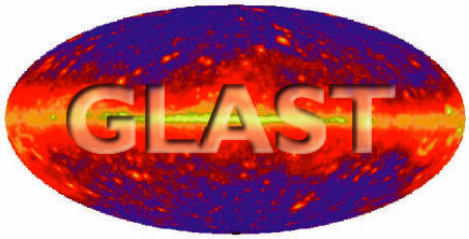

LAT Flight Software

FSW Build / Test / Export Process

Type: Discussion Document
Version: V0-0-0
Author: A.P.Waite
Created: 18 January 2002
Updated: 21 March 2007
Printed: 30 October 2008

How to build, test, and export a FSW release.

Contents

0 Overview 1
0.0 Five Easy Pieces 1
1 Before The Build 2
1.0 Identifying The Build 2
1.1 Garner Resources 4
2 The Build 5
2.0.0 Setting Up The Build Instance 5
2.0.1 Setting Up CMX In The New Instance 5
2.0.2 Building CMX In The New Instance 5
2.0.3 Populating The Build Instance 6
2.0.4 Running The Build 6
2.0.5 Checking The Build 7
3 Testing The Build 8
3.0 Sanity Check 8
3.1 Take a Deep Breath 8
3.2 Upload To Testbed / Test Stands 9
3.3 Test Again From Internal Resources 11
4 Annotating The Build 12
5 Exporting The Build 14
5.0 Exporting The “Other Stuff” 14

Figures

Figure 1 Listing of what packages have changed in a build 3
Figure 2 Listing of what flyable constituents have changed in a build 4
Figure 3 Setting up CMX in a new build instance 5
Figure 4 Building CMX in a new build instance (linux) 6
Figure 5 Building CMX in a new build instance (sun) 6
Figure 6 Populating a new build instance 6
Figure 7 Running a build (sun) 7
Figure 8 Running a build (linux) 7
Figure 9 Spying the progress of a build (sun) 7
Figure 10 Spying the progress of a build (linux) 7
Figure 11 Importing a build into FMX 9
Figure 12 LICOS configuration file for uploading a build to EPUs 10
Figure 13 LICOS configuration file for uploading a build to an SIU 11
Figure 14 Products associated with a build 14

Tables

Error! No table of figures entries found.

0 Overview

Building a FSW release has historically been a very hand held operation. It has not been until recently that process has settled down to the point where it could legitimately be made the subject of a procedure. It may be argued that it isn't at that point yet (and this document will be littered with examples of where the process could be improved), but it is certainly getting to the point that someone other than myself and a collection of experts could do it.

0.0 Five Easy Pieces

The process can be broken down into five steps:

- Before the build.
- The build.
- Testing the build.
- Annotating the build.
- Exporting the build.

The following sections will deal with each step in turn.

1 Before The Build

1.0 Identifying The Build

This is probably the most familiar step of the five, and certainly generates the most mail/meetings/… . To date the upshot of all the meetings is that we simply snapshot all the current production versions of all FSW packages into the build. This includes those packages for which FSW plays host, but which are arguably, not really flight software:

- FSW “ground” packages (also referred to as “Q” packages because the packages all begin with the letter “Q”). While it is true that, by rule, no “Q” package can contribute a flyable constituent, “Q” packages often need to cooperate with flyable constituents (usually by virtue of a data format specification), so it quite legitimate for “Q” packages to be part of a build.
- Virtual spacecraft (project VSC) packages. For these packages, CMX plays more of a “hosting” role, providing a convenient (package level) build environment. It is by now rare that there are any direct correlation between VSC packages and FSW packages, but each FSW build continues to include all the VSC packages.

It has certainly been very convenient to snapshot all production branches, and some home-grown tools actually leverage that fact. If we ever go to a non-all-production build (something I do not advocate because it starts to smell of code branching), it may be necessary to upgrade some tools.

As part of the identification process, I usually send round a build “pre-announcement” containing:

- A list of all package changes (added, removed or modified).
- A list of changed flyable constituents.

