ASTROPHYSICS and SPACE RESEARCH GROUP
DEPARTMENT OF PHYSICS AND SPACE RESEARCH
UNIVERSITY OF BIRMINGHAM
PRG/001
Asterix System Note
ASTERIX Programmer Note 001
David J Allan
27 May 1996

ASTERIX Programming

Contents

Chapter 1

Introduction

Chapter 2

BDI – The Binned Data Interface

- BDI knows what shape every component in an NDF should be given the dimensions of the data array. Checks dimensions of all file components on read/update. No create component routines needed.
- Supports access to structures of type ARRAY as if they are simple arrays of numbers. Lets you do

  ```
  add file.axis(1).data_array 2.0 over
  ```

  without worrying about representation of the array
- Fewer basic routines. Switching by object is performed using the item argument. Means applications can loop over file components.
- Consistent argument ordering

  fileid, [¡iax¿], ¡item¿, ¡mode¿, [¡type¿], status
- Access multiple bits of data simultaneously, means fewer calls required.
- Asymmetric widths supported on all axes
- Can refer to axes using physical quantity codes, eg. X,Y,E,T,P for X position, Y position, energy, time and phase, eg.

  ```
  bdi_mapr( id, 'E_Axis_Data', 'READ', eptr, status )
  ```
- Invented data supported for read, update and write where appropriate.
- No create routines needed. Objects are created on demand so its harder to create objects which are subsequently unused
- No common block, no limit on number of active datasets

Chapter 3

EDI – The Event Data Interface

Chapter 4

PSF – Point Spread Function Access

4.1 Introduction

The psf system provides an instrument independent interface to psf information for ASTERIX applications. A client application associates a psf with a dataset (`psf_associ`), which can be a binned dataset or an event dataset. The psf system identifies the spatial axes within the dataset (using BDI and EDI), and the energy axis if present. The psf system returns an ADI identifier to the application which contains the psf state variables for that dataset. Subsequent calls to the PSF system need only supply this identifier.

The psf system is in a fairly good state – the only unfinished work was to split up the `psfrtn.f` module to document the routines properly and make debugging easier.

4.2 How Does it Work?

The psf system chooses a routine to initialise a particular psf by looking at the name supplied at the PSF prompt by the user. If this name is a valid abbreviation of any known psf then the initialisation routine is called. The mapping between the full name and the routine is defined using the internal routine `PSF0_DEFPSF` in the `PSF_INIT` module. The initialisation routrine has the standard signature,

```
SUBROUTINE psf_init_rtn( INTEGER psid, INTEGER status )
```

where *psid* is the psf identifier. The dataset with which the psf is associated can be extracted from the `FileID` member of the structure referenced by *psid* (see the file `$AST_ETC/psf.adi` for the structure definition).

The psf initialisation routine's job is to ask for or locate any additional information required to specify the psf when actual data is requested. This extra information is known as psf *instance data* as it is peculiar to the particular psf/dataset combination for which it is computed. This data can be stored in *psid* using the `psf0_setid0`*t* routines for subsequent retrieval by the routines evaluating the psf data.

The most important job in initialisation is to associate other routines to perform other functions in the psf system – only one of these is mandatory, the data routine.

4.3 The Data Routine

The data routines are the most important part of the psf system. Given a psf identifier, a detector position in radians, bin sizes and dimensions a data routine returns a 2-D probability array. An offset may also be applied to shift the psf centre from the centre of the 2-D array. It is important to realise that when writing the data routine it could be called with any bin size and not just some default detector pixel size, or multiply thereof. After writing a new data routine evaluate the psf using the CREPSF application using different bin sizes and image extents to check whether the normalisation is being computed correctly. Extra information beyond that supplied as input arguments should be passed using the psf instance data described in the previous section.

