CMT

Configuration Management Tool

Version v1r14

Christian Arnault

arnault@lal.in2p3.fr

(eg. by *prefixing* components of the package by its mnemonic).

CMT is operated through one main user interface : the **cmt** command, which handles the **CMT** conventions and which provides a set of services for :

```
csh> cd ~/mydev/Foo/v1/cmt
csh> source setup.csh
```

or

```
dos> cd \mydev\Foo\v1\cmt
dos> call setup.bat
```

The **FOOROOT** and **FOOCONFIG** environment variables are defined automatically by tp cs

```
csh> vi requirements
...
application FooTest FooTest.c
csh> gmake
csh> source setup.csh
csh> FooTest.exe
Hello Foo
```

follow the convention :

1. there is a first directory level exactly named according to the package name (this is case sensitive),
2. then (optionally) the next directory level is named according to the version tag,
3. then there is a branch named **cmt** ,
4. lastly there is a *requirements* file within this **cmt** branch.

Thus the list of access paths is searched for until these conditions are properly met.

The actual complete search list can always be visualized by the command:

```
> cmt show path
# Add path /home/arnault/dev from CMTPATH
# Add path /ProjectB from CMTPATH
# Add path /lal from default path
#
/home/arnault/dev:/ProjectB:/lal
```

---

6 -environnts

Many configuration parameters are supposed to be described into this requirements file - see the

Carefully describing this configuration is essential both for maintenance operations (so as to remember the precise conditions in which the package was built) and when the development area is *shared*

- The current minor version id of CMT is a valid tag and takes the form **CMTr<n>** (eg. **CMTr14**)
- The current patch id of CMT is a valid tag and takes the form **CMTp<n>** (eg. **CMTp20030616**)

3. User defined tags can be explicitly or implicitly activated:

```
tag newtag tag1 tag2 tag3
```

which means that:

- **newtag** defines a tag
- when **newtag** is active, then both tag1, tag2 and tag3 are simultaneously active

- Tags may be declared as *exclusive* using the **tag_exclude** syntax.

```
tag_exclude debug optimized
```

This example implies that the two tags **debug 11 Tf 9.16299 0a Tf e23 Td ( )Tj tag /R116 11 Tf 23.81.384Td (**

Eg the following syntax installed in a requirements file will force the tag **foo** :

```
tag_apply foo
```

```
> cmt show tags
CMTv1 (from CMTVERSION)
CMTr14 (from CMTVERSION)
CMTp0 (from CMTVERSION)
```

```
Linux (from uname)
Linux-i686 (from CMTCONFIG) package CMT implies [Linux]
tag1 (from arguments)
tag2 (from arguments)
tag3 (from arguments)
Default (from Default)
```

The **Foo_linkopts** conventional macro will be automatically inserted within the **use_linkopts** macro. And the shared library location will be automatically set to the installation areas.

Let’s consider as an example the project named **MyProject** . We may create the package named **MyProject** similarly to any other package :

```
csh> cd .....
csh> cmt create MyProject v1 /ProjectB
```

Then the **requirements** file of this new package will simply contain a set of **use** statements, defining the *official* set of validated versions of the packages required for the project. This mechanism also represents the notion of *global release* traditionally addressed in configuration management environments

```
package MyProject

use Cm v7r6
use Db v4r3
use El v4r2
use Su v5
use DbUI v1r2 Db
use ElUI v1r1 El
use VSUUI v3  Su/VSU
use VMM   v1
use VPC   v3
```

Then any user wanting to access the so-called *official* release of the package set appropriate to the project **MyProject** will simply do (typically within its login shell script) :

```
# a login script

...
```

the**MyProject**

- A user package willing to apply this behaviour will have to include in its requirements file a statement similar to the following:

```
document tex MyDoc -s=../doc doc1.tex doc2.tex
```

where:

1. The first parameter"tex" is the document-style
2. The second parameter "MyDoc" is used for building the constituent's makefile (under the name MyDoc.make) and for providing the make target "MyDoc".
3. The other parameters (doc1.tex and doc2.tex) are the sources of the document. Explicit location is required (since default is currently defined to be ../src)
4. The constituent's makefile MyDoc.make is built as follows :
    1. Install a copy of the **$CMTROOT/fragments/make_header** generic fragment
    2. Install a copy of the **$CMTROOT/fragments/tex_header** fragment
    3. For each of the sources, install a copy of the fragment "tex"
    4. Install a copy of the **$CMTROOT/fragments/cleanup_header** fragment

