Technical Documentation

James Tappin

11th February 2002

1 Introduction

This document is an attempt to give an outline of the internal workings of IDL_HS, anyone wishing to extend the package should read this in conjunction with the "Reference Manual" and the "User's Guide".

For the general user it is probably of interest mainly to understand why things are done the way that they are.

2 Overall structure

2.1 History and rationale

Prior to version 4, IDL_HS consisted of a largely *ad hoc* collection of display programs which shared a number of settings and communicated via a vast and largely unseen collection of common blocks (like the cables in the false floor of old-fashioned computer machine rooms). On top of this was layed a widget interface, which was added when the number of routines became too large for anyone to remember any but the commonest controls.

The major problem that this caused was that design decisions taken early in the Ulysses mission, when there was a far smaller dataset and the intended scope of the package was rather less, imposed considerable constraints on the way in which the package could be developed. In addition the lack of a consistent internal structure made it very difficult to extend the code without undesirable side-effects.

As a result of this,combined with a desire to be able to see EPAM & ULEIS data on the same plot, a decision was taken in late 1999 to rebuild the entire system from scratch.

2.2 Basic structure

At the outset 4 key decisions were made:

1. The new version would make use of the new IDL object system for its internal operation as this would greatly simplify communication between different bits of the code, and greatly reduce the need for the common blocks.
2. Despite this, the traditional IDL direct graphics was to be retained as they are better adapted to data plotting than the new object graphics system. (This is a matter of how RSI chose to implement such things as text scaling in object graphics).
3. Time values would be stored as JD numbers.
4. The GUI would be an integral part of the system, but there would also be a command-line interface that did not require the user to manipulate IDL objects directly. The CLI was to be more directed towards scripting than interactive use as the majority of users have access to X displays.

It quickly became clear that the easiest way to implement the desired function was to create a system of datasets and streams; in which a dataset contained the information about timing, scaling etc. while the actual data for each data channel was contained within streams attached to the dataset.

The overall structure of the resulting system consists of a cross-linked net of objects, summarized in figure 1.

In essence, there is a single top-level container object (sometimes referred to as the environment) to which everything else is attached. Directly attached to the environment are the datasets and global data.

Streams are then attached to the datasets.

3 Global objects

Global objects are those which apply to the whole system. These can be described under 3 heads:

3.1 HS_CONTAINER

The HS_CONTAINER object is a master container which acts as a container for all the other objects used by the system. It contains references to other objects and to pointers, most of its methods are related to accessing the other objects of the system.

In addition, there are a number of housekeeping variables that are maintained by the master container.

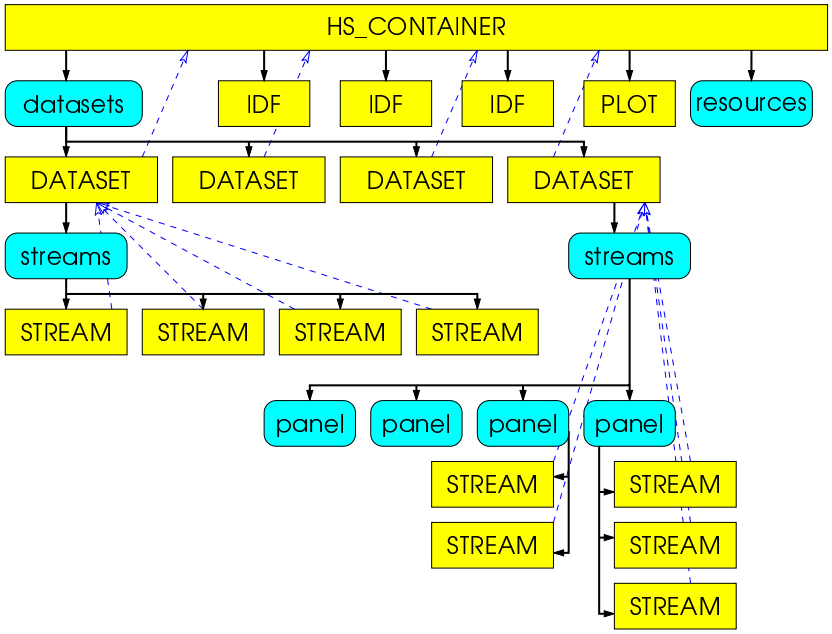

Figure 1: Schematic diagram of the structure of the IDL_HS display system. The yellow boxes represent objects; the cyan boxes with rounded corners, major pointers. The black arrows indicate the primary references linkages, and the blue dashed ones indicate links back to higher levels.

3.2 IDF

IDF objects are instrument definition specifications. These contain information such as energy ranges, sectoring etc. which are specific to the particular data type.

Any instrument or data class that is added to the system must have an IDF object defined for it. By their very nature there are considerable internal differences between the different IDFs. However, the methods of access are reasonably consistent. Even so care is needed as the differences between instruments do at times need to be resolved at a higher level (for example, ULEIS requires time information to get geometry factors).

Each IDF object appears as a member of the hs_container object. Currently defined are:

- HISCALE Rates (**lan_idf**)
- HISCALE Track/PHA (**trk_idf**)
- EPAM Rates (**lan_idf**)
- EPAM Track/PHA (**trk_idf**)
- HISCALE MFSA (**mfsa_idf**)
- EPAM MFSA (**mfsa_idf**)
- ULEIS (**uleis_idf**)
- UDS (**uds_idf**)
- FLARE (**xflare_idf**)

The HISCALE and EPAM IDFs use the same object class.

