XMM

XMM-Newton: CCF testing and validation procedures

XMM-SOC-CAL-TN-0025 Version 1.1

C. Gabriel & G. Vacanti (XMM-SOC)

August 7, 2002

Revision history

Revision number	Date	Revision author	Comments
0.5	February 14, 2002	C. Gabriel	First draft
1.0	March 8, 2002	C. Gabriel	First version
Version 1.1	August 7, 2002	CG + GV	updated version

Contents

1 Introduction

This is a technical note which should eventually lead to a document containing the procedures for every individual CCF constituent production. In its actual form it only describes the general procedures for CCF releases, which have to be followed by the whole XMM Instrument Calibration Team.

This document supersedes the former procedures from ref.[1] (although the present document uses partly the contents of those procedures it has a wider scope and therefore replaces it).

The Current Calibration File (CCF) is a collection of files (constituents), each having a separate issue number and an independent update cycle. This is all described in the CCF ICD [2].

CCF constituents are as from Jan.21, 2002 under configuration control of the SAS-CCB. As such, whenever a new constituent is to be released a procedure has to be followed, as outlined below.

2 General CCF release procedure

The procedure for a CCF constituent release differs partly depending if the structure of the calibration file is affected (same as if it is a new defined file) or not.

2.1 Changes to CCF contents, not affecting the structure of the calibration file

These should be initiated i) via a CCR on CCF [SOC-VILSPA configuration Control Tracking Tools - Item CCF (3.2.05)], ii) the insertion of the new constituent into the CCF system, iii) the insertion of the Release Notes into the VILSPA documentation system (http://xmm.vilspa.esa.es/cgi-doc/ccb/DOC_page).

- The CCR must contain the CCF Release Note (which should have been put before into the documentation system). In the CCR the E-mail address (last entry in the bottom) should be sas-ccb@xmm.vilspa.esa.es. This is the way to notify all the SAS-CCB members the existence of a new CCF-CCR.

  The CCF Release Note has to follow the already used structure:

  1) **CCF components**: table with the new constituents to be added to the CCF
  2) **Changes**: specifying the changes wrt to the former version
  3) **Scientific Impact of this Update**
  4) **Estimated Scientific Quality**
  7) **Expected Updates**
  *) **References**: to whatever documents would be seen as relevant for it.

  with the add of a
  section 5) **Test procedures**, containing the steps and scripts to be used for testing of the particular CCF, and
  section 6) **Summary of the test results**, with the relevant plots and / or numerical output from the test procedures,

  The release notes should be inserted into the documentation system, at the same time of submitting an CCR, containing also attached those release notes (format: ps.gz). Also simultaneously the CCF constituents are put into the CCF development area.

  The SAS-CCB will dispose in its next following meeting about this CCF. In urgent cases (use of "Urgent" category in the CCR), CCFs can be approved within 2 working days. If it is approved, the secretary of the SAS-CCB will notify via e-mail the SOC CCB for closure of the CCR, and make the release notes visible from the calibration files pages.

- Insertion of the new constituent into the CCF devel area is done via a procedure described in Appendix A.

2.2 Release of a new defined CCF / Changes affecting the structure of a CCF

This kind of change makes necessary an update of the Calibration Access Layer (CAL) and imply therefore an SCR on it in the SAS Configuration Control System. (At the moment this system is outside the SOC VILSPA Configuration Tracking Tools, and will be kept for practical reasons as it is now for a while). It should be initiated also via CCR on CCF as under 2.1, with the difference that the release notes will be only partial and to be completed with the test report once the SAS SCR has been implemented.

The I/F to the SAS is controlled by the CAL HB [3]. Therefore such a change must be accompanied by a DCP to the CAL HB.

3 Test procedures: general considerations

Whenever a CCF constituent is produced it should be thoroughly tested. At least is required that a number of different tests with the SAS has to be exercised, despite of other tests which could be seen as necessary desirable.

- Using the SAS task CALVIEW:

  Look at the constituent values making sure that you change the input parameters in CALVIEW as to exercise a wide range of conditions. It it can be read and displayed then its structrue is correct and its contents are at least reasonable.

  Note: a static check of the new CCF constituents can be spotted by adding a so-called calviewable if needed. For this an SCR on SAS CAL is necessary.

- For spotting more subtle numerical errors and also to see the improvements the new constituent has eventually produced, it is necessary to process some data through the relevant SAS tasks for this CCF (eg. em/ep/rgsproc, omichain).

  A comparison between processed data with a CIF from the public CCF and data processed with the new CCF should show the improvements. This requires two runs with the same CIF, the second time using the command line CCF constituent override (ccffiles NEW_CCF1.CCF NEW_CCF2.CCF ...). You need to make the SAS noticeable of the existence of those new files by running a task "ccfextseq" (eg.: ccfextseq sets=' "NEW_CCF2.CCF.CCF" "NEW_CCF2.CCF).

