HIRDLS

SW-OXF-287

High Resolution Dynamics Limb Sounder

Originator: John Barnett

Date: 4 June 2001

Subject/Title: Standards and Conventions for Postlaunch SAIL Tasks

Contents:

This document outlines conventions which SAIL tasks used after launch will need to adhere to in order to facilitate memory verification, configuration control, and the ability of ground controllers to detect problems.

Key Words: SAIL

Atmospheric, Oceanic and Planetary Physics,
Oxford University, Department of Physics,
Clarendon Laboratory,
Oxford OX1 3PU
United Kingdom

This document outlines conventions which SAIL tasks used after launch will need to adhere to.

1. SAIL Task Identified numbers.

This is composed of a 6 character hex string (valid characters 0-9 and A-F, not case sensitive). Character 1 (counting from left) gives the type of task, according to the following list:

1	Instrument integration and test
2	Instrument prelaunch calibration
3	Spacecraft integration and test
4	Postlaunch activation tasks
5	Baseline postlaunch scanner task
6	Sunshield postlaunch scanner task
7	Schedule postlaunch scanner task
8	Heater postlaunch scanner task
9	Housekeeping postlaunch scanner task
A	Telemetry postlaunch task
B	Postlaunch calibration tests (pitch down, phasing etc)
C	Postlaunch hardware test task
D	available to be defined
E	available to be defined
F	Other

Characters 2-4 give the task number; this allows 4096 different tasks within each category.

Characters 5-6 give the version number; this allows for 256 versions. If this needs to be exceeded, a new number should be started.

A task should be included only once; e.g. where the same task is used in various prelaunch categories then after launch for hardware testing, it might be given the category of first use.

2. File names and extensions.

These will consist of a name followed by a period then an extension; characters must not be assumed to be case sensitive (to avoid problems transferring to and from DOS systems). The ID string has already been defined to use only hex characters; only 0-9, A-Z, _ and - may be used elsewhere since other characters sometimes cause special actions with various operating systems.

The name is composed of the 6 character task ID followed by up to 26 other characters. This is to enable users to use their own naming conventions but to retain the identifier. It has the advantage that under DOS (which is currently used by the SAIL compiler and assembler), the identifier will normally be part of the truncated name (e.g. A12345~1.sai). Different names should not be used with the same ID, since for many purposes the ID alone will be bused to identify the task.

The extensions may be up to 3 characters and the following have so far been defined:

sai	SAIL source code
a, p1, p2, log	intermediate files generated during the SAIL compilation and assembly
c	SAIL code relocatable image for loading into the IPU
d	SAIL data relocatable image for loading into the IPU

3. Use of ID in SAIL tasks

The first global variable shall be an int (32 bit integer) into which the task will write the 6 hex character identifier; we will call it IDcblock for the purposes of this document.

The second global variable shall be an int (32 bit integer) initially containing the task 6 hex character identifier; we will call it IDdblock.

The first statement in main shall be of the form
IDcblock=taskID; where taskID has been defined in a #define statement, or
IDcblock=0xC12345; where C12345 is the example ID;
(note that variable names are not case-sensitive).

The statement must be exactly of this form to allow extraction of the ID value from the .c code image file. e.g. IDcblock=0xC00000 ! 0x012345; is not allowed.

The result of these requirements is that the ID can be simply extracted from the code and data image files as a check on the string in the file name, the same can be done with memory dumps, and inspection of the first 4 bytes of the memory dump of data block will reveal whether the task has ever run and if so with what task ID.

e.g. a minimum task is

```
#define taskID        C12345
int IDcblock;
int IDdblock = taskID;
main() {
 IDcblock=taskID;
}
```

Any program extracting the identifier from the code file, will need to be able to skip over and instructions which initialise variables in main. This it will do by decoding the instruction type, then the address modes, then calculate the instruction length. It will only recognise a subset of possible instruction types. It will do this until it finds an instruction of the required form to be IDcblock=taskID (it will be a MOV !????????, 1E where ??????? is the ID). Hence it would be possible to place certain statements before the IDcblock=taskID. The only useful statement is probably of the form oldIDcblock=IDcblock, where oldIDcblock is used to record the previous contents of IDcblock so that the task may determine whether the data area has been used before and by what task. Such a program has been written and functions successfully.

