Table of Contents

Skeleton Code Generators **1**
- Goal of this page 1
- Introduction 1
- Available Generators 1
- Using the Generator 1
  - mkedanlzr 1
  - mkedfltr 2
  - mkedprod 3
  - mkesprod 3
  - mkedlpr 4
  - mkdatapkg 4
  - mktsel 4
  - mkrecord 5
  - mkskel 5
  - mktmpl 5
  - Skeleton package 7
- Review Status 7

Skeleton Code Generators

Complete:

Goal of this page

To introduce to the user scripts which generate basic code structure for the new EDM.

Introduction

When setting up code for the new EDM (such as creating a new EDProducer) there is a fair amount of 'boiler plate' code that you must write. To make writing such code easier CMS provides a series of scripts that will generate the necessary directory structure and files needed so that all you need to do is write your actual algorithms.

Available Generators

The following generators are available

- **mkedanlzr** : makes a skeleton of a package containing an EDAnalyzer
- **mkedfltr** : makes a skeleton of a package containing an EDFilter
- **mkedprod** : makes a skeleton of a package containing an EDProducer
- **mkesprod** : makes a skeleton of a package containing an ESProducer
- **mkrecord** : makes a complete implementation of a Record used by the EventSetup
- **mkdatapkg** : makes a skeleton of a package containing an DataPkg
- **mkedlpr** : makes a skeleton of a package containing an EDLooper
- **mkskel** : makes a skeleton of a package containing a generic C++ code
- **mktsel** : makes a skeleton of a package containing an TSelector

all of them are based on generic **mktmpl** script which is capable to generate any type of code from given template (see below). Every script shown above supports `-h` flag to invoke the help, e.g. `mkedprod -h`.

Using the Generator

The generators are available in your PATH after the scram runtime environment is setup (e.g., `eval` `` `scramv1 runtime -csh` ``).

mkedanlzr

How to makes a skeleton of a package containing an EDAnalyzer named <name>

- go to the subsystem directory where you want the new package. (You can make a new subsystem just by doing `mkdir <name>` in the project's `src` directory)
- `mkedanlzr <name>`. This will create a new directory named `<name>` which has the proper sub-structure.
- edit the newly create `<name>/plugins/<name>.cc` file

How to get examples added to the skeleton

- When running the `mkedanlzr` command add a option when generating the skeleton
  - Example: `mkedanlzr MyTrackAnalyzer example_track`

- this will add the appropriate entries to the BuildFile.xml, the necessary includes in the source file as well as a working example of how to use the data
- to use the new module, add the following (your can change the track label to your preferred tarcks) in your configuration file:

```
process.demo = cms.EDAnalyzer('MyTrackAnalyzer',
         tracks = cms.untracked.InputTag('generalTracks')
   )
```

Here we show the help output of the script which further explains its usage

```
shell# mkedanlzr -h

mkedanlzr script generates CMS EDAnalyzer code
Usage  : mkedanlzr MyAnalyzer -author "FirstName LastName" <example_track> <example_histo>
Output :
        MyAnalyzer/
        |  plugins/
        |  |-- BuildFile.xml
        |  |-- MyAnalyzer.cc
        |  python/
        |  |-- CfiFile_cfi.py
        |  |-- ConfFile_cfg.py
        |  test/
        |  doc/
Example:
        # create new EDAnalyzer code
        mkedanlzr MyAnalyzer
        # create new EDAnalyzer code with given author
        mkedanlzr MyAnalyzer -author "First Last"
        # create new EDAnalyzer code with track example
        mkedanlzr MyAnalyzer example_track
        # create new EDAnalyzer code with histo example
        mkedanlzr MyAnalyzer example_histo
```

mkedfltr

How to makes a skeleton of a package containing an EDFilter named <name>

- go to the subsystem directory where you want the new package. (You can make a new subsystem just by doing `mkdir <name>` in the project's `src` directory)
- `mkedfltr <name>`. This will create a new directory named `<name>` which has the proper sub-structure.
- edit the newly create `<name>/plugins/<name>.cc` file

```
shell# mkedfltr -h

mkedfltr script generates CMS EDFilter code
Usage  : mkFilter MyFilter -author "FirstName LastName"
Output :
        MyFilter/
        |  plugins/
        |  |-- BuildFile.xml
        |  |-- MyFilter.cc
        |  python/
        |  test/
        |  doc/
Example:
        # create new EDFilter code
        mkFilter MyFilter
        # create new EDFilter code with given author
        mkFilter MyFilter -author "First Last"
```