The result for our example is:

```
========== MyDoc.make ==============================

#====================================
#  Document MyDoc
#
#   Generated   by
#
#====================================

help ::
@echo 'MyDoc'

doc1_dependencies = ../doc/doc1.tex
doc2_dependencies = ../doc/doc2.tex

MyDoc :: ../doc/doc1.ps

../doc/doc1.dvi : $(doc)doc1.tex
        cd ${doc}es = ../doc/doc1.tex
doc2_dependencies = ../doc/doc2.tex



../doc/doc1.dvi : $(doc)doc1.tex
doc2_depend../doc/doc2.tex= ../doc/doc2.-10.acccccccccc.C r tai9x
```

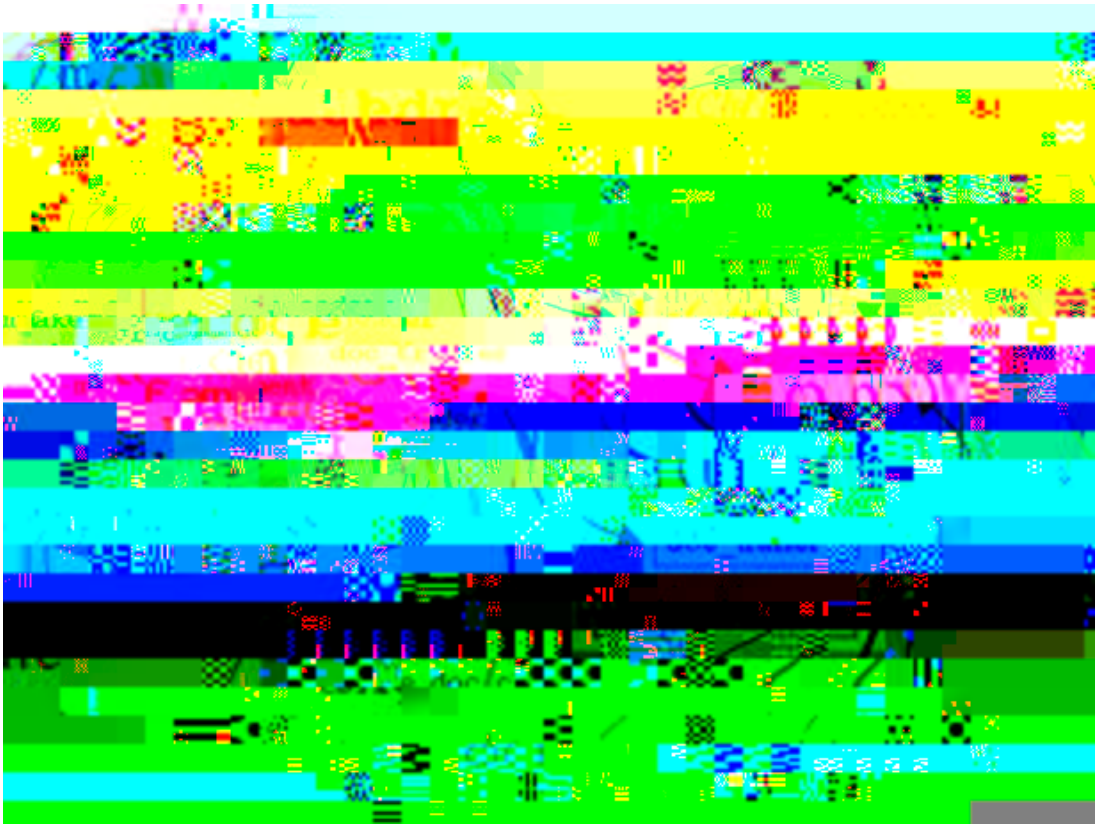
```
        cd .

MyDocclean ::
=============================================================
```

---

10. 2 - How to create and install a new document style

This section presents the general framework for designing a document generator.

1.



```
========= agetocxx_header ==================================
${CONSTITUENT}lib       = $(bin)lib${CONSTITUENT}.a
${CONSTITUENT}stamp     = $(bin)${CONSTITUENT}.stamp
${CONSTITUENT}shstamp   = $(bin)${CONSTITUENT}.shstamp

${CONSTITUENT} :: dirs ${CONSTITUENT}LIB
        @/bin/echo ${CONSTITUENT} ok

${CONSTITUENT}LIB :: $(${CONSTITUENT}lib) $(${CONSTITUENT}shstamp)
        @/bin/echo ${CONSTITUENT} : library ok

$(${CONSTITUENT}lib) $(${CONSTITUENT}stamp) :: ${OBJS}
        $(ranlib) $(${CONSTITUENT}lib) ${OBJS}
s $(bin)lerdev/null >NSTITUENT}stamp) :: ${O-21.6 Td ($(${CONSTITUENT}lib) $p) )Tj 0 {CONSTI
```

Each statement is composed of words separated with spaces or tabulations.

The first word of a statement is the name of the configuration parameter.

Applications and libraries are assigned a name (which will correspond to a generated make fragment, and a dedicated make target).

