SCHOOL OF PHYSICS AND ASTRONOMY
UNIVERSITY OF BIRMINGHAM

PROG/017

Asterix Data Analysis
Programming Note 017

Richard Beard
June 23, 1998

Defining and Calling ASTERIX Methods

Contents

1 Introduction

The execution of many high level operations in ASTERIX depends upon the type and format of the data being passed to these subroutines. The most common cases are those of I/O to either a HDS or a FITS file, and the difference between data types such as images, spectra or time series. Thus a particular *method* might be to output data quality information, but the *execution* of that method differs depending upon the file and/or data type.

2 Defining a New Package

All ASTERIX methods are grouped into packages. Each package has assigned to it a different number, defined as an integer constant in the include file `/star/asterix/kernel/lib/inc/AST_PKG`. To define a new set of methods, simply add a unique identifier and number to this file, eg.

```
INTEGER                   NEW__PKG
  PARAMETER               ( NEW__PKG = 123 )
```

If adding more methods and/or definitions to an existing package then no changes need be made to this file.

3 Method Definitions

Before any particular method can be used the argument list it requires must first by defined by the calling application. This is done with a call to an initialisation routine called, in this example `NEW0_INIT`. Some of the lines it contains are,

```
IF ( AST_QPKGI( NEW__PKG ) ) RETURN
```

Logical function that returns `.TRUE.` *if the package is already defined.*

```
CALL ADI_REQPKG( 'new', STATUS )
```

Requires the use of data models defined in the file `new.adi`.

```
CALL ADI_DEFFUN( 'NewRead(_Image,_HDSfile)', NEW1_IMREAD, DID, STATUS )
CALL ADI_DEFFUN( 'NewRead(_,_HDSfile)', NEW1_READ, DID, STATUS )
CALL ADI_DEFFUN( 'NewRead(_Image,_FITSfile)', NEW2_IMREAD, DID, STATUS )
CALL ADI_DEFFUN( 'NewRead(_,_FITSHDSfile)', NEW2_READ, DID, STATUS )
```

Arguments are; DEFINITION, SUBROUTINE, DUMMY POINTER, STATUS

The method `NewRead` requires two arguments to be passed, the first describing the type of data to be read and the second the type of file to be accessed. When this method is called its arguments are checked against all stored definition of `NewRead`, and the first one to match the type of both arguments is executed.

Thus if the user requires data to be read in from a HDS files to an `Image` structure pointer then the subroutine `NEW1_IMREAD` will be called. If, however, the data is to be read into any other type of object (as indicated by '_') then the subroutine `NEW1_READ` is called instead.

3.1 Data models

A data model is a template used by ADI to create data structures for the storage and manipulation of information. Such a structure for a HDS container file stores the file name, access mode, locator for the container file, sub-structure within the file and the depth at which information is being accessed.

The data model for the *new* package is defined in the file `new.adi`, which is stored at the location `/star/asterix/kernel/etc/` and installed in the directory `$AST_ETC`.

The model contains the following definitions

```
;; The rest of this line is a comment
;;
defclass Data {
  CHAR      Type,                     ;; Basic data type
            Shape,                    ;; Its dimensions
  LOGICAL   Modified = False,         ;; Has the data been altered?
            Value                     ;; The data
}

defclass Image Data {
  CHAR      Name,                     ;; Name of object
}
```

The class `Image` extends the class `Data` as defined earlier in the file. This corresponds to the string `_Image` as used in calls to the subroutine `ADI_DEFFUN`. Each definition line has the following format,

TYPE NAME [= Initial value]

Where *TYPE* is one of LOGICAL, [U]BYTE, INTEGER, REAL, CHAR or STRUC. The type STRUC refers to an object previously defined with a call to `defclass`. Where the *TYPE* is blank a pointer will be used such that arrays can be stored (as with `Shape` and `Value` or the type of data required can be decided at run-time.

4 Package Initialisation

For a particular package to be available the method definitions as detailed above must be loaded at run-time. This should be done only once for each package, and to avoid overheads during application start-up is done on an as-required basis. The following section of code from `NEW_READ` ensures that the package is loaded before any of its methods are called.

```
*  Global Constants:
      INCLUDE 'AST_PKG'

*  External References:
      LOGICAL                  AST_QPKGI
        EXTERNAL               AST_QPKGI

*  Check initialised
      IF ( .NOT. AST_QPKGI( NEW__PKG ) ) CALL NEWO_INIT( STATUS )
```

5 Calling a Method

To access a previously defined and loaded method the following code is used, usually called from the subroutine NEW_READ,

```
*  Local Variables:
      INTEGER                   ARGS(2)                 ! Function arguments
      INTEGER                   OARG                    ! Return value

*    Fill in the arguments
      CALL ADI_NEW0( 'Image', ARGS(1), STATUS )
      ARGS(2) = FILE_ID
      OARG = ADI__NULLID

*    Invoke the function
      CALL ADI_FEXEC( 'NewRead', 2, ARGS, OARG, STATUS )
```

The call to ADI_NEW0 creates a new pointer to an object of type `Image`, the value of which is stored as the first arguement. The pointer to the file to be accessed is assinged as the second argument (here we assume from a HDS file, which implies the ADI type will be `HDSfile`.

The previously defined methods are now searched for a fuction which matches the types `Image` and `HDSfile`, and as can be seen from above this corresponds to the function NEW1_IMREAD. The subroutine is of the form,

```
      SUBROUTINE NEW1_IMREAD( NARG, ARGS, OARG, STATUS )
...
*  Arguments Given:
      INTEGER                   NARG, ARGS(*)

*  Arguments Returned:
      INTEGER                   OARG

*  Status:
      INTEGER                   STATUS                  ! Global status
...
      DATA_ID = ARGS(1)
      FILE_ID = ARGS(2)
...
c     read the information into the data identifier
...
      CALL ADI_NEWV0I( NELM, OARG, STATUS )

      END
```

Here the return argument is made to point at an integer which holds the number of data points read in. This information can be extracted up return using the following piece of code,

```
      CALL ADI_GET0I( OARG, NELM, STATUS )
```