mkedprod

How to makes a skeleton of a package containing an EDProducer named <name>

- go to the subsystem directory where you want the new package. (You can make a new subsystem just by doing `mkdir <name>` in the project's `src` directory)
- `mkedprod <name>`. This will create a new directory named `<name>` which has the proper sub-structure.
- edit the newly create `<name>/plugins/<name>.cc` file

```
shell# mkedprod -h

mkedprod script generates CMS EDProducer code
Usage  : mkedprod ProducerName -author "FirstName LastName" <example_myparticle>
Output :
        MyProd/
        |  plugins/
        |  |-- BuildFile.xml
        |  |-- MyProd.cc
        |  python/
        |  |-- CfiFile_cfi.py
        |  |-- ConfFile_cfg.py
        |  test/
        |  doc/
Example:
        # create new EDProducer code
        mkedprod MyProd
        # create new EDProducer code with given author
        mkedprod MyProd -author "First Last"
        # create new EDProducer code with myparticle example
        mkedprod MyProd example_myparticle
```

mkesprod

How to makes a skeleton of a package containing an ESProducer named <name> which produces the space separated list of data <data to create> that resides in the EventSetup Record <record>.

- go to the subsystem directory where you want the new package. (You can make a new subsystem just by doing `mkdir <name>` in the project's `src` directory)
- `mkesprod <name> <record> <data to create>`. This will create a new directory named `<name>` which has the proper sub-structure.
- edit the newly create `<name>/plugins/<name>.cc` file

```
shell# mkesprod -h

mkesprod script generates CMS ESProducer code
Usage  : mkesprod MyProd RecName DataType1 DataType2 ...
         -author "FirstName LastName"

         MyProd = name of the producer
         RecName = name of the record to which the producer adds data
         DataType1 = data type created by the producer
Output :
        MyProd/
        |  plugins/
        |  |-- BuildFile.xml
        |  |-- MyProd.cc
        |  python/
        |  test/
        |  doc/
Example:
        # create new ESProducer code
        mkesprod MyProd
```

```
        # create new ESProducer code with given author
        mkesprod MyProd -author "First Last"
        # create new ESProducer code with given RecName
        mkesprod MyProd MyRec
        # create new ESProducer code with given RecName/DataType
        mkesprod MyProd MyRec MyType
```

mkedlpr

```
shell# mkedlpr -h

mkedlpr script generates CMS EDLooper code
Usage  : mkedlpr MyLooper -author "FirstName LastName"
Usage  : mkedlpr MyLooper RecName DataType1 DataType2 ...
         -author "FirstName LastName"

         MyProd = name of the producer
         RecName = name of the record to which the producer adds data
         DataType1 = data type created by the producer
Output :
        MyLooper/
        |  plugins/
        |  |-- BuildFile.xml
        |  |-- MyLooper.cc
        |  python/
        |  test/
        |  doc/
Example:
        # create new EDLooper code
        mkedlpr MyLooper
        # create new EDLooper code with given author
        mkedlpr MyLooper -author "First Last"
```

mkdatapkg

```
shell# mkdatapkg -h

mkdatapkg script generates CMS DataPkg code
Usage  : mkdatapkg MyDataPkg -author "FirstName LastName"
Output :
        MyDataPkg/
        |-- BuildFile.xml
        |  src/
        |  |-- classes.h
        |  |-- classes_def.xml
        |  interface/
        |  test/
        |  doc/
Example:
        # create new DataPkg code
        mkdatapkg MyDataPkg
        # create new DataPkg code with given author
        mkdatapkg MyDataPkg -author "First Last"
```

mktsel

```
shell# mktsel -h

mktsel script generates CMS TSelector code
Usage  : mktsel ProducerName -author "FirstName LastName" <example_track>
Output :
        MyTSel/
        |-- BuildFile.xml
        |  src/
```

```
        |  |-- MyTSel.cc
        |  interface/
        |  |-- MyTSel.h
        |  |-- classes.h
        |  |-- classes_def.xml
Example:
        # create new TSelector code
        mktsel MyTSel
        # create new TSelector code with given author
        mktsel MyTSel -author "First Last"
        # create new TSelector code with myparticle example
        mktsel MyTSel example_track
```

mkrecord

- go the `interface` directory of the package you want the record to be kept in
- `mkrecord <name>`
- `cp <name>.cc ../src`

```
shell# mkrecord -h

mkrecord script generates CMS C++ record code
Usage  : mkrecord MyRecord -author "FirstName LastName"
Output :
        MyRecord.cc
        MyRecord.h
Example:
        # create new C++ Record
        mkrecord MyRecord
        # create new C++ Record with given author
        mkrecord MyRecord -author "First Last"
```

