- What is the impact of using CMT wrt SRT for a normal developer
  - Differences.
  - Some new concepts
- Preparing a release with CMT
- Forthcoming events
  - Tomorrow (thursday) morning : interactive session 40-R-D10
    - » Discussions, demos,
    - » interactive work : conversion of user packages?
  - March, 28th and 29th : workshop
    - » Tutorials
    - » Management issues
    - » Evaluation of the release
    - » Work model discussions

Package organization

- Packages are still the heart of the software decomposition.
  - What makes a package a CMT package?

    Prefix / *package* / *version-tag* / **cmt** / **requirements**
    - » Where **version-tag** takes the form

      letters **n** [*letters* **m** [*letters* **p**]]

      v1

      v1r2p3

      AGDD-00-00-37

      offline-01-03-01
- Logical vs physical structure
  - Packages can be hierarchized

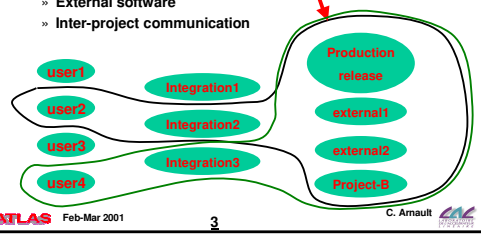
    somewhere/A/v1/cmt

    somewhere/A/B/v1/cmt

    somewhere/A/B/C/v1/cmt
  - A, B and C *are* CMT packages
  - There is *no* implied logical relationship between A, B and C
  - Logical relationships are only specified through the *use* constraints

Package organization

- General organization : software areas
  - The environment variable CMTPATH defines software areas where packages are searched for
    - » Production release
    - » Integration areas
    - » User areas
    - » External software
    - » Inter-project communication



Setting up CMT

- Minimal environment setup
  - For CMT itself :

    zsh> . /afs/cern.ch/sw/contrib/CMT/v1r7/mgr/setup.sh

    csh> source /afs/cern.ch/sw/contrib/CMT/v1r7/mgr/setup.csh
  - Site specific configurations : CMTSITE

    > export CMTSITE=CERN
  - Package areas : CMTPATH

    > export CMTPATH=area1:area2:area3
- Should be provided in the group login scripts (specific to each site)
- May also be specified in **~/requirements**

```
> cd
> cmt config (to be done once)
> . setup.sh
```

```
set CMTSITE CERN
path CMTPATH area1
path_append CMTPATH area2
path_append CMTPATH area3
```


The minimal HelloWorld

```
cd mydevarea
cmt create hello v1
cd hello/v1/src
vi hello.cxx
cd ../cmt
vi requirements
gmake check
Hello world
```

```
#include <iostream>
int main ()
{
  std::cout << "Hello world" << std::endl;
  return (0);
}
```

```
package hello

application hello -check hello.cxx
```


Individual package structure

package / *version* / **cmt** / **requirements**
/ **src** / *source-files*
/ **include** / *header-files*
/ *package* / *header-files*
/ **test** / *test-applications*
/ **classes** / *java-classes*
/ **scripts** / *scripts*
/ **i386-linux22** / *binary-files*
/ **VisualC** / *binary-files*
/ **insure** / *binary-files*
/ ... /

Only **../cmt** is required, all others branches are optional (conventional)

../src is considered by default as source location (if it exists)

Binary branches are automatically generated at build time, and take the form **../$(tag)**

> gmake

Or generated according to user tags

> gmake tag=insure

Configuration information is entirely specified inside the **requirements** file

The version tags

- **Basic definition**

  letters **n** [*letters* **m** [*letters* **p**]]

  - **n** : major or version id
  - **m**: minor or revision id
  - **p** : patch id

    v1
    v1r2p3
    AGDD-00-00-37
    offline-01-03-01

- **Issues in compatibility**
  - Different major ids means no compatibility
  - Backward compatibility for increasing minor ids
  - Full a-priori compatibility for different patch ids
- **Proposed convention for Atlas**
  - Use the current convention `<package>-nn-mm-pp`
  - Time scale : temporary for the migration or definitive?
  - Can be mixed with other conventions (TDAQ, Gaudi)

