DEPARTMENT OF SPACE RESEARCH
BIRMINGHAM UNIVERSITY

PROG/012

Asterix Data Analysis
ASTERIX Programming Note 012

TJ Ponman
15 January 1990

Introduction to ASTERIX Programming

Contents

1 INTRODUCTION

This document is a route map for those wishing to contribute software to ASTERIX. It aims to give a concise guide to what you need to do, and where you can find further details.

2 UNDERSTANDING THE SYSTEM AND ITS PHILOSPHY

Before contributing to the system it is important that you are familiar with the system as a user, and get the 'feel' of it. This is best done by reading the User Guide (USER_001), performing some analysis, and studying the two major programmer documents: PROG_002, which describes the standard datasets and conventions on which the system is based, and PROG_001, which specifies the rules for ASTERIX programming. You will also find the yellow ADAM user guide, SG4, useful if you are unfamiliar with ADAM/ICL.

The basic shape of the system is a set of instrument interfaces, and a body of instrument-independent software, here we concentrate on the latter. A guide to writing an instrument interface will be provided later as a separate document. An ASTERIX application should be:

- robust - it should make the minumum of assumptions about the nature and size of the input data. It should work whenever it is given enough information to proceed sensibly, making defaults as necessary, and warning the user wherever this seems appropriate.
- easy-to-use - it should provide a modest number of informative prompts (additional parameters may be invisibly defaulted in the interface file, and hence available to the expert user), with on-line help, and a detailed description of method and usage in the ASTHELP system.
- atomic - each application should perform one or a small number of jobs thoroughly, on a wide variety of different inputs. Separate functions should be implemented in separate programs, unless there is some very good reason to the contrary. This makes the system more flexible and comprehensible.

3 THE RULES

PROG_001 should be carefully digested, and the rules in the 'Immediate' section should ALL be observed. The second section is more advisory. Some points to watch are:

- Always use the standard ASTERIX program headers.
- Use standard ASTERIX subroutines for data access and user interfaces wherever possible. In particular the BDA_* subroutines should be used when reading, writing or copying standard components of binned datasets.
- Primitive data input (i.e. a simple array rather than a whole dataset) should be handled wherever this might be useful.

- QUALITY and VARIANCE are supported by ASTERIX applications wherever they are relevant (though in some cases the support might consist only of warning the user that bad data are present, for example).
- Bulk data should be mapped, and assumptions about its size avoided (dynamical storage is available using the DYN_* package).
- Overwriting of a dataset should in general occur only via an OVER parameter entered on the command line.
- User i/o should be grouped together as far as possible within an application; usually either in the top level, or in a small number of subroutines.
- Applications should exit if the user responds !! (abort) in response to a prompt (i.e. check STATUS after each user input and EXIT if it is bad).
- Use the PRS_* routines to obtain any range specifications from the user.
- Make frequent STATUS checks to avoid program crashes.
- Write a HISTORY entry wherever a dataset is created or modified.
- All Starlink standard components should be propagated to the output file, except where they are known to be inappropriate or wrong.
- Check STATUS at the end of all applications and subroutines and provide error traceback.

4 FINDING EXAMPLES

ASTERIX source code is kept in text libraries. Application code is kept in a collection of monolith libraries SPECLIB:, TIMLIB: etc. (type SHO LOG *LIB to list all the defined libraries), along with application-specific subroutines, whilst general ASTERIX subroutines are kept in a single ASTLIB library. Source code can be examined or extracted from these text libraries using the DOCLIB tool e.g. type DOCLIB ASTLIB: and then follow the prompts; * acts as a wildcard.

Interface files for applications are stored in parallel text libraries stored in the monolith directories (the ASTERIX directory structure is explained in PROG_009), e.g. SPECLIB_IFL.TLB. The interface file for a particular application can be extracted with LIB/TEXT/EXTRACT.

Some exemplary programs and interface files can be found in directory AST_EXAMPLE.