This serves three purposes:

- Gives developers a chance to verify their contribution, (or if you prefer…).
- Shakes the packages out of the trees.
- Puts developers on notice to get any changes to the *secondary boot scripts* into the system.

Developers should certainly know what a secondary boot script is, but even developers may not be aware of how central the secondary boot scripts are to the *export* step. An accurate export of FSW to I&T or ISOC is critically dependent on the secondary boot scripts being correct. That’s why I put that request for developers to feed back any changes to the FTS package (the package

holding the secondary boot scripts) as part of my pre-announcement.

The first list I publish (package changes) can be generated as shown in the following example:

- It compares two named instances.
- The instance name “slac” refers to the “all current production” instance (at SLAC).
- I have deleted a few lines to make it fit neatly on the page.

```
flora01:apw> cmx_differ B0-6-8 slac
================================================================================
Packages removed
--------------------------------------------------------------------------------
<none>
================================================================================
Packages added
--------------------------------------------------------------------------------
QSD
          DUMP      V0-0-0
VSC
          OCS1553   V0-2-2
================================================================================
Packages modified
--------------------------------------------------------------------------------
APP
          LRA       V0-1-0    V1-0-0
          LSE       V1-0-3    V1-0-4
FST
          FSWINI    V6-0-4    V6-0-6
          NBTLMV    V3-0-0    V3-0-1
          STU       V0-1-8    V0-1-9
ICS
          LATC      V7-6-3    V7-6-5
          LCFG      V0-0-1    V0-0-2
          RIM       V0-0-2    V0-0-4
          THS       V1-4-1    V1-5-0
QSD
          QCFG      V0-0-1    V0-0-2
          QPIG      V1-0-0    V1-0-1
          QSE       V1-2-3    V1-3-2
          QSEP      V1-3-0    V1-3-1
SYS
          FTS       V0-1-6    V0-1-7
          LCAT      V1-11-1   V1-11-3
          VPI       V2-8-0    V2-8-1
VSC
          VSCHW     V2-9-1    V2-9-2
```

Figure 1 Listing of what packages have changed in a build

The second list I publish (flyable constituent changes) can be generated as shown in the following example:

- It cross-checks that the CMX instance named in the command is indeed the current session’s CMX instance. In this case we want to generate a report of what would happen

if “all current production” was imported into FMX. “All current production” is, in effect, the CMX instance “slac”.

- Analysis is based on the secondary boot scripts, so they must be accurate.
- To avoid a mistake we have made before, the `fmx import` command has been restricted to running on Sun hosts.
- Without a special option, the `fmx import` command always operates in “dry-run” mode. It does not take any real action, just generates a report.

```
flora01:apw> fmx import slac
Import CPU_DB    V0-2-4    cpu_db_server           relocateable  : (dry-run)
Import CPU_DB    V0-2-4    cpu_epu                 cdm           : (dry-run)
Import CPU_DB    V0-2-4    cpu_siu                 cdm           : (dry-run)
Import EDS       V2-3-3    eds                     relocateable  : (dry-run)
Import ITC       V3-5-1    itc                     relocateable  : (dry-run)
Import LATC      V7-6-5    latc                    relocateable  : (dry-run)
Import LATC      V7-6-5    latc_cmn                relocateable  : (dry-run)
Import LDT       V0-1-1    encdec                  relocateable  : (dry-run)
Import LSE       V1-0-4    lsew                    relocateable  : (dry-run)
Import PBC       V4-12-0   epu_e_promimage.objz    pbc           : (dry-run)
Import PBC       V4-12-0   epu_f_promimage.objz    pbc           : (dry-run)
Import PBC       V4-12-0   pbc                     relocateable  : (dry-run)
Import PBC       V4-12-0   siu_e_promimage.objz    pbc           : (dry-run)
Import PBC       V4-12-0   siu_f_promimage.objz    pbc           : (dry-run)
Import THS       V1-5-0    ths                     relocateable  : (dry-run)
Import FTS       V0-1-7    siu.fmx.template        sbs           : (dry-run)
Import FTS       V0-1-7    epu.fmx.template        sbs           : (dry-run)
```

Figure 2 Listing of what flyable constituents have changed in a build

1.1 Garner Resources

I usually hand this step off to Kim.

