CMT

Configuration Management Tool

2 - The conventions

This environment relies on a set of conventions, mainly for organizing the directories where packages are maintained and developed :

2 - *Structuring a sofware base.*

---

3 - The architecture of the environment

This environment is based on the fact that one of its packages (named `CMT`

Therefore, we assume that *some* root directory has been selected by the system manager, and that `CMT` is already installed there. One first has to *setup*

The package creation occured from the current directory, creating from there the complete

```
csh> cd ~/m 1 0 0dev/Foo/v1/cmt
csd> or
```

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

Directly running the application is possible since the application has been installed after being built in an automatic *installation area* reachable through the standard PATH environment variable

This can also be integrated in the build process by providing the -check option to the application definition:

```
csh> cd ../cmt
csh> vi requirements
...
application FooTest -check FooTest.c
csh> gmake check
Hello Foo
```

---

5 - Localizing a package

In the next sections, we'll see that packages *reference* each other by means of *use* relationships. Generally packages are found in different locations, according to the project - or sub-project - they belong to. relationshij /t0 -10 witht lits11 Tf 6d o.

The configuration parameter `CMTPATH` *can be specified either in the environment variable named* `CMTPATH` *or in* `.cmtrc` *files, which can themselves be located either in the current directory, in the home directory of the developper or in* `${CMTROOT}/mgr` *. In the Windows environment, this configuration parameter may also be installed as a Registry under either the keys:*

- `HKEY_LOCAL_MACHINE/Software/CMT/path`
- `HKEY_CURRENT_USER/Software/CMT/path`

(A graphical tool vailable in

2. `/ProjectB/A/BarA/v1/cmt`
3. `/lal/A/BarA/v1/cmt`
4. the sub-directory `A` within the same path as the current package,

The packages are searched assuming that the directory hierarchy below the access paths always 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.

As an example, suppose we have set up the following CMTPATH :

```
$HOME/work-for-A:/ProjectA:/ProjectB
```

The basic contents of such a package is the requirements file including

7.4 - The release package

This package is one particular example of the container concept, but dedicated to manage the project-wide activities. This release package is the primary target of the project manager. It will generally receive as its version tags the version tags assigned to the project releases themselves.

7.5 - The glue or interface package

This kind of package defines an interface to an existing software product not managed in the context of the project itself. Typical examples concern:

- packages shared from external projects that don’t use CMT as their configuration tool
- third party software (free software, commercial products, ...) locally installed on the development platform.

The primary goal fo such a glue package is to convert the management conventions and

```
macro AnapheTOP "" \
      CERN  "/afs/cern.ch/sw/lhcxx" \
      BNL   "/afs/rhic/usatlas/offline/external/lhcxx" \
      LBNL  "/auto/atlas/sw/lhcxx"
```

---

9 - Configuring a package

The first ingredient of a proper package configuration is the set of configuration parameters which has to be specified in a text file uniquely named _____

Other configuration parameters are also optionally inserted from the HOME and USER *context* requirements files

Typical examples of these query functions are:

(typically using shell commands):

- `CMTCONFIG` describes the current settings for producing binary objects. One default value is provided automatically by CMT, but generally project will override it to apply specific conventions.

be active and during the build of `B` , the tag named

10. 2 - Defining the user tags

The user configuration tags can generally be specified though various complementary mechanisms:

- CMTSITE and CMTCONFIG can be specified using standard shell commands (setenv, export, set)

  ```
  sh> export CMTSITE=CERN
  ```

-

11 - Working on a

```
------> Foo ok
```

```
> [g]make QUIET=1
------> (Makefile.header) Rebuilding constituents.make
------> (constituents.make) Rebuilding setup.make Linux-i686.make
setup.make ok
```

11. 3 - Working on a test or external application

It is also possible to work on a *testnal*

Ibenefi ifromtohe packages configur d usng oal

Then any user wanting to access the so-called *official*

```
========== 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}; latex $(doc)doc1.tex

../doc/doc1.ps : ../doc/doc1.dvi
MyDoc  cd ${doc};  cd= ../doc/doc2.tex

M2Doc :: ../doc/doc1.ps

../doc/do21.dvi : $(doc)doc1.tex
        cd ${docncies = ../doc/doc2.te21.6 Td (M2Doc  -21.6 Td (M2D.ps : ../doc/doc1.dvi )Tj
```

3.