A document is first associated with a document type (which must correspond to a previously declared make fragment). The document name is then used to name a dedicated make fragment and a make target.

Various options can be used when declaring a constituent:

option	*validity*	*usage*
-windows	applications	When used in a Windows environment,

2.

It’s possible to specify in the list of parameters one or more pairs of **variable-name** = **variable-value** (without any space characters around the **"="** character), such as in the next example:

```
make_fragment doc_to_html                                    (1)
```

The tag is used to select one alternate value to replace the default value, when one of the following condition is met:

-

Typical examples are the definition of the search path for shared libraries (through the **LD_LIBRARY_PATH**

```
include_path ${PackA_root}/PackA/
```

2. Providing a value to the **LD_LIBRARY_PATH** environment variable

   On some operating systems (eg. Linux), shared library paths must be explicited, through an environment variable. The following pattern can automate this operation:

```
pattern ld_library_path \
path_remove LD_LIBRARY_PATH "/<package>/" ; \
path_append LD_LIBRARY_PATH ${<PACKAGE>ROOT}/${CMTCONFIG}
```

   operation:-99.9011j /tmp/LD_LIBRARY_PATH Td (pattern T}/${CMTCONFIG})Tj /R122 11 mentschema inst

   opera7.60417IG}

```
path        CMTPATH "/ProjectA"
path_append CMTPATH "/ProjectB"

cmtpath_pattern \
  macro_prepend pp_cppflags " -I<path>/InstallArea/include "
```

11. 2.14 - include_path - -

11. 2.17 - public, private

Introduce a section for

Tags or associations of tags are propagated using the -tag=<tag-list>options to the cmt command driver, but the Make command can also accept them through the conventional macros *$(tag)* and *$(extra_tags)* . However, the *$(tag)* macro itself can only accept one value (instead of a list), and therefore in order to give a list of additional tags, one should use the *$(extra_tags)* (such as in *gmake tag=Linux extra_tags=debug*)

Finally, running the setup script (through the *source setup.[c]sh* or *call setup.bat* command) can also receive tag specifications using the *-tag=tag-list* options.

---

11. 3 - The general cmt user interface

11.3.1.2 - Templates in the shell command

Similarly to what exists in the pattern mechanism, some standard *templated* values can be embedded inside the command to be executed by the broadcast action. They take a standard form of **<*template-name*>** . These templates acquire their value on each package effectively reached during the broadcast scan, and the effective value is substituted before launching the command. The possible templates are:

<package_cmtpath>	The element in the CMTPATH search list where the package has been found
<package_offset>	The directory offset to cmtpath
<package>	The package name
<version>	The version of the package

All such constituent fragments are automatically included from the main Makefile.

Although this command is meant to be used internally (and transparently) by **CMT**

- **vsnet**

  This command generates workspace and project files required for the Visual.net tool.

- **os9_makefile**

  This command generates external dedicated *makefile* fragments for each individual component of the package (ie. libraries or executable applications) to be used in OS9 context. It generates specific syntaxes for the

11. 3. 3 - cmt check configuration

11.3.12 - cmt lock [<package> <version> [<area>]]

This command tries to set a lock onto the current package (or onto the specified

11. 3.16 - cmt set version

```
csh> set compiler=‘cmt show macro_value cppcomp‘
csh> ${compiler} ....
```

1. Connection to the current version of the **CMT** package.
2. Setting the set of user defined public variables specified in the requirements file (including those defined by all used packages). This is achieved by running the **cmt setup** utility into a temporary file and running this temporary file.
3. Activation of the user defined setup and cleanup scripts (those specified using the

Although none of these are required, the **cmt** general command provides a few utilities so as to simplify the use of these practices. It should be noted that the added features provided by cmt rely on the possibility to *query* CVS about the existing **CMT** packages and the possible tags setup for these packages. CVS does not by default permit such query operations (since they require to scan

take care of this effective location.

```
sh> cd <new-tag>/cmt
sh> cmt config
sh> source setup.csh
sh> [g]make
```

12.5 - Getting a particular tagged version out of CVS

The previous example presented the standard case where one gets the *most*

```
package OPACS

include_dirs ${Wo_root}/include ${Co_root}/include ${Xx_root}/include \
${Ho_root}/include ${Go_root}/include ${Xo_root}/include

public
macro OPACS_cflags      "-DHAS_XO -DHAS_XM"
macro OPACS_cppflags    " $(OPACS_cflags) "

macro OPACS_linkopts    "$(Wo_linkopts) $(Xo_linkopts) $(Go_linkopts) \
$(Glo_linkopts) $(Xx_linkopts) $(Ho_linkopts) $(Htmlo_linkopts) \
$(W3o_linkopts) $(Co_linkopts) $(X_linkopts)"
```

Then every package or application, client of this

2. Platform dependent libraries
3. Public header files
4. Platform independent applications (eg Java applications)
5.

Another important concern is the

however caused and on any theory of liability, whether in contract, strict liability, or tort (including negligence or otherwise) arising in any way out of the use of this software, even if advised of the possibility of such damage.