At this stage, the only action necessary is to ask SCCS for a new *AFS volume*. Each of our builds is built on a new AFS volume. We now have ~40 such volumes. We ask SCCS to give the volume a “trivial” volume name like:

`G.glast.fli.cmx.cmx000`

(replacing the last three digits with a simple ascending number), and to mount the volume in a standard place:

`/afs/slac.stanford.edu/g/glast/flight/cmx`

We are in the process of improving this step. Instead of keeping all our builds going back as far as the ISIS build, we are converting to a set of rotating disks with the same style of AFS volume names and directory names. When a new build comes along, the oldest disk is archived using the MSTORE system and is then made the new build disk.

Kim should annotate how to use MSTORE and where the archives are located.

2 The Build

2.0.0 Setting Up The Build Instance

Assuming a build name of `Bx-y-z` and that an instance volume has been acquired from SCCS, something like:

`/afs/slac.stanford.edu/g/glast/flight/cmx/cmx123`

(referred to in what follows as `<stem>`).

These instructions are a shortened version of the instructions in the CMX manual for setting up a new CMX host/instance. For more detail, please see the CMX manual.

2.0.1 Setting Up CMX In The New Instance

This could be done from either a Sun box or a Linux box. The environment variable `CMX_I_CMX` and `CMX_C_DEV` must be legitimate (it almost always is for a flight software developer by virtue of the common group login file).

```
flora01:apw> cd <stem>
flora01:apw> mkdir –p DAQ/source/CMX/CDB
flora01:apw> setenv CVS_RSH ssh              <- If you don’t already have it
flora01:apw> cp $CMX_I_CMX/cmt/INSTALL_CMX .
flora01:apw> ./INSTALL_CMX Bx-y-z nodevelopment
.
.
<lots of output>
.
.
flora01:apw> cp $CMX_C_DEV/tag.db.slac ./DAQ/source/CMX/CDB/tag.db.Bx-y-z
flora01:apw> cp $CMX_C_DEV/arch.db.slac ./DAQ/source/CMX/CDB/arch.db.Bx-y-z
```

Figure 3 Setting up CMX in a new build instance

2.0.2 Building CMX In The New Instance

CMX itself has some compiled components, so the following must be done on each of Linux and Sun. The Sun is a little more complicated because that’s the only platform on which we can cross-compile:

```
noric01:apw> cmx set instance Bx-y-z
noric01:apw> cmx build CAB –-all --prod
noric01:apw> cmx build CMX –-all --prod
.
.
<lots of output>
.
.
noric01:apw>
```

Figure 4 Building CMX in a new build instance (linux)

```
flora01:apw> cmx set instance Bx-y-z
flora01:apw> cmx build CAB –-all --prod
flora01:apw> cmx build CMX –-all --prod
.
.
<lots of output>
.
.
flora01:apw> cmx set vxworks –-architecture=x86
flora01:apw> cmx build CAB –-all --prod
flora01:apw> cmx build CMX –-all --prod
.
.
<lots of output>
.
.
flora01:apw> cmx set vxworks –-architecture=ppc
```

Figure 5 Building CMX in a new build instance (sun)

2.0.3 Populating The Build Instance

This can be run from either a Sun or Linux host:

```
noric01:apw> cmx set instance Bx-y-z
noric01:apw> cmx populate project –-directory=<stem>
noric01:apw> cmx populate package
```

Figure 6 Populating a new build instance

2.0.4 Running The Build

There's a very common pratfall at this point.

Certainly when I do this, I have two terminal sessions open, one each for linux and sun. Only one is used to populate all the packages. The other session has not yet learned about all this new material. Try to build in this environment, and the ignorant session gets very upset.