The binary tags

- **Principles, default values**
  - Computed in one shell script:
    ${CMTROOT}/mgr/cmt_system.sh
  - Provide default values for each OS (mix of `uname`, `fs sysname`, etc…)
- **Extending tag values or defining new tags**

  ```
  tag linuxdebug Linux debug
  ```

  - debug, optimized
  - Insure
  - User tags
- **Activating a user defined tag**

  ```
  > gmake tag=linuxdebug
  ```

  - Will select `debug` in addition to current `Linux` tag

Converting SRT packages

- **Everything goes into the `requirements` file in terms of CMT statements**
  - PACKAGE
    - » Author — `author, manager`
    - » Use — `use`
    - » Linkset — `macro <package>_linkopts`
  - GNUmakefile.in
    - » Library, application — `library, application`
    - » Make macros — `macro`
    - » Test procedures
  - configure.in
    - » #define xxx — `macro cpp_flags "-Dxxx"`
  - env.sh.in
    - » Export x=value — `set x "value"` / `path_append LD_LIBRARY_PATH`
  - Conventions, behaviors
    - » Include path — `include_path, include_dirs`
    - » Shared libraries, document generation — `make_fragment, pattern`
    - » language support — `language`

Converting SRT packages

- **Operation performed on offline-01-03-00**
  - Work done in
    /afs/cern.ch/atlas/project/art/CMTDev
  - Check out top directory of all packages tagged as offline-01-03-00
  - Retrieve the proper package version tag
  - Create the corresponding CMT package with exactly the same prefix
  - Fills in the use statements into the requirements files from similar statements found in the PACKAGE file
  - Complete the requirements files when possible from the previous work done on offline-01-02-xx (work originally done last summer). Note that will not be accurate for many packages.
  - Create a package AtlasRelease that contains the exhaustive list of these packages
  - Create a package AtlasPolicy that contains the Atlas conventions
  - This will be used to check in all requirements files as a startup step from which one can start tuning then in a realistic manner.
  - Project: to spend month of march to actually build a complete release based on CMT

An example

```
title         Muonbox
author        Marc Virchaux <virchaux@hep.saclay.cea.fr>
version       muonbox-05-05-01:20010115100133
includedir    include
use commons   00
use geant3    08-21
use atutil    00
use dice      00
use agetof    00-00-01 Tools
```

```
package muonbox
author        Marc Virchaux <virchaux@hep.saclay.cea.fr>
use commons commons-01-03-02
use geant3 geant3-21-08-33
use atutil atutil-01-43-00
use dice dice-03-21-11
use agetof agetof-00-00-08 Tools
ignore_pattern default_include_path
include_path $(MUONBOXROOT)/include
...
```


An example

```
# $Name: muonbox-05-05-01 $
# $Id: GNUmakefile.in,v 1.8 2000/06/06 12:15:17 laportej Exp $
################################################################
SUBDIRS = muonbox_uti muonbox_dig muonbox_rec main_code muonbox_dice \
muonbox_tup muonbox_amd muonbox_mat muonbox_ntuple muonbox_demo \
muonbox_covmat muonbox_resolu muonbox_cpp
...
```

```
package muonbox
...
branches muonbox_uti muonbox_dig muonbox_rec main_code muonbox_dice \
muonbox_tup muonbox_amd muonbox_mat muonbox_ntuple muonbox_demo \
muonbox_covmat muonbox_resolu muonbox_cpp
...
```

