DEPARTMENT OF SPACE RESEARCH
BIRMINGHAM UNIVERSITY
Asterix Data Analysis
ASTERIX Programming Note 013

PROG/013

DJ Allan
7 December 1991

Source Search Data Access Routines

Contents

1 Introduction

The set of subroutines described here are designed to allow access to ASTERIX source search results datasets (SSDS). They supersede the 1st generation routines which followed the "think of everything" approach, which failed conspicuously to keep up with with the functionality required of them.

The SSO_ subroutines have been designed to be as few and as flexible as possible - they are not optimised for speed as few applications use more than one or two of the routines.

The facilities provided by the SSO package are designed to map easily onto a typical database access package - it is hoped that the need for these routines will eventually be removed by a Starlink database system and thus provide the applications which use them a more flexible choice of data format.

2 The contents of an SSDS

The SSDS represents a specialised database - the storage of source search results. Following the usual database jargon, a record comprises the information on one source, and a field a piece of information present in all records. By using HDS, the SSO file format is field oriented (ie. accessing all the data in one field for all records is considerably easier than accessing all the fields for one record).

The structure of an SSDS can be described schematically,

```
<SSDS>
  <BOOK>[]
    <MORE>
    <PARAMETERS>
  <NSRC>
  <FIELD>
    <DATA_ARRAY>
    [<ERROR>]
    [<ITEMS>]...
```

The BOOK structure (short for book-keeping) contains information propagated from the dataset searched to create the SSDS - currently the MORE box is copied. In addition, database parameters are written into BOOK - eg. the version of the program which created the SSDS, the parameters which were used to run the program, and others. If the file type is SSDS_SET, then BOOK is actually an array of structures holding all the book-keeping data from its constituent SSDS's.

NSRC is the number of records in the SSDS - it may be zero.

Each SSDS field may contain several components. The only one required is the DATA_ARRAY. This is a primitive array of values, whose last (and usually only) dimension is the value of NSRC. A field may optionally contain an ERROR array giving errors on the field values. The ¡ITEMS¿

components are extra information required to interpret the field values or errors correctly. The SSO system does not standardise the names of these ITEMS, but the the following are the only ones currently used.

```
Item      Type      Contents

UNITS   _CHAR*40  The units of the field value
ELEVS     _REAL[]   The error confidence levels of the ERROR data
NULLERROR _REAL     The value used to denote a "bad" field error
NULLVALUE any       The value used to denote a "bad" field value
```

2.1 Reserved Field Names

Many ASTERIX make assumptions about field names in SSDS, usually to do with spatial information. Linearised detector coordinates are assumed to be in fields called X_CORR and Y_CORR, right ascension and declination in RA and DEC. The general rule is to inherit the axis units from the dataset from which the SSDS is derived.

3 Functional Subsets

The routines are divided by function into the following subsets:

```
a)  INIT/CLOSE      - initialise and tidy-up internal system
b)  VALID,RELEASE   - check/release from internal system
c)  CHKxxx          - check existence of an SSDS object
d)  LOCxxx          - get locator to an SSDS object
e)  MAPxxx     - map a field data/error object
f)  UNMAPxxx     - unmap a field data/error object
g)  GETxxx          - get value of field item/parameter
h)  PUTxxx          - write value of field item/parameter/NSRC
i)  CRExxx          - create a field/field error/NSRC
```