To avoid this, I now standardly invoke `cmx stop` and `cmx set instance Bx-y-z` in both sessions. Now they're talking the same language and are prepared for the build step.

This requires two sessions, one on Sun and one on Linux (and the two builds can be run in parallel). Whatever the host type, the instructions are the same:

```
flora01:apw> cmx set instance Bx-y-z
flora01:apw> cd $CMX_C_CDB
flora01:apw> cmx rebuild –-production –-noregister &
```

Figure 7 Running a build (sun)

```
noric01:apw> cmx set instance Bx-y-z
noric01:apw> cd $CMX_C_CDB
noric01:apw> cmx rebuild –-production –-noregister &
```

Figure 8 Running a build (linux)

To spy the progress of the build (and these commands must be issued from the exact host machine on which the build was initiated):

```
flora01:apw> cmx set instance Bx-y-z
flora01:apw> cd $CMX_C_CDB
flora01:apw> tail –f sun4x_59_rebuild.log
```

Figure 9 Spying the progress of a build (sun)

```
noric01:apw> cmx set instance Bx-y-z
noric01:apw> cd $CMX_C_CDB
noric01:apw> tail –f i386_linux24_rebuild.log
```

Figure 10 Spying the progress of a build (linux)

The Linux build will complete in about an hour. The Sun build takes more like five hours.

2.0.5 Checking The Build

This is just a simple sanity check. Once the builds have completed, I pick up the log files and check for instances of the strings “error” and “warning”. There are a few well-known instances which can be ignored (for instance, the GNU `readline` package is based on code that was written so long ago that a modern compiler finds plenty of things to complain about).

More recently, our builds (in particular host builds) have become cluttered with assorted compiler warnings. This is a result of host compiler upgrades, resulting in a much fussier compiler. So far, we have not detected any instances where these warnings have indicated truly broken code, but they are annoying because they tend to litter the build log, making genuine problems more difficult to see. We have an outstanding JIRA to allow us to fix these annoyances up (it was originally denied … go figure), but our current stance is that developers are urged to fix these warnings up on an opportunistic basis, i.e. if you’re working on a package, try to get it to compile cleanly. So far, these warnings only occur on host computers, and only when the code is compiled optimized (which is not the default optimization level when compiling in your own test area! Beware!)

Other than that, assuming both builds run to completion, it’s time to hand them off to testing.

3 Testing The Build

There are usually two steps in testing a build.

3.0 Sanity Check

To avoid committing to the new build, it's useful to simply sanity check it first. In this mode, any new flyable constituents are loaded across the network into the test stands. There are even two sub-variants of this style of testing:

- Load *all* flyable constituents across the network (easiest).
- Load only *new* flyable constituents across the network (more realistic).

Testing consists of using the LTX tool to run our suite of test programs. This was originally the Final Qualification Test (FQT) test suite, though it has evolved since (and continues to evolve).

3.1 Take a Deep Breath

Up to this point, any testing errors can be corrected by fixing up the code and absorbing the changes into the build, without changing the build number. It amounts to simply redefining what the build content is. The next step brings that era to a close. Once the next step is completed, the build content is irrevocably set. Find a mistake after this point, and the only remedy is to define a new build.

OK, that should have frightened you enough.