13. 1 - The requirements file

13. 1. 1 - The general requirements syntax

option	*validity*	*usage*
`-s=`*`directory`*	any	switch to a new default directory

Then it is possible to change the default search location as well as to use a simplified wildcarding syntax:

```
library A -s=A *.cxx -s=B *.cxx
```

- `-s=A` means that next source files should be taken searched from `../src/A`
- `-s=B` means that next source files should be taken searched from `../src/B` . Note that this new specification is *not* relative to the previous `-s=A` but relative to the default search path `../src`
- `*.cxx` indicates that all files with a `.cxx` suffix in the current search path should be considered

It's also possible to select or exclude files using regular expressions from general wildcarding techniques:

```
library A -s=A -x=[0-9] *.cxx -s=B -k=^B *.cxx
```

- The exclusion specification `-x=[0-9]` added to the statement will exclude all files from `../src/A` containing a *number* in their name.
- The selection specification `-k=^B` added to the statement will select files from `../src/B` strictly starting with the `B` letter.

2.

When several constituents need to share source files, (a typical example is for building different libraries from the same sources but with different compiler options), it is possible to specify an optional output suffix with the `-suffix=<suffix>`

4.

For any constituent that has the `-target_tag` option set, a dedicated *tag* named `target_<constituent>` is automatically constructed by CMT. This tag becomes active during the construction of this constituent when using `make` , and therefore can be used as any other tag to select symbol values, or other configuration parameters.

Some languages (this has been seen for example in the IDL generators in Corba environments) do provide several object files from one unique source file. It is possible to specify this feature through the (repetitive) `-extra_output_suffix` option like in:

```
language idl -suffix=idl -fragment=idl -extra_output_suffix=_skel
```

symbol : *symbol-type symbol-name default-value* [*tag-expr value* ...]

symbol-type : *definition*

| *modification*

definition : macro

| set

| path

| action

| alias

modification : macro_prepend

| macro_append

| macro_remove

| macro_remove_regexp

| macro_remove_all

| macro_remove_all_regexp

| set_prepend

| set_append

| set_remove

| set_remove_regexp

| path_prepend

| path_append

| path_remove

| path_remove_regexp

tag-expr : *tag* [& *tag* ...]

●

13.2.7.1 - actions

Actions are one of the possible symbols. Their definition as said previously follow the generic conventions for any symbol type, and they implement the concept of a generic shell command.

An example of a symple action:

13. 2. 8 - use

Describe the relationships with other packages; the generic syntax is :

```
use <package> [ <version> [ <root> ] ]
```

Omitting the version specification means that the most recent version (ie. the one with

```
private
use C v1
use D v1
----------------
```

- all operations done in the context of package B will *see* both packages C and D
- all operations done in the context of package A will *see* both packages B and D, but not package C

13.2.9 - patterns

Often, similar configuration items are needed over a set of packages (sometimes over

In this example, an additional pattern (<other_sources>) permits the package to

```
ignore_pattern <name>
```

13. 2.10 - cmtpath_patterns

These patterns act quite similarly to the *global* patterns previously described, ie they defines a set of CMT statements to be applied in a generic way. The difference is that instead of being applied to *packages* , they are automatically applied to all entries in the CMTPATH list.

Only few system parameters can be used here:

-

```
<project>_prototypes
<project>_no_prototypes
<project>_with_install_area
<project>_without_install_area
<project>_config
<project>_no_config
<project>_root
<project>_no_root
<project>_cleanup
<project>_no_cleanup
```

---

13. 2.13 - setup_script, cleanup_script

Specify user defined configuration scripts, which will be activated together with the execution of the main `setup` and `cleanup` scripts.

The script names may be specified without any access path specification, in this case, they are looked for in the `cmt` or `mgr` branch of the package itself. They may also be specified without any `.csh` or `.sh` suffix, the appropriate suffix will be appended accordingly when needed. Therefore, when such a user configuration script is specified, `CMT`

The fragments defined in `CMT` can be:

A package declaring, and implementing a make fragment may override a fragment of

```
Jaki.h  ReadPDGtable.h  Tauola_i.icc  Taurad.icc  polhep.inc  tauola_cblk.inc
#-------------------------------------------------------------
```

The prototype header files (named <file-name>.ph) will contain prototype definitions for every global entry point defined in the corresponding C source file.

The effective activation of this feature is controled by the build strategy of `CMT`. The build strategy may be freely and globally overridden from any requirements

13. 3. 7 - cmt cleanup [-csh|-sh]

This command generates (to the standard output) a set of shell commands (either for

The second mode explicitly provides an alternate path.