mkskel

mkskel script allows you to create source and/or header files anywhere (within or outside of your package/module area). Typically, you visit your package interface area and create your desired header file, e.g. mkskel Foo.h. Later, you can create a new source file either within src or plugins area, e.g. cd src; mkskel Foo.cc. But sometimes it is desired to create both header and source files within one common directory, to do so you'll invoke mkskel script as following: mkskel Foo and it will generate both header and source files for you.

```
shell# mkskel -h

mkskel script generates CMS C++ Skeleton code
Usage  : mkskel MySkeleton -author "FirstName LastName"
Output :
        MySkeleton.cc
        MySkeleton.h
Example:
        # create new C++ Skeleton
        mkskel MySkeleton
        # create new C++ Skeleton with given author
        mkskel MySkeleton -author "First Last"
```

mktmpl

mktmpl is a general purpose script which can be used to generate any type of code from your template. For example if you'd like to generate CMS EDProducer package you can use **mktmpl** with the following set of options

```
mktmpl --tmpl=EDProducer --name=MyProducer
```

You can list available templates via the following command:

```
mktmpl --templates
```

Here is full list of supported options

```
shell# mktmpl -h
Usage: mktmpl [options]

Options:
  -h, --help           show this help message and exit
  --debug              debug output
  --tmpl=TMPL          specify template, e.g. EDProducer
  --name=PNAME         specify package name, e.g. MyProducer
  --author=AUTHOR      specify author name
  --ftype=FTYPE        specify file type to generate, e.g. --ftype=header,
                       default is all files
  --keep-etags=KETAGS  list examples tags which should be kept in generate
                       code, e.g. --keep-etags='@example_trac,@example_hist'
  --tdir=TDIR          specify template directory,
  --tags               list template tags
  --etags              list template example tags
  --templates          list supported templates
```

To feed **mktmpl** with your own template package you need to create a simple directory where you'll place your template files, e.g.

```
MyC++Template/
      Driver.dir
      Makefile
      c++11.cpp
      c++11.h
      main.cpp
```

here we show output of tree command for `!MyC++Template` directory with bunch of files (typical for generic C++ project). The only file which requires special attention is `Driver.dir`. Its content should list the desired directory stricture you'd like to create from your template package, e.g.

```
Makefile
src/c++11.cpp
src/main.cpp
include/c++11.h
```

Here we instruct **mktmpl** script to create in top level directory the Makefile, then create src/include directories and place over there desired source and header files respectively. Finally you can place your code to your template files and generate the new package as following:

```
mktmpl --tdir=$PWD --tmpl=MyC++Template --name=MyCode
New package "MyCode" of MyC++Template type is successfully generated
MyCode/
|-- Makefile
|  src/
|  |-- c++11.cpp
|  |-- main.cpp
|  include/
|  |-- c++11.h
Total: 2 directories, 4 files
```

Skeleton package

The **mktmpl** script is part of CMS Skeleton package. The package consists of Skeleton engine and set of pre-defined CMS scripts. Here we discuss several rules used by Skeleton engine for template parsing.

There are two types of tags supported by Skeletons package, template and example tags. Template tags can use any name which should be enclosed with double underscores, e.g. `__test__`, `__prodname__`, `__abc123__`. The example tags should start with `@example_` prefix followed by the name, e.g. `@example_track`.

In addition to template and example tags, user can write python snippets in their template, which should be enclosed with two statements: `#python_begin` and `#python_end`. PLEASE NOTE: you must use proper indentation for python snippets similar to normal python code requirements. For example:

```
#python_begin
     for idx in range(1,3):
         print "std::shared_ptr<int> myPointer%s;\n" % idx
#python_end
```

Here we used python loop to create several shared pointers, the Skeleton engine will parse this code and generate the following output

```
    std::shared_ptr<int> myPointer1;
    std::shared_ptr<int> myPointer2;
```

Event though we do not impose any restrictions on tag naming convention it is wise to use them appropriately, e.g. for your C++ class template you better use `__class__` and similar tag naming conventions.

Review Status

Reviewer/Editor and Date (copy from screen)	Comments
Main.chrjones - 28 Jul 2005	page author (Chris Jones)
JennyWilliams - 31 Jan 2007	editing to include in SWGuide
PedroParracho - 27 Oct 2008	Added links to definition of EDProducer, EDAnalyzer, EDFilter and EventSetup

Responsible: Main.chrjones
Last reviewed by: Sudhir Malik- 24 January 2009.

This topic: CMSPublic > SWGuideSkeletonCodeGenerator
Topic revision: r16 - 2015-12-12 - JuanEduardoRamirezVargas