUT
# element * .../@PRIVATE@/LATEST
element /vob/src/ccase/... /main/LATEST
# SP-MERGE-TIME-BEGIN:
time 01-Jan-2010
element * .../spy/LATEST
# SP-MERGE-TIME-END.
# NOCHECKOUT-BEGIN:
element /vob/notouch/... SPCL_LBL   -nocheckout
# NOCHECKOUT-END.
# NOMERGE-BEGIN:
element /vob/tools/... BE_10.20_FCS -nocheckout
# NOMERGE-END.
# MERGE-TIME-BEGIN:
time 01-Jan-2010
# MERGE-TIME-END.
element /vob/src/HP/... /main/LATEST-mkbranch spy
element /vob/pkgy/... .../pkgy_bugfix/LATEST \
                                    -mkbranch spy
element /vob/pkgy/... PKG_Y_DROP_6  -mkbranch spy
element * .../relx/LATEST           -mkbranch spy
element * RELX_START                -mkbranch spy
# MCB-CTRL-BEGIN:
element /vob/src/... /main/LATEST   -mkbranch spy
# MCB-CTRL-END.
element * /main/LATEST              -mkbranch spy
time 01-Jan-2001
# MCB-CTRL-BEGIN:
element /vob/src/... /main/0        -mkbranch spy
# MCB-CTRL-END.
element * /main/0                   -mkbranch spy
```

9.0 Future Directions

We believe that we need to rewrite the CS generation tool, createcs, in Perl 5 in order to increase maintainability and portability. This will also enable us to use the same tool and technique for producing CSs for use on Windows/NT, although there will need to be other changes, e.g. VOB tags are different on NT than on UNIX.

10.0 Acknowledgments

I would like to thank all those who participated in the effort of which Configuration Generation is just a part.

I would especially like to thank for their help with this paper (presented alphabetically):

Mary MacGregor
Ralph Morrison
Bill Sommerfeld
Mark Vincenzes

(c) 1996 Hewlett Packard Company