SCHOOL OF PHYSICS AND ASTRONOMY
UNIVERSITY OF BIRMINGHAM
Asterix Data Analysis
Cook Book 001

COOK/001

Richard Beard
22 January 1998

Introduction to DIY ASTERIX programming

Contents

1 Introduction

This documents is designed to allow the Fortran programmer to manipulate ASTERIX data files from within their own code. The example covered here, though basic, can be easily adapted to perform a variety of useful operations.

The first document is a basic introduction to the concepts of ASTERIX programming. It explains in detail how to construct a simple ASTERIX application; using as an example a program that reads in a dataset, modifies its values in some mathematical way and then updates the original file.

It also covers such topics such as program structure, compilation, linking and execution which are common to all ASTERIX applications. The next document will provide a more comprehensive tutorial on the reading and writing of various data types, axes, errors, quality and other basic information.

2 Requirements

A general working knowledge of both Fortran and UNIX is assumed. If the user has no previous experience with Starlink programming then it is suggested they consult the "For Programmers" section of Starlink Guide 4, "ADAM - The Starlink Software Environment", should they require further knowledge on the concepts and methods covered (http://star-www.rl.ac.uk/star/docs/sg4.htx/sg4.html).

The following ingredients are requires to create the application recipe supplied. It is assumed that ASTERIX and Starlink are installed on the local machine.

Starlink Utilities

- alink - to link the compiled code.
- compifl - to compile the interface file.

ASTERIX Utilities

- ast_link_adam - for creating a list of the various Starlink and ASTERIX libraries necessary to link the application - available in the $AST_BIN directory.

ASTERIX Files

- adi_par - a Fortran include file specifying the value of various constants. Look on the FTP or WWW site (details below) to obtain a copy.

ASTERIX Libraries

- libastlib.a & libastlib_adam.a - available in the $AST_LIB directory.

Copies of the various files used can be obtained from the Birmingham FTP site (ftp.sr.bham.ac.uk, directory pub/asterix/cookbook) or WWW site (http://www.sr.bham.ac.uk/asterix/cookbook/). The necessary libraries should all be available though the local installations of ASTERIX and Starlink.

3 General Overview

ASTERIX code is written under the rules of Starlink programming, a standard to which it generally conforms with 99% of the time. Thus much of the code is written as calls to various subroutines. These provide various facilities designed to make the job of code writing as simple as possible. The outline of this first application, to open an ASTERIX data file, read some values from it, modify those values and update the original file - is as follows;

- All ASTERIX applications are written as a subroutine. The main code from which this is called is automatically and invisibly generated during the linking stage.
- The ASTERIX system must be initialised. This also takes care of all other Starlink initialisations.
- A file name must be supplied and this file then located and opened. All this is taken care of by the parameter sub-system (see Starlink User Note 114, PAR - Interface to the ADAM Parameter System, for more details).
- The required data values are located and mapped within the data file. This is done through the use of pointers for the opened file and mapped data.
- A further subroutine is called to expand and evaluate the obtained data pointers, such that use can be made of the values to which they point.
- The ASTERIX system is closed down. This releases all resources, pointers, unmaps data values and closes opened files.

4 Application Code

Full code for the `simple` application and all necessary subroutines is listed here, interspersed with more comprehensive comments and notes written in *italics*. A copy of this file can be obtained from the Birmingham FTP or WWW site. The code has been reduced to the almost-minimum required - the next document will cover more comprehensive methods that perform extra checking and involve access axis, error and quality information.

```
      SUBROUTINE SIMPLE( STATUS )
```

All ASTERIX applications are written as subroutines with an INTEGER status value as their sole argument. The program wrapper is generated during linking.

```
*  Type Definitions:
      IMPLICIT NONE

*  Global Constants:
      INCLUDE '/star/include/sae_par'
      INCLUDE 'adi_par'
```

Choose a central place to store the ASTERIX include files (eg. /star/asterix/include) and either create a symbolic link (%ln -s /star/asterix/include/adi_par) or specify that directory when compiling (%f77 -c simple.f -I/star/asterix/include).

```
*  Status:
      INTEGER            STATUS

*  Local variables:
      INTEGER            IFID                ! Input file identifier
      INTEGER            IDPTR               ! Input data array pointer
      INTEGER            NDIM                ! Number of dimensions
      INTEGER            DIMS(ADI__MXDIM)    ! Size of dimensions

      LOGICAL            DATAOK              ! Is the data present?

*  Initialise ASTERIX
      CALL AST_INIT()
```

Must be called before any serious work can start.

```
*  Obtain name of data file, and get an identifier to it
      CALL USI_ASSOC( 'INP', 'BINDS', 'UPDATE', IFID, STATUS )
      IF ( STATUS .NE. SAI__OK ) RETURN
```

ASSOC implies that a pointer is to be associated with a file. INP is the parameter name as specified in the interface file. BINDS is the type of data expected in this file (a binned data set). UPDATE instructs the file to be opened for reading and writing. IFID is the pointer to the opened file.

```
*  Look for components.
      CALL BDI_CHK( IFID, 'Data', DATAOK, STATUS )
      IF ( .NOT. DATAOK ) STATUS = SAI__ERROR
```

The file is checked to see if contains a data section (ie. a component called 'INP'.DATA_ARRAY).

```
*  Get shape of input data
      CALL BDI_GETSHP( IFID, ADI__MXDIM, DIMS, NDIM, STATUS )
```

Get the dimensionality and size of the main data component in the file.

```
*  Map the data
      CALL BDI_MAPR( IFID, 'Data', 'UPDATE', IDPTR, STATUS )
```

The data component is mapped to the pointer IDPTR as REAL values and for both reading and writing.

```
*  Modify the array
      CALL SIMPLE_TASK( %VAL(IDPTR), NDIM, DIMS, STATUS )
```