4 Individual Testing procedures per CCF

TBW using eg inputs from release notes

A How to develop CCF constituents

This section addresses the issue of how to create CCF constituents, how to document their structure and keep it under version control.

A.1 Preliminaries

- The creation of constituents requires the SAS tasks **deceit** and **ccfextseq**. In the following I assume familiarity with the deceit data format. Make sure you also read the deceit documentation about the function `fillColumn`, because there it is explained how to format the many possible binary table types.
- Version control of the dataset structure is achieved via CVS. A basic knowledge of CVS is assumed. There is an excellent introduction to CVS here: `http://www.cvshome.org/docs/`.
- The make file structure described below only works with GNU make. Try `make -v` to see whether you're using the correct make. Also note that some FTOOLS configuration files set the environment variable `MAKE`. Check if this is your case, and if it is unset that variable.

A.2 Version control

Version control of the CCF components is achieved through version control of the structure of the components and the data used to fill them in. The actual CCF constituents are not kept under version control. They are created and kept in an archive at the SOC.

The structure of the CCF constituents is described through **deceit** templates, kept under version control in a CVS repository. The data files (usually plain text files) needed to fill the data structures are also kept under version control in the same repository. The actual CCF datasets are not stored in the same repository.

The CCF is broken down into packages. Each package addresses one aspect of the XMM-Newton calibration. How the CCF constituents are divided up into packages is an important issue that I address in the following section. Here I describe a general infrastructure to create CCF constituents through a high level description of their contents.

A.3 The environment

This configuration applies to xvsas01.vilspa.esa.es.

1. Login, and source `~ccflib/ccf-setup.csh`. This will set the correct version of the SAS, define CCFDEV and CVSROOT as required.
2. The first time you need to check out one or all of the CCF packages. Choose a directory of your liking (for instance `~ccf`) and do for instance:

   ```
   cd ~ccf
   cvs checkout ccfdev/packages/emos-quantumeff
   ```

   If you want to check them all out:

   ```
   cvs  checkout ccfdev/packages
   ```

   The checkout command must be issued from the directory `~ccf`.
3. now you can start working on your CCF constituents.

A.4 Detailed example

Let us create a dataset, call it `r1`. First create a dataset description file (`r1.desc`). This is reproduced below:

```
ISSUE 1
TYPE aduconv
SCOPE rgs1
CATEGORY xmmccf
VALDATE 1998-01-01T00:00:00
EVALDATE 2010-10-10T01:02:03
```

Note: `EVALDATE` is optional. If it is not specified none will be used, and this is taken to mean that no end of validity date applies to the CCF consituent in question.

The description file simply contains the attributes defining a CCF constituent. The first three define a CCF constituent as per the CCF ICD. The category field can be one of xmmccf or scisimccf. For production CCF constituents use only xmmccf.

The create a make file (`Makefile`):

```
DATASETS = r1
include $(CCFDEV)/Make.include
```

Now type: `make depend; make`. This will create and empty CCF file: `RGS1_ADUCONV_0001.CCF`.
This file is rather useless. Let's say that it contains two blocks. Edit `Makefile` to add the following:

```
r1_COMPONENTS = aducoeff gain
```

Now edit `aducoeff.dct` and `gain.dct`. They can contain any block definition in **deceit** format. In our case, `aducoeff.dct` is:

```
table "ADUCOEFF" rows("aducoeff.dat") ""
<
  column "CCD_ID"  int8 "CCD identifier"      ""         fillColumn(col, "aducoeff.dat", 0)
  column "NODE_ID" int8 "CCD node identifier" ""         fillColumn(col, "aducoeff.dat", 2)
  column "OFFSET"  real32 "ADC offset"        "chan"     fillColumn(col, "aducoeff.dat", 3)
  column "GAIN"    real32 "ADC gain"          "eV/chan"  fillColumn(col, "aducoeff.dat", 4)
>
```

And `gain.dct` is:

```
table "GAIN" rows("gain.dat") ""
<
 column    "CCD_ID" int8   "CCD identifier" "" fillColumn(col, "gain.dat", 0)
 column(2) "GAIN"   real32 "Column gain"    "" fillColumn(col, "gain.dat", 2, ":")
 column(2) "OFFSET" real32 "Column offset"  "" fillColumn(col, "gain.dat", 1, ":")
>
```