4. Data block memory verification marker.

After launch it will sometimes be necessary to verify the integrity of task images, including when they have been resident in memory for a long period, since procedural, hardware or software errors could conceivably have caused a change. This includes the ability to verify tasks that have been run, so will have different values for variables from those initially present in the loaded memory image files.

There is no problem in verify code blocks since they should be unchanged by being run.

The only areas of data blocks which must be unchanged are those which the user does not want to be changed (e.g. because they are calibration constants used within the task); to achieve that they will declare the variables global and given them initial values. [Note that where variables are given initial values with function, then those values are written in each call from values stored within the code space.] In order to allow verification, the following rule must be complied with:

Variables requiring verification must be put at the start of the global variables, followed by an int initialised to 89ABCDEF.

Additional groups of variable to be verified can be specified by preceding them by an int initialised to 89ABCDEE and following by an int initialised to 89ABCDEF.

In the special case of there being no int containing 89ABCDEF, then no verification will take place, other than of the task identifier in the second global int.

The variables containing 89ABCDEE and 89ABCDEF may be used for other purposes by the SAIL task (i.e. used as a variable) since they will only be searched for by ground software in the data block image file.

The choice as to whether a variable is to be verified or not is important. If it is to be verified, then it must not be changed by SAIL, since it would then fail verification. Hence global variables that are used as general working variables must not be verified. Conversely, items which the task requires to have certain values must either be verified or be set up by an executable statement each run of the task.

5. Error and status flags

Most SAIL tasks will check for error conditions and need to report them by setting flag bits and possibly displaying values in parameter slots. Sometimes they will then suspend themselves, and on other occasions they will continue operating. Frequently these error conditions will be undetectable in the very limited Engineering Data Stream telemetry available to the ground controller during the few minutes of each real-time contact at intervals of one to two orbit periods. In other cases, elaborate code would be needed on the ground to detect the same condition, particularly since not even the Science Data Stream will contain all of the telemetry available to SAIL tasks.

Tasks will set flags in two ways:

1) In parameter slot E, tasks will use the lower order 16 bits to display error flags for the duration of the error; in some cases this will be just for 1 maf.

2) In parameter slot F, tasks will use the lower order 16 bits to display errors which have occurred at some time in the last approximately 6 hours. The bit assignment must be the same as for slot E.

The Engineering Data Stream will only contain parameter F for each task, whereas the Science Data Stream will contain parameters E and F each major frame.

It is proposed that bits 0-7 be reserved for serious errors where the ground controller would attempt to contact one of the HIRDLS team by telephone, whereas 8-F would be reserved for less serious errors where an email message would suffice.

Possible simple SAIL code to achieve this is as follows:

```
     int cumflags[4];    // cumulative flags, in global in order to preserve them
                         // - need to be zeroed at start of task

// Each maf display flags accumulated this maf:
      _shared[baseaddress+14] = flagbits;
      flagbits=0;                                  // clear them for next maf

// display accumulated flag bits for at least 5.24 hours and at most 6.99 hours
// (32768 major frames is 6.99 hours):
      subframe = _maf >> 13 & 3;                   // subframe is an int variable
      if (_maf & 0x1FFF == 0)  { cumflags[subframe] = 0; }  // zero periodically
      cumflags[subframe] = cumflags[subframe] | flagbits;   // OR in any errors

      _shared[baseaddress+15] =
         cumflags[0] | cumflags[1] | cumflags[2] | cumflags[3];
```

does an ops log get recorded for each up load?
Can we get the ENG stream in packet form?
- HSTER