All IDF objects need to inherit the IDF search path class, so that the locations of appropriate data can be specified.

ULEIS IDF's have attached **uleis_calib** objects corresponding to the different calibrations for different times (thus anything that needs to get ULEIS calibration information needs to pass a time).

The UDS IDF has a separate **uds_instrument** object for each data type as defined in the spec files.

All IDF classes inherit the **idf_search** class to store where the data are kept.

3.3 PLOT

The plot object (**hs_plot**) contains all the information about the plotting device and its settings. This includes *inter alia* the current device, page and window sizes and the spooling commands for hard copy output. In addition, it keeps a record of the current plot transforms.

4 Datasets

A dataset is the object that is acquired and plotted. There are currently 6 classes of datset defined:

rate_dataset A time-series plot. Allows the display of flux or count rates as a function of time.

pad_dataset, displays fluxes or count rates as a function of pitch angle.

spect_dataset, displays spectra of particle flux against energy.

matrix_dataset, displays the ΔE vs E matrix from the HISCALE or EPAM WART detector.

xflare_dataset, generate a display of the X-ray flare lists from NGDC.

dyn_spect_dataset, displays a dynamic spectrum, for a group of channels.

In addition there is a pseudo-dataset class (**file_dataset**) defined to look at the intternals of an HSIO format file.

4.1 Helper classes

Since many of the classes of dataset share properties, there are a number of helper classes that exist simply to save code.

generic_dataset this is inherited by ALL other datasets and contains the connections to the top-level container and to the dataset's streams. This also contains the time range and the name of the dataset.

dshelper_average inherited by any dataset for which time-averaging is meaningful.

dshelper_background inherited by any dataset for which background subtraction is meaningful

dshelper_error inherited by any dataset needing error bars.

dshelper_flux inherited by any dataset which may need to convert between count rates and fluxes.

dshelper_hierarchical inherited by those datasets in which channels are grouped together into panels.

dshelper_lan_archive inherited by any dataset class that needs to access LAN rates data.

dshelper_long_colour inherited by any dataset that needs to display colour-coded data.

dshelper_raw_spect inherited by the two spectral dataset classes for their raw data.

dshelper_sector inherited by datasets that need to be able to select sectoring options.

5 Streams

A stream object contains the data and other necessary information for a single channel (or equivalent). A separate stream class is needed for each instrument and dataset class combination supported.

Stream classes are frequently heavily subclassed, since (for example) while all rate streams share many features there are some that are specific to (say) LAN rates, and of those some are HISCALE or EPAM specific.

In PAD and SPECT datasets, the streams are organized into panels. There is however no panel object, just that in those datasets, the streams pointer is a pointer to pointers rather than directly to the streams.

For matrix and flare datasets, only a single stream can be attached.

Many properties of the dataset (e.g. sectoring) can be overridden for specific streams.

5.1 The basic stream inheritance scheme

This list summarizes the inheritance scheme of streams. Only those classes that are underlined will normally be instantiated.

- **rate_stream**
  - **lan_rate_stream**
    - **epam_rate_stream**
    - **hiscale_rate_stream**

  - mfsa_rate_stream
    - epam_mfsa_rate_stream
    - hiscale_mfsa_rate_stream
  - track_rate_stream
    - epam_track_rate_stream
    - hiscale_track_rate_stream
  - mag_rate_stream
    - epam_mag_rate_stream
    - hiscale_mag_rate_stream
  - uleis_rate_stream
  - uds_rate_stream
  - ratio_rate_stream
- spect_stream
  - lan_spect_stream
    - epam_spect_stream
    - hiscale_spect_stream
  - mfsa_spect_stream
    - epam_mfsa_spect_stream
    - hiscale_mfsa_spect_stream
  - comp_spect_stream
    - epam_comp_spect_stream
    - hiscale_comp_spect_stream
  - uleis_spect_stream
- pad_stream
  - pad_pad_stream
    - lan_pad_stream
      - epam_pad_stream
      - hiscale_pad_stream
    - uleis_pad_stream
  - pad_look_stream
    - lan_look_stream
      - epam_look_stream
      - hiscale_look_stream

- **dyn_spect_stream**
  - **lan_dyn_spect_stream**
    - **epam_dyn_spect_stream**
    - **hiscale_dyn_spect_stream**
  - **mfsa_dyn_spect_stream**
    - **epam_mfsa_dyn_spect_stream**
    - **hiscale_mfsa_dyn_spect_stream**
  - **comp_dyn_spect_stream**
    - **epam_comp_dyn_spect_stream**
    - **hiscale_comp_dyn_spect_stream**
  - **uleis_dyn_spect_stream**
- **matrix_stream**
  - **epam_matrix_stream**
  - **hiscale_matrix_stream**
- **matrix_histogram**
- **xflare_stream**

5.2 Helper Classes

Like datasets, there are stream helper classes that allow code-sharing between streams that need to have similar properties. There is also a **generic_stream** class inherited by all streams that has the necessary attachment points etc.

6 Non-object code

The routines for i/o (HSIO and ULIO interfaces and the routines to read a record from those files) and utilities (e.g. date conversion) are not written as object methods, but are normal IDL procedures and functions.

7 Common Blocks

There are now only two common blocks used in IDL_HS: one for the top-level container object so that it need not be exposed at the main-program level and the other for the HSIO and ULIO entry points.