DEPARTMENT OF SPACE RESEARCH
BIRMINGHAM UNIVERSITY
Asterix Data Analysis
ASTERIX Programming Note 005

PROG/005

P Andrews and DJ Allan
10 Febuary 1989

USI_ Access and Related Calls

Contents

1 Introduction

These are intended to provide an i/o interface for applications.

The USI_ASSOC routines now have an ACCESS argument, a character string with meaningful values of READ, WRITE, and UPDATE, to specify how the input object is accessed. Output objects are always opened for UPDATE.

The common block holding the necessary deinition is USI_CMN. It is included with the line,

```
INCLUDE 'USI_CMN'
```

2 Routine Descriptions

USI_INIT - Initializes USI_CMN

```
SUBROUTINE USI_INIT
```

Either this or AST_INIT must be called by each application.

USI_ASSOCI - Associates one input object.

```
SUBROUTINE USI_ASSOCI (PAR, ACCESS, LOC, INPRIM, STATUS)
```

INPRIM is a logical and is TRUE if the input is a primitive object.

USI_ASSOCO - Associates one output object.

```
SUBROUTINE USI_ASSOCO (PAR, TYPE, LOC, STATUS)
```

TYPE is the HDS type of the object to be created.

USI_ASSOC2 - Associates one input and creates one output object.

```
      SUBROUTINE USI_ASSOC2 (PAR1, PAR2, ACCESS, LOC1, LOC2, INPRIM,
     :                                                         STATUS)
```

This is a wrap up of USI_ASSOCI and USI_ASSOCO. TYPE is determined from the type of the input object (or its parent object if primitive).

USI_ASSOC3 - Associates two input and creates one output object

```
      SUBROUTINE USI_ASSOC3 (PAR1, PAR2, PAR3, ACCESS1, ACCESS2, LOC1,
     :                      LOC2, LOC3, SAME12, INPRIM1, INPRIM2, STATUS)
```

This is a wrap up of two USI_ASSOCI calls and one USI_ASSOCO call. TYPE is taken from the first input object (as above).

USI_NAMEI - Returns file and path names of input object(s) obtained via the USI_ASSOCx routines.

```
      SUBROUTINE USI_NAMEI (NLINES, TEXT, STATUS)
```

The character array TEXT should allow for 4 lines per input object.

USI_NAMEO - Returns file and path names of output object(s) obtained via the USI_ASSOCx routines.

```
      SUBROUTINE USI_NAMEO (NLINES, TEXT, STATUS)
```

The character array TEXT should allow for 4 lines per output object.

USI_RANGES - Allows range selection on each dimension of the data array

```
SUBROUTINE USI_RANGES (NDIM, LABEL, START, STOP, WIDTH, APP_DIM,
     :                                                 AMIN, AMAX, STATUS)
```

The following routines need to be explained in greater detail. When any of the USI_ASSOCx routines associate an object with a program parameter, the name of the object is stored. The names can be accessed later in the program in one of two ways.

The first is to use the USI_NAMEx routines. These output details of all the objects ASSOCed so far in a given access mode. These routines have certain disadvantages. The most obvious is that the programmer has no control over the precise contents of the text produced - it is called either 'input' or 'output'. The text could be edited, but this is cumbersome and hard codes assumptions about the number of text lines used.

The two routines below work on a common principle. Each processes text looking for substrings of the form {X}, where X is the name of a parameter ASSOCed using USI_ASSOCx. When such a reference is found, the name of the object is expanded into the filename and the component name if needed.

USI_SHOW - Displays the name of a parameter using the string given.

```
Invoked :
  SUBROUTINE USI_SHOW (INTEXT, STATUS)

Arguments :
  CHAR*(*)   INTEXT       A string containing a parameter reference.

Example :
  Suppose a parameter 'INP' had been associated with TEST.DATA_ARRAY
  The call  CALL USI_SHOW ('Weights data {INP}', STATUS) would produce

  Weights data file :
    DISK$USER1:[USER]TEST.SDF;2
  Weights data object :
    TEST.DATA_ARRAY
```

USI_TEXT - Processes an array of text strings expanding any parameter references found. This is especially useful for creating history text.

```
Invoked :
  SUBROUTINE USI_TEXT (INLINES, TEXT, OUTLINES, STATUS)

Arguments :
  INTEGER    INLINES      The number of text lines to be processed.

  CHAR*(*)   TEXT()       On entry contains an array of strings
                          possibly containing parameter references. On
                          exit contains the expanded text strings.

  INTEGER                 On entry, contains the maximum number of
                          lines which TEXT be expanded to - ie. the
                          size it was declared as. On exit contains
                          the actual number of text lines used.

  INTEGER     STATUS      Status is set bad if the text array
                          overflows.
```

3 Example

The following is typical code sequence using USI_TEXT. The array 'TEXT' is declared as TEXT(MAXLINES)*(string length). Note that not all lines need contain a parameter reference.

```
TEXT(1) = 'Input data {INP}'
TEXT(2) = 'Weights data {WEIGHTS}'
TEXT(3) = 'Errors assumed Poissonian'
```

```
NLINES = MAXLINES
CALL USI_TEXT  (3, TEXT, NLINES,    STATUS)
CALL HIST_PTXT (OLOC, NLINES, TEXT, STATUS)
```