To use the values mapped by a pointer it must first be eVALuated by passing it as an argument to a subroutine.

```
*  Tidy up
      CALL AST_CLOSE()
      CALL AST_ERR( STATUS )
```

Closing down ASTERIX unmaps the data and file pointers, closes the opened file, and frees all other resources that have been used internally by ASTERIX and Starlink. The modified data values are automatically written back to the data file, as it was opened for UPDATE and the data mapped for UPDATE. Then, if any errors have been encountered, print an explanatory message to the screen.

```
      END

      SUBROUTINE SIMPLE_TASK( DATA, NDIM, DIMS, STATUS )

*  Global constants:
      INCLUDE '/star/include/sae_par'

*  Status:
      INTEGER           STATUS

*  Import:
      REAL              DATA(*)             ! Array of data values
```

The data pointer has now become a standard data type after evaluation.

```
      INTEGER           NDIM                ! Number of dimensions
      INTEGER           DIMS(*)             ! Size of dimensions
*  Local variables:
      INTEGER           NELM                ! Number of data elements
      INTEGER           I                   ! Loop variable

*  Check inherited global status
      IF ( STATUS .NE. SAI__OK ) RETURN

*  Get size of the data array
      CALL ARR_SUMDIM( NDIM, DIMS, NELM )
```

Find the total number of data values by multiplying up the dimensions.

```
* Perform the task as a vector operation on the data
      DO I = 1, NELM
        DATA(I) = SQRT( DATA(I) )
      END DO
```

The pointer has been evaluated as a 1-D data array to allow for simple processing of the values. The actually file may contain a scalar value or an array with from 1 to 7 dimensions.

```
      END
```

5 Compilation and Linking

All the necessary compilation and linking is handled by the Starlink utility `alink`. The general format of an `alink` command is,

```
% alink code options libraries
```

- *code* - a list of filenames containing the code to be compiled. Can be either a Fortran file (`simple.f`) or an object file (`simple.o`).
- *options* - compiler options for the linking stage, common choices are `-g` for de-bugable code and `-O` for optimised code.
- *libraries* - a list of libraries required for the linking stage. The ASTERIX utility `ast_link_adam` is used here, enclosed in 'back-quotes', causing it to be expanded on the command line. On the Birmingham Suns this produces,

```
-L/star/asterix/sun4_Solaris/lib -lastlib_adam -lastlib -lwfclib -lcat \
-L/star/lib -lfitsio -lref -lhds -lprm -lprm_a -lard_adam -lsla -lgrp_adam \
-lgrp -lerr -lerr_adam -lpsx -lcnf -lfio_adam -lfio -lastpda -lpda -lems -lchr
```

Thus the executable application `simple` can be produced with the following command,

```
% alink simple.f `ast_link_adam`
dtask_applic.f:
        dtask_applic:
./simple.f:
        simple:
        simple_task:
Linking:
```

All trace of the `dtask_applic` wrapper is removed after linking. The final executable binary will have a size of 2+ Mb due to all the linked libraries.

6 Interface Files

All ASTERIX applications require an interface for execution. It is used to create an easily adapted link between program variables and application execution by defining prompts, help messages, argument order and expected data type.

If such a file cannot be found in the same directory as the executable is located then the following error message will be issued.

```
!! Interface file for simple not found
!
!! DTASK_DCLTASK: failed to start parameter system SUBPAR__IFNF, failed to find
!     interface module
```

An interface file has the same name as the application, with a file extension of `.ifl` for an ASCII file or `.ifc` for a compiled binary file (see Starlink User Note 115, ADAM - Interface Module Reference Manual, for more information on this topic). The utility `compifl` is used to compile `simple.ifl`. When running the application either type of interface file is valid, but the compiled version ensures correct syntax.

Here is the interface file for `simple` (also available at the Birmingham FTP or WWW site);

```
INTERFACE SIMPLE
  PARAMETER INP
    POSITION 1
```

```
    TYPE 'LITERAL'
    PROMPT 'Dataset'
    HELP 'Enter the filename of an SDF data file'
  ENDPARAMETER
ENDINTERFACE
```

The various fields supplied are explained below;

- INTERFACE - The interface is defined for the application `simple`.
- PARAMETER - Its has only one possible parameter, which will be referenced through the name *INP* from within the code.
- POSITION - The value for *INP* can be supplied on the command line, in which case it must be the first argument given.
- TYPE - The argument type is of a *LITERAL* unquoted string.
- PROMPT - The string 'Dataset' is used as a prompt.
- HELP - The given text is printed if the user enters '?'.

7 Execution

To test the aplication first obatin the file `simple1.sdf` from the Birmingham FTP or WWW site (binary transfer mode). Then make a copy of it for comparison.

```
% cp simple1.sdf simple2.sdf
% htrace simple1
HTRACE Version 2.1-0
 SIMPLE           <BINDS>        {structure}
   DATA_ARRAY[50,50] <_REAL>        128902.,129698.,128552.,131797.,
                                    ... 126910.,123719.,129015.,126602.,130871.
```

Next run the application on the copy of this file to modify its values.

```
% simple simple2
% htrace simple2
HTRACE Version 2.1-0
 SIMPLE           <BINDS>        {structure}
   DATA_ARRAY[50,50] <_REAL>        359.029,360.137,358.541,363.038,
                                    ... 356.244,351.738,359.187,355.812,361.761
```

The following example shows the interface file and error reporting in action,

```
% simple
INP - Dataset > ?
Enter the filename of an SDF data file
INP - Dataset > simple3
!! ADI_FOPEN : Error opening file object
!  Unable to associate parameter INP with file simple3
```