The next step is to import the build into FMX. This is very similar to the command used previously to characterize the new flight constituents, but this time, it's run with the `--nodry-run` option. With this option, the command does a significant amount of work. Expect it to run slowly!

```
flora01:apw> fmx import Bx-y-z --nodry-run
Import CPU_DB    V0-2-4    cpu_db_server          relocateable  : OK
Import CPU_DB    V0-2-4    cpu_epu                cdm           : OK
Import CPU_DB    V0-2-4    cpu_siu                cdm           : OK
Import EDS       V2-3-3    eds                    relocateable  : OK
Import ITC       V3-5-1    itc                    relocateable  : OK
Import LATC      V7-6-5    latc                   relocateable  : OK
Import LATC      V7-6-5    latc_cmn               relocateable  : OK
Import LDT       V0-1-1    encdec                 relocateable  : OK
Import LSE       V1-0-4    lsew                   relocateable  : OK
Import PBC       V4-12-0   epu_e_promimage.objz   pbc           : OK
Import PBC       V4-12-0   epu_f_promimage.objz   pbc           : OK
Import PBC       V4-12-0   pbc                    relocateable  : OK
Import PBC       V4-12-0   siu_e_promimage.objz   pbc           : OK
Import PBC       V4-12-0   siu_f_promimage.objz   pbc           : OK
Import THS       V1-5-0    ths                    relocateable  : OK
Import FTS       V0-1-7    siu.fmx.template       sbs           : OK
Import FTS       V0-1-7    epu.fmx.template       sbs           : OK
```

Figure 11 Importing a build into FMX

3.2 Upload To Testbed / Test Stands

This is most easily (and most accurately) accomplished using LICOS tools. It might take a while to fire up a complete LICOS environment (and the rules seem to be changing a little as I&T methods and ISOC methods are converging), but the level of automation and the strict interface to FMX are well worth the effort. Once LICOS is fired up, look for the application:

`$ONLINE_ROOT/LICOS_Scripts/seApps/FileSetUpload.py`

As usual with LICOS applications, it expects to be driven by a configuration file. These configuration files need to be updated for each new build. It is also common practice separate uploading the EPUs from uploading the SIU (with the EPU uploads being performed first). Each CPU type has a distinctive script.

I keep private copies of these configuration files so the following examples should not be regarded as definitive. The real expert here (and the writer of the `FileSetUpload.py` script) is Jim Pannetta.

The only other advice I can provide is that the target instrument should be set up in quiescent mode, with any EPUs powered on and in applications mode before firing up the script.

```
; B1-0-8 upload to EPUs on instrument arnor

[config]
; Version and release info
version=$Revision: 1.2 $
release=$Name: L00-59-00 $

; Unit to test
instrument = arnor

[steps]
; Step sequence = Step name, [enabled:1, disabled=0]
1=uploadSecondary,1
2=uploadPrimary,1

[uploadSecondary]
prerequisite=secondaryBoot, lfsStatus
verification=verifyUploadShort, lfsStatus
; Other step specific options
bootFrom     = ee0
debug        = False
stageTo      = /siu1/ram
targetSet    = /epu1/ee0, /epu1/ee1, /epu0/ee0, /epu0/ee1
targetSet    = /epu0/ee1
uplObjects   = sbs/FTS/V0-3-21/B1-0-8/epu.fmx.template


[uploadPrimary]
prerequisite=primaryBoot
verification=secondaryBoot, verifyUploadShort, lfsStatus
; Other step specific options
bootFrom     = ee0
debug        = False
stageTo      =
targetSet    = /epu0/mm0, /epu0/mm1
uplObjects   = sbs/FTS/V0-3-21/B1-0-8/epu.fmx.template
```

Figure 12 LICOS configuration file for uploading a build to EPUs

```
; B1-0-8 upload to an EPU on instrument arnor

[config]
; Version and release info
version=$Revision: 1.2 $
release=$Name: L00-59-00 $

; Unit to test
instrument = arnor

[steps]
; Step sequence = Step name, [enabled:1, disabled=0]
1=uploadSecondary,1
2=uploadPrimary,1

[uploadSecondary]
prerequisite=secondaryBoot, lfsStatus
verification=verifyUploadShort, lfsStatus
; Other step specific options
bootFrom     = ee0
debug        = False
stageTo      = /siu1/ram
targetSet    = /siu1/ee0, /siu1/ee1
uplObjects   = sbs/FTS/V0-3-21/B1-0-8/siu.fmx.template


[uploadPrimary]
prerequisite=primaryBoot
verification=secondaryBoot, verifyUploadShort, lfsStatus
; Other step specific options
bootFrom     = ee0
debug        = False
stageTo      =
targetSet    = /siu1/mm0, /siu1/mm1
uplObjects   = sbs/FTS/V0-3-21/B1-0-8/siu.fmx.template
```

Figure 13 LICOS configuration file for uploading a build to an SIU

3.3 Test Again From Internal Resources

This is the final testing step. The complete test suite is run again, but this time the testbed and test stands are set up to do a full flight autoboot, and to read all code from the on-board file system. Make it through this level of testing and the time has come to declare a fully qualified build.