A minimal configuration is installed for this new package:

Note that the search on clients is *not* performed recusively. Thus only clients

```
> cmt -public show uses
```

A typical output produced by this command is:

```
> cmt show uses
# use GaudiPolicy v*                              [1]
# use GaudiKernel v*
#   use GaudiPolicy v5r*                          [2]
# use CLHEP v*  (native_version=1.8.2.0)          [3]
#   use ExternalLibs v4r*
#
# Selection :                                     [4]
use CMT v1r16 (/afs/cern.ch/sw/contrib)
use ExternalLibs v4r2p0  (/afs/cern.ch/atlas/offline/external/Gaudi/0.12.1.5)    [5]
use CLHEP v2r1820p0  (/afs/cern.ch/atlas/offline/external/Gaudi/0.12.1.5)
use GaudiPolicy v5r11p2  (/afs/cern.ch/atlas/offline/external/Gaudi/0.12.1.5)
use GaudiKernel v13r5p1  (/afs/cern.ch/atlas/offline/external/Gaudi/0.12.1.5)
```

1. The first section of the display (up to the `Selection` keyword) displays the

1. hae`The -no_auto_import /R136 11 T`

One flavour of these scripts is generated per shell family (`csh` , `sh` and `bat`), yielding the following scripts :

```
setup.csh
setup.sh
setup.bat
cleanup.csh
cleanup.sh
```

Lastly, the actual behaviour of the prototype generator is defined in the standard make macro

Then the version tag is supposed to be moved forward, either increasing its minor identifier (in case of simple additions) or its major identifier (in case of changes).

The following actions should be undertaken then :

1. understand what is the latest version tag (typically by using the `cmt cvstags` command). Let's call it `old-tag` .

2. select (according to the foreseen amount of changes) what will be the next version tag. Let's call it `new-tag` .

3.

- provide a *nickname* for this external package,
- adapt the version tag convention consistently to the project, hiding the version tag

```
${CMTROOT}/mgr/cmt_install_action.sh
${CMTROOT}/mgr/cmt_uninstall_action.sh
${CMTROOT}/mgr/cmt_install_action.bat
${CMTROOT}/mgr/cmt_uninstall_action.bat
```

The default architecture of this installation scheme is by default set for each CMTPATH entry to:

```
<path>/InstallationArea/$(tag)/bin/...                [1]
                       /$(tag)/lib/...                [2]
                       /include/<package>/...         [3]
                       /share/bin/...                 [4]
                       /share/lib/...                 [5]
                             /...                     [6]
                       /doc/<package>/...             [7]
                           /...                       [8]
```

1. Platform dependent executables
2. Platform dependent libraries
3.

17. 2

These targets have to be specified as follows :

```
sh> gmake clean
sh> gmake Foo
```

18. 3 - Standard macros predefined in CMT

18. 3. 1 - CMT static

18.3.3 - Language related macros

These macros are purely conventional. They are expected in the various make fragments available from CMT itself for providing the various building actions.

During the mechanism of new language declaration and definition available in the CMT syntax, developers are expected to define similar conventions for corresponding actions.

Their default values are originally defined inside the requirements file of the CMT package itself but can be *redefined* by providing a new definition in the package's requirements file using the macro statement. The original definition can be *completed*

macro	*usage*

macro	*usage*
<*package*>_native_version	specifies the native version of the external package referenced by this *interface* package. When this macro is provided, its value is displayed by the `cmt show uses` command
<*package*>_export_paths	specifies the list of files or directories that should be exported during the deployment process for this package. Generally this is only useful for glue packages refering to external software
<*package* >_home	specifies the base location for external software described in glue packages. This macro is generally used to specify the previous one

<*type*>_<*constituent*>_cflags	specific C flags
<*type*>_<*constituent*>_pp_cflags	specific C preprocessor flags
<*type*>_<*constituent*>_cppflags	specific C++ flags
<*type*>_<*constituent*>_pp_cppflags	specific C++ preprocessor flags
<*type*>_<*constituent*>_fflags	specific Fortran flags
<*type*>_<*constituent*>_pp_fflags	specific Fortran preprocessor flags
<*constituent*>linkopts	provides additional linker options to the application. It is complementary to - aooptions ._ould not be tuefused withptiplic140 11 Tf 72.6 0T -13.2
<*type*>_<*constituent*	

`<PACKAGE>ROOT`	The access path of the package (including the version branch)