You also need to provide the required data files `aducoeff.dat` and `gain.dat`.
Type `make` again. The CCF file will look more useful:

```
[src] fstruct RGS1_ADUCONV_0001.CCF
  No. Type     EXTNAME      BITPIX Dimensions(columns)      PCOUNT  GCOUNT

   0  PRIMARY                   8     0                         0    1
   1  BINTABLE ADUCOEFF         8     10(4) 18                  0    1
   2  BINTABLE GAIN             8     17(3) 9                   0    1

[src] flcol RGS1_ADUCONV_0001.CCF
___Column_Names_________Formats______Dims______Units___
CCD_ID                  B
NODE_ID                 B
OFFSET                  E                   chan
GAIN                    E                   eV/chan

___Column_Names_________Formats______Dims______Units___
CCD_ID                  B
GAIN                    2E        (2)
OFFSET                  2E        (2)
```

Often, you'll want to create two CCF files that share the same structure but require different ASCII input files. It is possible for two or more datasets to share the same `COMPONENTS` as follows.
Edit `Makefile`:

```
DATASETS = r1 r2
r1_COMPONENTS = aducoeff gain
r2_COMPONENTS = aducoeff gain
include $(CCFDEV)/Make.include
```

Write `r1.desc` and `r2.desc` as required. Write the component files as follows.
`aducoeff.dct` becomes:

```
table "ADUCOEFF" rows("@dataset@aducoeff.dat") ""
<
  column "CCD_ID"  int8 "CCD identifier"      ""         fillColumn(col, "@dataset@aducoeff.dat", 0)
  column "NODE_ID" int8 "CCD node identifier" ""         fillColumn(col, "@dataset@aducoeff.dat", 2)
  column "OFFSET"  real32 "ADC offset"        "chan"     fillColumn(col, "@dataset@aducoeff.dat", 3)
  column "GAIN"    real32 "ADC gain"          "eV/chan"  fillColumn(col, "@dataset@aducoeff.dat", 4)
>
```

And `gain.dct` becomes:

```
table "GAIN" rows("@dataset@gain.dat") ""
<
 column    "CCD_ID" int8   "CCD identifier" "" fillColumn(col, "@dataset@gain.dat", 0)
 column(2) "GAIN"   real32 "Column gain"    "" fillColumn(col, "@dataset@gain.dat", 2, ":")
 column(2) "OFFSET" real32 "Column offset"  "" fillColumn(col, "@dataset@gain.dat", 1, ":")
>
```

During the make stage, the token `@dataset@` will be replaced by `_r1_` or `_r2_`, depending on whihc dataset is being made. In our case, you would have to have the following ASCII data files: `_r1_gain.dat`, `_r1_aducoeff.dat`, `_r2_gain.dat`, `_r2_aducoeff.dat`.

This is all you have to know to create CCF constituents. If you look at the FITS header, you'll see that behind the scenes the datasets are created with all the correct CCF attributes. For instance:

```
[src] fdump RGS1_ADUCONV_0001.CCF
...
FILENAME= 'RGS1_ADUCONV_0001.CCF'
TELESCOP= 'XMM     '           / mission name
SCOPE   = 'RGS1    '           / calibration constrituent scope
TYPEID  = 'ADUCONV '           / calibration identifier
ISSUE   =                    1 / issue number
VALDATE = '1998-01-01T00:00:00' / start of validity date
EVALDATE= '2010-10-10T01:02:03' / end of validity date
CATEGORY= 'XMMCCF  '
...
```

A CCF package can be structured in subdirectories. Use the `SUBDIRS` target in the make file. You must have the files `VERSION` and `ChangeLog` in the top level directory.

Do not forget to commit your changes to the CVS repository.

A.5 Version and issue number

Care should be taken not to confuse the *issue* number of a CCF constituent, and the *version* number of a CCF package. The two need not be the same.

Each time a new CCF package is created and uploaded, its version number (in the file `VERSION`) must be increased. The version number is made up of two integers `M.m`, called the major and the minor version number. Change the major version number when the structure of the CCF constituents in a package changes. Otherwise continue increasing the minor version number.

The issue number is attached to the individual CCF constituents. It is an integer, and is always increased by one unit at a time.

A.6 Words of wisdom

- It is important that CCF constituents have all the same *look and feel* about them. Consistent naming conventions for columns, blocks and attributes should be followed.
- We should be able to distinguish between the CCF used to analyze SciSim data and the real one. The `CATEGORY` in the description file allows you to do this. For SciSim data files a different file naming convention is applied, by adding the word `SCISIM` to the names of the constituents.

A.7 CCF packages

As hinted at above, I suggest that the CCF production be split in packages by scope/type. Scopes should be grouped by instrument: rgs for rgs1 and rgs2, xrt for xrt1, xrt2 and xrt3, emos for emos1 and emos2, pn for pn.

