# Table of Contents

# CMSSW Developer's Guide

%COMPLETE50%

## Introduction

This guide is intendend to orient developers in the world of CMSSW. The suggested solutions will not be the only ones. Please feel free to contribute to this page, in case you think there is a better way to do things.

## CMSSW source tree

The Source Tree is organized in two levels: `Subsystems` and `Package` . A package is identified by these two coordinates, as in `DataFormats/DetId`

## Git and GitHub

the CMSSW software development process is GitHub based. Please refer to CMSSW at GitHub for more information.

## SCRAM

Scram is our build and configuration management system. Useful wiki pages that describe the use of SCRAM with CMSSW are available :

- Mostly used SCRAM commands for CMSSW developers
- CMSSW SCRAM BuildFile tags/flags

## Integration Builds

As software is integrated in CMSSW, it gets tested in the integration build system. Results are available here

## Code conventions

Somewhat outdated CMS coding conventions document and SWGuideFrameWork

## Documenting your code

Your classes should contain the appropriate doxygen markup or a package level README.md file where.

### Add tests to your package

Authors of CMSSW packages are encouraged to have unit tests for their packages. A unit test is a software verification and validation method in which one can test if individual units of source code are fit for use. A unit is the smallest testable part of an application/code. The unit tests used by CMSSW Framework developers are provided in `FWCore/Framework/test` directory. In CMSSW, one runs the unit tests for a package by creating a working area, checking out that package, building it with scram, and then giving the command `scram b runtests`. A typical output from runing this command in the directory `FWCore/Framework/test` is shown HERE.

Unit tests are also run as part of the Integration Builds (IB). One can look at the unit tests performed on all the packages in a IB by looking at the CMSSW Integration build webpage. On this webpage click on any "details" in the "builds" column. This will bring another webpage where one can look (in the column `UnitTest logfile`) at the log file for the unit tests run for different packages as part of IB. Not all packages have a unit tests.

Another thing to pay attention is that IB infrastructure expects a 0 exit code for successful tests, and any other number indicates a failure. `cmsRun` applications already obey this rule. Developers writing their own scripts as tests should be aware of this. In the unit test out put HERE search for word "status" and see if a unit test ran fine i.e. has `status = 0`.

The unit tests should be defined as a bin target in the `BuildFile` file that is in the `test` directory of a package or a lower directory. All the .cc .cpp files that your unit test needs must also reside in the `test` directory. Look at the following as examples:

- `FWCore/Framework/test/BuildFile.xml`
- `FWCore/Integration/test/BuildFile.xml`
- `PhysicsTools/FWLite/test/BuildFile.xml`

Let us take couple of snippets from a BuildFile to see what they do/mean:

**The first snippet:**

```
<bin   name="TestFWCoreFrameworkView" file="View_t.cpp">
  <use   name="DataFormats/Common"/>
  <use   name="cppunit"/>
</bin>
```

The 'bin' part defines a new test that should be run where 'name' is going to be the name for the executable and 'file' contains a comma separated list of all files which should be compiled together to create the binary. The 'use' within the 'bin' tells scram what packages (DataFormats/Common) or external tools (cppunit) are needed to compile and link the executable.

**The second snippet:**

```
<bin   name="TestFWCoreFrameworkInputTagFailure" file="TestDriver.cpp">
  <flags   TEST_RUNNER_ARGS=" /bin/bash FWCore/Framework/test test_InputTag_cache_failure.sh
  <use   name="FWCore/Utilities"/>
</bin>
```

To know what this snippet means have look HERE.

Here are some links that may serve as examples of different kind of unit tests:

- test that runs a simple executable with a "main" function. The output of this test is HERE.
- test that uses the cppunit package which is an external product specifically designed to be used for unit tests. The output of this test is HERE.
- one that uses a shell script to run `cmsRun`. The output of this test is HERE.

The unit tests are expected to be fast. Less than a second is good, a few seconds is OK, but if it takes minutes to run (or longer), then it should probably be redesigned to run faster or not be a unit test.

# The Release Procedure

See this page, ReleaseSchedule, for information about release policy, and particular releases.

# FAQ

Here is the link to the FAQ page

# Review Status

| Reviewer/Editor and Date (copy from screen) | Comments |
|---|---|
| Main.argiro - 10 Apr 2006 | page created |
| BenediktHegner - 20 Dec 2006 | page content last edited |
| JennyWilliams - 07 Feb 2007 | twiki tidying |
| AndreasPfeiffer - 31 May 2007 | added ref to coding rules in SWGuide |
| AndreasPfeiffer - 25 July 2007 | updated Integration Builds (former "nightly") |
| IanTomalin - 28 Aug 2008 | Added "changing CMSSW version during development cycle" section |
| ShahzadMuzaffar - 14 Nov 2008 | Added ".admin/developers for CMSSW CVS access control" section |
| SudhirMalik - 13 April 2010 | Added description of the unit tests for a release |
| MiguelOjeda - 05 May 2011 | Updated several sections regarding TagCollector |
| MiguelOjeda - 19 July 2017 | Removed obsolete things and added github references |
| MiguelOjeda - 19 July 2017 | Removed obsolete things and added github references |
| GaelleBoudoul - 13 Feb 2018 | Changed BuildFile to Buildfile |

Responsible: Main.argiro
Last reviewed by: Reviewer

---

This topic: CMSPublic > SWGuideDevelopersGuide
Topic revision: r53 - 2018-02-13 - GaelleBoudoul