`<package>_root`

`use_cflags`	C compiler flags
`use_pp_cflags`	Preprocessor flags for the C language
`use_cppflags`	C++ compiler flags
`use_pp_cppflags`	Preprocessor flags for the C++ language
`use_fflags`	Fortran compiler flags
`use_pp_fflags`	Preprocessor flags for the Fortran language
`use_libraries`	List of library names
`use_linkopts`	Linker options
`use_stamps`	Dependency stamps
`use_requirements`	The set of used requirements
`use_includes`	The set of include search paths options for the preprocessor from the used packages
`use_fincludes`	The set of include search paths options for the fortran preprocessor from the used packages
`includes`	The overall set of include search paths for the preprocessor
`fincludes`	The overall set of include search paths options for the fortran preprocessor

18.3.9 - Utility macros

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

macro	*usage*	*default on Unix*
X11_cflags	default on	flags

The list of

`make_hosts`

tag name

CONSTITUENT	name of the constituent	<*language* > java jar make_header jar_header java_header library_header application_header protos_header library_no_share library application dependencies cleanup_header cleanup_library cleanup_application check_application document_header <document> trailer dsw_all_project_dependency dsw_project dsp_library_header dsp_shared_library_header

PROTOTARGET	prototype target

<

| -prototypes

| -preprocessor_command=*preprocessor_command*

| -fragment=*fragment*

| -output_suffix=*output-suffix*

| -extra_output_suffix=*extra-output-suffix*

library : library *library-name* [*constituent-option* ...]

[*source* ...]

action : action *action-name* [

| *set_append*

| *set_prepend*

| *set_remove*

| *set_remove_regexp*

tag : tag *tag-name* [*tag-name* ...]

tag_exclude : tag_exclude *tag-name* [*tag-name* ...]

tag-expr : *tag-name* [& *tag-name* ...]

use : use *package-name* [*version-tag* [*access-path*]]

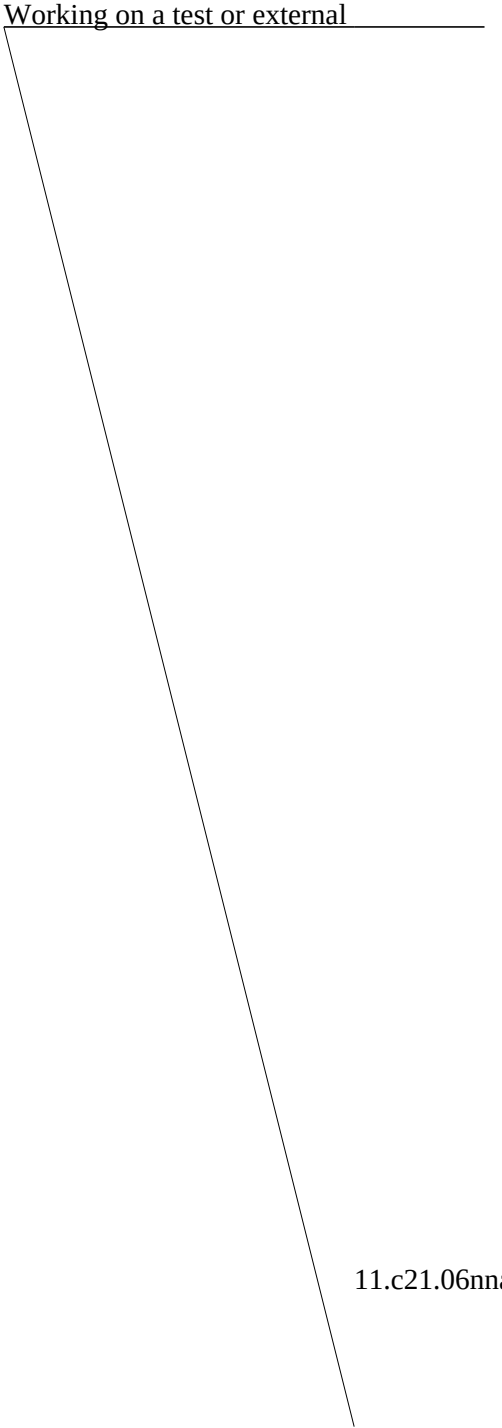
[*use-option*]

version : version *version-tag*

version-tag : *key version-number*

[*key release-number* [___

1. Find a location for working with temporary files. This is generally deduced from the `${TMPDIR}` environment variable or in `/tmp`



13. 2. 9. 1 _____

13. 3.17 cmt set version <version>

13. 3.18