Having initialized the environment to work on the production of CCF files, do (for instance):

```
cvs checkout ccfdev/packages
cd ccfdev/packages
mkdir rgs-aduconv
cvs add rgs-aduconv
cd rgs-aduconv

[make more directories, edit make files, etc.]

cvs add <whatever you have added>
cvs commit <from the top level package directory>
make package <create a tgz package that can be uploaded>
upload package-name
```

A.8 The evolution of the CCF and the CAL

We can expect the CCF and the CAL to evolve separately. On the one hand, given a certain CCF structure, better algorithms may become available. On the other hand, changes in the instruments may warrant changes in the CCF components and correspondingly changes in the CAL.

The CAL must be able to determine what algorithm to use given a certain CCF. For this we make use of the keyword `ALGOID`.

Ideally the most recent CAL should always be able to process any past CCF constituents[1].

A.8.1 Invalid CCF constituents

There can happen that a number of public CCF constituents are found either to be wrong, or to have been invalidated through a backward incompatible change in the CAL. Besides making sure that this piece of information reaches the public, you should also communicate the list of invalid CCF constituents to the CAL developer, that will update the CAL CCF configuration file. Through this file the usage of an invalid CCF constituent triggers a warning in the SAS.

A.9 Notes about CVS

- Before you decide what files should be added to the CVS repository, make sure that you have done a `make clobber`. Only files that are left after a make clobber need to be stored. The other files will be regenerated as needed.
- If you decide to store binary files in the archive (for instance: fits files from which data are loaded into the CCF files), you must add them so: `cvs add -kb something.fits`.
- You have made some changes, and you have possibly added some new files that you go in CVS. How do you know? Do first a `make clobber`, then `cvs -n up`. Any file that is listed with a `?` next to it is not in CVS.

A.10 Available make targets

In make-speak, a target is the word you write after the `make` command itself. The default target, is the target that is *made* when no target name is given on the command line.

[1] This cannot always be achieved without complicating the CAL more than it is probably worth doing. Consult with the CAL maintainer to see if and when backward compatibility should be broken.

- `datasets`: this is the default target. It assembles the `deceit` templates, it creates the CCF constituents, and it runs `ccfextseq`.
- `depend`: builds the dependencies between `deceit` templates and data files.
- `clobber`: cleans up.
- `package`: creates a tar-gzipped archive of the current package.
- `upload`: as above, plus the changes are committed to CVS and tagged, plus the package is uploaded.
- `check`: it executes `clobber`, `depend`, `datasets`, plus it checks that the package and/or the created CCF constituents have not yet been delivered.
- `commit`: it commits changes to CVS. It issues a warning if some of the files in the package have not been committed to CVS.
- `update`: it updates the current package from the CVS repository.
- `info`: it lists a number of variables important to make..

B Check list

Once you have checked out the package you intend to work on then:

1. Update the ASCII data files with the correct values:
2. increment issue number by one (in *.desc) This file contains also the VALIDITY DATE (valdate), change it if necessary (eg. new bad pixels from a certain date appearing.)
3. increment version number by one (in VERSION) is just one line with: M.m If the CCF constituent has changed structure then M is incremented. If only the numerical contents of the constituent have changed then m is incremented.
4. update ChangeLog. Note the first line of the updates should contain in one line a text like the following Example:

```
****
ChangeLog for rgs-badpix
------------------------
Version 1.3 - 2001-05-22 (CG)
-----------
        + RGS1 issue 5, RGS2 issue 5 (using same issue, since former not in)
        + Wrong input file used for 1.2 - fixed
Version 1.2 - 2001-05-19 (CG)
-----------
...
****
```

5. create the new constituents:

```
make clobber
make depend
make
```

6. Use the new CCF in the processing of a few typical data.
7. Write a release note including the test results.

   Examples of release notes are in the xvsas01 machine, under ~cgabriel/documents/CCF/relnotes/. You first write the corresponding release note, "latex", "dvips" and "gzip" it.

8. Submit the release note into the XMM-Newton CCB documentation system.
9. Write the CCR (Item CCF) taking into account following points:
   - the release note should be included as **a gzipped postscript file**,
   - the e-mail address in the CCR page should be: **sas-ccb@xmm.vilspa.esa.es**,
   - in the "Description of change" field please include the string [**Public**] if this should become a public release note (this would cause that the RN gets automatically copied to its final public destination). A link to the RN from the calibration pages will be first activated once the SAS-CCB has given its approval.

   Submit it.

References

[1] Christian Erd. Procedures of updating CCF components. XMM-PS-TN-37, June 30, 2000.

[2] ESA. Interface Control Document for the XMM Current Calibration File. XMM-SOC-ICD-0005-SSD, ESA/SSD, Nov 1998 Issue 3.3

[3] Christian Erd et al. Calibration Access and Data Handbook. XMM-PS-GM-20, ESA/SSD, Jan 14 2000.