4 Annotating The Build

Information from Kim Lo. Essentially quoting LAT-TD-07275-01 “LAT Flight Software Build Procedure” (who knew!).

Joking aside, this section really needs fleshing out, both in terms of overall concepts and a more detailed description of what the steps are for and how they’re accomplished. Right now this looks like a lot of hand work. It could probably be further automated.

So you have a new build and it’s passed regression testing. Now’s the time to let the world know about it.

The steps go something like this:

- Create a home for all the documents, this is traditionally at:

```
noric01:apw> cd $GFSROOT/cmx/FSW_releases
noric01:apw> mkdir Bx-y-z
```

- Modify some scripts to make them build appropriate:
  - Edit `$FSW/web-dev/bin/cat_get` as follows:
    - I have no idea
  - Edit `$FSW/web-dev/bin/use_tree_get` as follows:
    - I have no idea
  - Edit `$FSW/web-dev/lib/fsw_lcat.pl` as follows:
    - I have no idea
- Run a script to generate web pages and PDF documents

```
noric01:apw> source $FSW/web-dev/release_gen_cat.sh
```

  - What web pages/documents (and their directories) does this produce?
- Copy all (???) the PDF documents to `$GFSROOT/cmx/FSW_releases/Bx-y-z.`
- Add a new release link to the FSW home page.
- Run another script to collect useful documents (???, doxygen pages, PDF documents) for this release and link them with the new release link made in the previous step.

```
noric01:apw> cmx_release
```

    - With what arguments? How does it know that this is `Bx-y-z`?

- Post FSW release notice on Snitz.

5 Exporting The Build

The final upload of the new build to the spacecraft falls under the control of ISOC and MOC, but they will also need a few associated products. Examples of associated products are:

- A new command and telemetry database (this goes to ISOC first for post-processing, then MOC).
- Assorted python "database" files which are in fact just an alternative description of the command and telemetry database. (LICOS/ISOC have been converting to a common database strategy so these products may not be so relevant as they once were).
- A revised MSG database.

These days, these products are automatically generated by the build, and the results put in the `$CMX_C_CDB` directory. You should find:

Kim, please clean up this file list. I don't know which variants are the ones actually exported.

```
-rw-r--r--   1 apw      ey        227972 Oct 29 22:03 LCAT_cmd_db.py
-rw-r--r--   1 apw      ey        143492 Oct 29 22:03 LCAT_msg_db.py
-rw-r--r--   1 apw      ey       2877792 Oct 29 22:03 LCAT_tlm_db.py
-rw-r--r--   1 apw      ey         69125 Dec 15  2004 itos_msgcode.txt
-r--r--r--   1 apw      ey        178965 Oct 29 22:02 lat_cmd_itos.dbx
-r--r--r--   1 apw      ey       1886914 Oct 29 22:02 lat_tlm_itos.dbx
-rw-r--r--   1 apw      ey        185843 Oct 15  2006 lcat_cmd_db.py
-rw-r--r--   1 apw      ey       1310355 Oct 15  2006 lcat_tlm_db.py
```

Figure 14 Products associated with a build

5.0 Exporting The "Other Stuff"

This section is a placeholder. I know that various organizations grab other FSW products ("Q" packages, …). Other organizations are sometimes involved in actually creating/maintaining these produces ("DFI"). What I don't know is how that process takes place, nor if there are any formal methods to control it. Should such processes be discovered, this might be a good place to document them.