5 TYPICAL STRUCTURE OF AN ASTERIX PROGRAM

The general structure of an ASTERIX program might look like this:

- Announce application name and version
- AST_INIT - Initialises ASTERIX subroutine packages

- USI_ASSOC* - Get locator(s) to input (and maybe also output) file(s)
- Check data (e.g. dimensionality) and map if OK (using BDA_* routines)

```
If data not primitive
        Check for regular axes and map          } Using
        Map variance                            } BDA_*
        Map quality (as array of logicals)      } routines
Endif
```

- Make any additional checks
- Further user input (N.B. Use PRS_GETRANGES for any numerical range specifications)
- USI_ASSOCO - create output file (if not already done above)
- Create components in output file (e.g. BDA_CREBDS) or copy from input file (BDA_COP*)
- Map output data, and maybe variance and quality (using BDA_*)
- Pass pointers to input and output data to main subroutine to perform the computations
- Copy ancillary information from input to output files (e.g. BDA_COPMORE)
- Write any special components, labels etc. to output file
- Make HISTORY entry in output file (using HIST_*)
- AST_CLOSE - shut down ASTERIX subroutine packages

6 EXISTING SUBROUTINES AND THEIR DOCUMENTATION

The most frequently used ASTERIX packages of subroutines are described in the documents PROG_003, 004, 005 and 006, which should be consulted before writing an ASTERIX application. Subroutines in ASTLIB are organised by name using a system of prefixes. A list of the currently allowed prefixes and their meanings can be viewed by typing PREFIX (after invoking ASTDEV). All these routines reside in ASTLIB, and are more fully documented in their program headers, which may be inspected with DOCLIB.

Several Starlink subroutine packages are also important: the main HDS access routines are (mostly) documented in SUN92, the PRIMDAT routines (useful for simple mathematical manipulation of mapped arrays) are described in SUN39 and the CHR_* character string manipulation package in SUN40. Brief descriptions of ADAM subroutines (such as PAR_* and DAT_ASSOC) may be found in SG4, and detailed documentation within directory ADAM_DOCS. Several other packages such as SLALIB (SUN67), may be useful for specific applications. These can mostly be spotted by inspecting the Starlink documentation list in DOCSDIR:DOCS.LIS.

7 ADDING TO THE ASTERIX SOFTWARE COLLECTION

New applications will be added to ASTERIX by Richard Saxton (LTVAD::RDS) or Bob Vallance (BHVAD::RJV). It would be wise to contact them to discuss an idea for a new program before starting to code or convert it. Applications should be developed and tested in private directories or in AST_TEST. Application-specific subroutines may be written at will, and should be given names of the form applic_suffix, where 'applic' is the name of the application. Subroutines which might be useful elsewhere in the system will be incorporated into ASTLIB. These should have names starting with one of the standard ASTLIB prefixes. Avoid the catch-all UTIL_* category if at all possible. If you feel the need for a new prefix category then contact RDS or RJV.

Applications should be tested as thoroughly as possible, on a wide range of input data, before being submitted for inclusion in the system. Some test data is available in directory AST_DEMO.

The following components will be required when submitting a program for inclusion in ASTERIX: the source code for the application and any new subroutines, an interface file and a HELP file.

8 PROVIDING DOCUMENTATION

Documentation should be provided at four levels:

- In-code comments (and reasonably structured programming) to make the code comprehensible.
- A description of the function and algorithm involved in the Description and Method sections of the program header.
- Online help (one line per parameter) in the interface file.
- A detailed help file (which can be partly adapted from the program header) for inclusion in ASTHELP. See PROG_008 for some further comments on this.

9 REFERENCES

```
ASTERIX documents
  USER_001: ASTERIX user guide
  PROG_001: ASTERIX Programming Standards
  PROG_002: ASTERIX standard structures and conventions
  PROG_003: The BDA_ binned data access routines
  PROG_004: The DYN_ dynamic memory routines
  PROG_005: The USI_ user interface routines
  PROG_006: The HIST_ history routines
  PROG_008: Notes on writing ASTHELP
  PROG_009: ASTERIX directory structure
```

Starlink documents

SG4:	ADAM - The Starlink Software Environment
SGP38:	Starlink Standard Data Structures
SUN39:	PRIMDAT - Processing of Primitive Numerical Data
SUN40:	CHR - Character Handling Routines
SUN67:	SLALIB - a library of subprograms
SUN92:	HDS Programmer's Guide