The data routine signature is,

```
SUBROUTINE psf_data_rtn( INTEGER psid, REAL x0, REAL y0, REAL qx, REAL qy,
                         REAL dx, REAL dy, LOGICAL integ,
                         INTEGER nx, INTEGER ny, REAL array[],
                         INTEGER status )
```

where $x0, y0$ is the detector position in radians, qx, qy is the offset from the centre of the output probability *array* to the psf centre, dx, dy are the bin sizes in radians and nx, ny the dimensions of the output array. The *integ* flag is historical and is not currently used.

4.4 The Psf Hint Routine

While applications should be written in an instrument independent fashion, it is still useful to obtain information about the properties of a particular psf, often resulting in substantial efficiency improvements. If a psf initialiser has specified a *hint routine* then the psf system will call it in attempt to get the value of a named hint. Standard hint names are defined in PSF_PAR. Not all hints need be supported – the psf system returns both a "hint available" flag and the hint data to the client.

The hint routine signature is,

```
SUBROUTINE psf_hint_rtn( INTEGER psid, CHAR hintname, BYTE data[],
                         INTEGER status )
```

Each hint has a natural data type. Most are logical but a few are single precision real numbers. Where the real number refers to a spatial scale the units are radians. A type hint might be whether or not the psf exhibits a particular symmetry on the detector, such as azimuthal symmetry.

4.5 Energy Profiling

The psf system provides a facility to evaluate the enclosed energy function for a pariticular psf at any point on the detector. The default way to do this is to evaluate the psf using the data routine in thin strips starting at the psf position on the detector. This technique is fine as long as the normalisation of the psf is rigorously computed in the data routine. If this is not the case the profile routine can return garbage or even loop indefinitely. If this happens with a new psf then either look at your normalisation in the data routine carefully, or explicitly supply a profile routine of your own. The latter option is often preferrable, as most instrument calibrations produce a parameterised energy profile early on in the calibration programme.

4.6 The Closure Routine

If the initialisation routine defines a closure routine then it will be invoked when the application releases the psf using the `psf_release` call. This allows any workspace allocated or files opened to be freed or closed before the application exits. The closure routine signature is the same as that of the initialisation routine.

4.7 Psf Models

Psf models are high level manipulations of the low level instrument psfs supported by the psf system. There are spatial models and energy models, and the two can be used together. The purpose of spatial modelling is to impose granularity of psf access to speed things up. An application like PSS which having determined using the hints system that a psf varies across the field, is most easily written to simply evaluate the psf at every position of interest. For some psfs this would be extremely slow, so spatial modelling by the psf system allows grids of psfs to be set up, the number of evaluations being reduced to the number of psfs in the grid.

Energy modelling serves a slightly different purpose. When processing image data from an instrument with an energy dependent psf it is not strictly correct to use a mean photon energy, even if the source spectrum is monochromatic. The psf system CSPEC psf option allows a channel spectrum of a source to be supplied which the high level data routine `psf_2d_data` uses to construct a composite psf by performing a weighted average of psfs at different energies. This is the correct way to get the best source flux estimate for a source in 2-D image.

4.8 The Energy Dimension

If the user supplies a spectral image to the psf system (this is only done in the cluster fitting at present) then when the data routine is called energy information must propogate from the client down to that point. This is achieved using either `psf_def` or `psf_defb`. In the first case the user supplies energy and time bands (the latter are not currently used anywhere) for psf evaluation. The alternative `psf_defb` supplies bin numbers in the energy and time dimensions for psf evaluation. The psf data routine for an energy dependent psf should check for the energy band psf instance values before computing the psf. See `psf_xrt_pspc` in `psfrtn.f` for an example. If the psf is energy dependent bu the user supplies only an image, then the initialisation routine should prompt for a mean photon energy.

4.9 Utility Routines

Two additional routines are supplied which don't require a psf identifier but which might be of more general use – they are `psf_convolve` and `psf_resample`. The first convolves a psf with a 2-D kernel. The second performs sub-pixel resamling of 2-D psfs by X and Y offsets. This latter routine is particularly useful in applications which must allow sources to appear at positions other than pixel centres.