An example

```
# include files containing AGE structures
AGE_FILES  =   $(addprefix include/muonbox/, \
               dbox.inc ibox.inc mbox.inc sbox.inc tbox.inc vbox.inc)
TARGET_LIBS             = @HASH@ libmuonbox.a libmuonboxAgeStructures.a
FLAGS                   = -I. -I$(srcdir) -I$(srcdir)/include
TARGET_REMAKE_CONFIGURE = yes
TARGET_REMAKE_MAKEFILES = GNUmakefile


    package muonbox
    ...
    library muonbox -no_share ../muonbox_uti/*.F ../muonbox_dig/*.F \
    ../muonbox_rec/*.F ../main_code/*.F ../muonbox_dice/*.F \
    ../muonbox_dice/*.age ../muonbox_amd/*.F ../muonbox_mat/*.F \
    ../muonbox_ntuple/*.F ../muonbox_demo/*.F ../muonbox_covmat/*.F \
    ../muonbox_resolu/*.F ../muonbox_cpp/*.cxx

    Library muonboxAgeStructures -no_share -s=../include/muonbox \
    dbox.inc ibox.inc mbox.inc sbox.inc tbox.inc vbox.inc
```


An example

```
CPP_TIME                := $(shell date +%H%M)
CPP_DATE                := $(shell date +%y%m%d)
version                 := $(shell cd @srcdir@; srt show -v)
CPP_TITLE               := $(shell echo $(version) | sed -e 's%[-].*%;s%^$$%Unknown%')
CPP_VERS                := $(shell echo $(version) | sed -e 's%^[^-]*%%;s%^-
%%;s%^$$%Unknown%')

libmuonbox.a_USES        = f77 age c++
libmuonbox.a_CXXCPPFLAGS = $(FLAGS)
libmuonbox.a_FCPPFLAGS   = -DCPP_DATE="$(CPP_DATE)" -DCPP_TIME="$(CPP_TIME)" \
    ...                          -DCPP_VERS="'$(CPP_VERS)'"  -
DCPP_TITLE_CH="'$(CPP_TITLE)'" )
    library muonbox -no_share ../muonbox_uti/*.F ../muonbox_dig/*.F \
    ../muonbox_rec/*.F ../main_code/*.F ../muonbox_dice/*.F \
    ../muonbox_dice/*.age ../muonbox_amd/*.F ../muonbox_mat/*.F \
libmuonbox.a_OPTION_SETS = "/srcdir/*.OPTION_SETS
    ../muonbox_demo/*.F ../muonbox_covmat/*.F \
    ../muonbox_resolu/*.F ../muonbox_cpp/*.cxx

    Library muonboxAgeStructures -no_share -s=../include/muonbox \
    dbox.inc ibox.inc mbox.inc sbox.inc tbox.inc vbox.inc
```