---

macro	*usage*

cpp The C++ compiler **g++**

cppcomp

The<package>_native_version is not subject to automatic concatenation.

< *package* >_cflags	specific C flags
< *package* >_pp_cflags	specific C preprocessor flags
< *package* >_cppflags	specific C++ flags
< *package* >_pp_cppflags	specific C++ preprocessor flags
< *package* >_fflags	specific Fortran flags
< *package* >_pp_fflags	specific Fortran preprocessor flags
< *package* >_libraries	gives the (space separated) list of library names exported by this package. This list is typically used in the **cmt build library_links** command.
< *package* >_linkopts	provide the linker options required by any application willing to access the different libraries offered by the package. This may include support for several libraries per package.

A ***package*** examsrary . 0 -2kowccesdefinportch ainccro couj 4be256.817 0 6d ( )Tj :R122 1140
packastry R122 11 Tf 49.6720 Td ( )Tj 2fi. 0 expor(vermak liargets). Whenl li a46.2811

< *package*
>_home

< ***category*** >_< ***constituent*** >**_cflags**	specific C flags
< ***category*** >_< ***constituent*** >**_pp_cflags**	specific C preprocessor flags
< ***category*** >_< ***constituent*** >**_cppflags**	specific C++ flags
< ***category*** >_< ***constituent*** >**_pp_cppflags**	specific C++ preprocessor flags
< ***category*** >_< ***constituent*** >**_fflags**	specific Fortran flags
< ***category*** >_< ***constituent*** >**_pp_fflags**	specific Fortran preprocessor flags
< ***constituent*** >**linkopts**	provides additional linker options to the application. It is complementary to - and should not be confused with - the < ***package*** >**_linkopts** macro, which provides exported linker options required by clients packages to use the package libraries.
< ***constituent*** >**_shlibflags**	provides additional linker options used when building a shared library. Generally, a simple shared library does not need any external reference to be resolved at build time (it is in this case supposed to get its unresolved references from other shared libraries). However, (typically when one builds a dynamic loading capable component) it might be desired to statically link it with other libraries (making them somewhat private).
< ***constituent*** >**_dependencies**	provides user defined dependency specifications for each constituent. The typical use of this macro is fill it with the name of the list of some other constituents which *have* to be rebuilt first (since each constituent is associated with a target with the same name). This is especially needed when one want to use the parallel gmake (ie. the -j option of gmake).
< ***group*** >**_dependencies**	provides user defined dependency specifications for each group. The typical use of this macro is fill it with the name of the list of some other constituents which *have* to be rebuilt first (since each constituent is associated with a target with the same name). This is especially needed when one want to use the parallel gmake (ie. the -j option of gmake).

17. 3. 5 - Source specific customizing macros

These macros do not receive any default values (ie they are empty by default). They are

17.3.7 - Utility macros

These macros are used to specify the behaviour of various actions in CMT.

X11_cflags	compilation flags for X11
Xm_cflags	compilation flags for Motif
X_linkopts	Link options for XWindows (and Motif)
make_shlib	The command used to generate the shared library from the static one
shlibsuffix	The system dependent suffix for shared libraries
shlibbuilder	The lon12243.5 43.5418 0 Td a

PROTOSTAMPS	prototype stamp files	protos_header
PROTOTARGET	prototype target name	library_header application_header
SUFFIX	document suffix	*< document* *ta >4.95 T202.807 -41.588.762TITLETd (document)Tj 27.797itle forsuffixsuffix<*

SUFFIX

Generated

17. 6 - The complete requirements syntax

The syntax of specification statements that can be installed in a **requirements** file are :

cmt-statement	:	*application*
	|	*apply_pattern*
	|	*apply_tag*
	|	*author*
	|	*branches*
	|	*build_strategy*
	|	

|

alias : alias *alias-name default-value* [*tag-expr value ...*]

application : application *application-name* [

document : document *document-name* [*constituent-option* ...]
[*source* ...]

ignore_pattern : ignore_pattern *pattern-name*

include_dirs : include_dirs *search-path*

include_path : include_path *search-path*

language : language *language-name* [*language-option* ...]

language-option : -suffix=*suffix*
| -linker=*linker-command*
| -prototypes
| -preprocessor_command=*preprocessor_command*
| -fragment=*fragment*
| -output_suffix=*suffix*

```
cvs -Q import -m cmt .cmtcvsinfos/<
```

9.4 Construction of a global environment