An example

```
CPP_TIME                := $(shell date +%H%M)
CPP_DATE                := $(shell date +%y%m%d)
version                 := $(shell cd @srcdir@; srt show -v)
CPP_TITLE               := $(shell echo $(version) | sed -e 's%[-].*%;s%^$$%Unknown%')
CPP_VERS                := $(shell echo $(version) | sed -e 's%^[^-]*%%;s%^-
%%;s%^$$%Unknown%')

libmuonbox.a_USES        = f77 age c++
libmuonbox.a_CXXCPPFLAGS = $(FLAGS)
libmuonbox.a_FCPPFLAGS   = -DCPP_DATE="$(CPP_DATE)" -DCPP_TIME="$(CPP_TIME)" \
    ...                          -DCPP_VERS="'$(CPP_VERS)'"  -
DCPP_TITLE_CH="'$(CPP_TITLE)'" )
    library muonbox -no_share ../muonbox_uti/*.F ../muonbox_dig/*.F \
    ../muonbox_rec/*.F ../main_code/*.F ../muonbox_dice/*.F \
    ../muonbox_dice/*.age ../muonbox_amd/*.F ../muonbox_mat/*.F \
libmuonbox.a_OPTION_SETS = "/srcdir/*.OPTION_SETS
    ../muonbox_demo/*.F ../muonbox_covmat/*.F \
    ../muonbox_resolu/*.F ../muonbox_cpp/*.cxx

    Library muonboxAgeStructures -no_share -s=../include/muonbox \
    dbox.inc ibox.inc mbox.inc sbox.inc tbox.inc vbox.inc
```



```
# include files containing AGE structures
AGE_FILES  =   $(addprefix include/muonbox/,                \
               dbox.inc ibox.inc mbox.inc sbox.inc tbox.inc vbox.inc)

TARGET_LIBS             = @HASH@ libmuonbox.a libmuonboxAgeStructures.a
FLAGS                   = -I. -I$(srcdir) -I$(srcdir)/include

      package muonbox
TARGET_REMAKE_CONFIGURE = yes
TARGET_REMAKE_MAKEFILES = GNUmakefile
      author        Marc Virchaux <virchaux@hep.saclay.cea.fr>

CPP_TIME                := $(shell date +%H%M)
      use commons v2r1
CPP_DATE                := $(shell date +%y%m%d)
      use geant3 v2r1
version                 := $(shell cd @srcdir@; srt show -v)
      use atutil v2r1
CPP_TITLE               := $(shell echo $(version) | sed -e 's%[-
      use agetof v2r1 Tools
].*%;s%^$$%Unknown%')
      use CxxFeatures v2r1 Utilities
CPP_VERS                := $(shell echo $(version) | sed -e 's%^[^-]*%%;s%^-
%%;s%^$$%Unknown%')

      branches _kumacs cmz include main_code muonbox_amd muonbox_cde muonbox_covmat
      muonbox_demo muonbox_dice muonbox_dig muon
libmuonbox.a_USES        = f77 age c++
      box_mat muonbox_ntuple muonbox_rec muonbox_resolu muonbox_tup muonbox_uti src
libmuonbox.a_CXXCPPFLAGS = $(FLAGS)
libmuonbox.a_FCPPFLAGS   = -DCPP_DATE="$(CPP_DATE)" -DCPP_TIME="$(CPP_TIME)"
\     ignore_pattern default_include_path
      include_path $(MUONBOXROOT)/include
-DCPP_VERS="'$(CPP_VERS)'"  -
DCPP_TITLE_CH="'$(CPP_TITLE)'"  \
      apply_pattern default_no_share_linkopts
                                -DATLAS_TYPE \
                                -I$(srcdir)/include
```


Libraries and applications

- **They are the constituents of the packages**
  - **Libraries and applications convert a source file of given languages into object files using compilers**

```
library mylib mylib.cxx A/*.f B/*.c
application myapp -s=../test a.cxx b.cxx
macro driver_sources "dummy.cxx" SunOS "driverA.cxx"
library driver -no_share driver.cxx $(driver_sources)
```

  - **Document generators may perform any kind of operation**
    - » **Generating files**
    - » **Converting files**
    - » **Installing or moving files**

```
document installer install $(bin)*.so \
        installer_output=${PRODAREA}

> gmake install
```


Expert specifications

- **Defining new languages and new document generators**
  - **Packages may provide make fragments, each defining a specific document generator**
    - » **Code or documentation generation**
    - » ***Install* operations**
  - **They can be associated with new language extensions**
  - **make fragments are inherited through the use chain, and can be overridden by client packages**
- **Overriding default CMT behaviors (eg. For predefined languages)**
  - **All language-related behaviors are defined through the standard open mechanism (through make fragments) thus any package may override any language support**
- **Defining new project specific behaviors using patterns**
  - **A pattern is like a model for a (set of) CMT specifications. Patterns can be applied systematically to all packages or on demand only**

```
pattern default_include_path -global \
        include_path ${<PACKAGE>ROOT}/<package>
pattern default_library library <package> *.cxx
```

  - **Patterns are typically defined in the AtlasPolicy package**

Basic commands

- **Query operations**
  - **Visualize all specifications entered in the requirements file(s) of the current package and of the used packages**

```
> cmt show xxx
   macros, sets, constituents, path, fragments,
languages, patterns, etc.
```

- **Management operations**

`cmt create`

`cmt remove`

» **Package creation and removal**

`cmt config`

» **Restore the original state of the ../cmt subdirectory**

`cmt broadcast` *`shell-command`*

» **Propagate a shell command to the sorted hierarchy of used packages**

» **Accept number of selectors for sub-hierarchy**

- **Build operations**

`cmt build xxx` **: internal use only**