4 How to Use Them

The use of the SSO routines is best illustrated by the small example application. It opens an input SSDS, copies to output, and adds a new field combining information from two of the existing fields.The status checks normal in such an application have been omitted for clarity.

```
CALL AST_INIT()                    ! Initialise
CALL SSO_INIT()

CALL SSO_ASSOCI( 'INP', 'READ', ILOC, SET, STATUS )  ! Get input and
CALL DAT_TYPE( ILOC, TYPE, STATUS )                  ! output and
```

```
      CALL USI_ASSOCO( 'OUT', TYPE, OLOC, STATUS )         ! copy one to
      CALL HDX_COPY( ILOC, OLOC, STATUS )                  ! the other

      CALL SSO_GETNSRC( ILOC, NSRC, STATUS )               ! Any records?
      IF ( NSRC .EQ. 0 ) GOTO 99

      CALL SSO_CHKFLD( ILOC, 'X_CORR', XOK, STATUS )       ! Input fields
      CALL SSO_CHKFLD( ILOC, 'Y_CORR', YOK, STATUS )       ! present?

      IF (XOK.AND.YOK) THEN
        CALL SSO_CREFLD( OLOC, 'RADIUS', '_REAL', STATUS ) ! Create and
        CALL SSO_MAPFLD( OLOC, 'RADIUS', '_REAL', 'WRITE', ! map field
                                     RPTR, STATUS )

        CALL SSO_MAPFLD( ILOC, 'X_CORR', '_REAL', 'READ',  ! Map input
                                     XPTR, STATUS )        ! fields
        CALL SSO_MAPFLD( ILOC, 'Y_CORR', '_REAL', 'READ',
                                     YPTR, STATUS )

        CALL SSO_CHKFITEM( ILOC, 'X_CORR', 'UNITS', UOK,   ! Units to
                                          STATUS )         ! be copied?
        IF ( UOK ) THEN
          CALL SSO_GETFITEMOC( ILOC, 'X_CORR', 'UNITS',    ! Copy units
                                   UNITS, STATUS )         ! Note extra arg
          CALL SSO_PUTFITEMOC( OLOC, 'RADIUS', 'UNITS',    ! needed when
                               40, UNITS, STATUS )         ! writing char
        END IF                                             ! strings

        CALL PYTHAG( NSRC, %VAL(XPTR), %VAL(YPTR),         ! See below
                           %VAL(RPTR), STATUS )

        CALL SSO_UNMAPFLD( ILOC, 'X_CORR', STATUS )        ! Unmap fields
        CALL SSO_UNMAPFLD( ILOC, 'Y_CORR', STATUS )
        CALL SSO_UNMAPFLD( OLOC, 'RADIUS', STATUS )

      END IF

      CALL SSO_RELEASE( ILOC, STATUS )                     ! Free files
      CALL SSO_RELEASE( OLOC, STATUS )

      CALL SSO_CLOSE()                                     ! Shutdown.
      CALL AST_CLOSE()

      SUBROUTINE PYTHAG( N, X, Y, R, STATUS )      ! The action
      REAL X(*),Y(*),R(*)
      IF (STATUS.EQ.SAI__OK) THEN
        DO I =1,N
          R(I)=SQRT(X(I)**2+Y(I)**2)
        END DO
      END IF
```

```
END
```

Points to note.

- Always enclose the application in the AST/SSO_INIT and AST/SSO_CLOSE pair. These routines ensure that common blocks are explicitly initialised and shutdown. This is essential for applications running inside monoliths.
- Communication with SSO is always using the same top-level locator for each file, just like BDA.
- Allow for the possibility that the SSDS contains no sources - a common occurrence with WFC data.
- Explicitly unmapping is not actually required, but does spread the load in time and free resources quicker

5 Short Routine List

The list below gives the arguments for all the SSO routines. The use of ";" delimits input and output arguments where appropriate. The type token <T> means all of "C","I","R","D","L", whereas "<T'>" means all of "I","R","D","L" (ie omitting "C").

- ```
  SSO_INIT()
  SSO_CLOSE()
  ```
- ```
  SSO_INIT()
  SSO_ASSOCI(PAR,MODE;LOC,SET,STATUS)
  SSO_VALID(LOC;OK,STATUS)
  SSO_RELEASE(LOC,STATUS)
  ```
- ```
  SSO_INIT()
  SSO_CHKBOOK(LOC;OK,NFILE,STATUS)
  SSO_CHKFLD(LOC,FLD;OK,STATUS)
  SSO_CHKFLDERR(LOC,FLD;OK,STATUS)
  SSO_CHKFITEM(LOC,FLD,ITEM;OK,STATUS)
  ```
- ```
  SSO_INIT()
  SSO_LOCBOOK(LOC,FILE;LOC,STATUS)
  SSO_LOCFLD(LOC,FLD;FLOC,STATUS)
  ```
- ```
  SSO_INIT()
  SSO_MAPFLD(LOC,FLD,TYPE,MODE;PTR,STATUS)
  SSO_MAPFLDERR(LOC,FLD,TYPE,MODE;PTR,STATUS)
  ```
- ```
  SSO_INIT()
  SSO_UNMAPFLD(LOC,FLD,STATUS)
  SSO_UNMAPFLDERR(LOC,FLD,STATUS)
  ```

- ```
  SSO_INIT()
  SSO_GETNSRC(LOC;NSRC,STATUS)
  SSO_GETFITEM0<T>(LOC,FLD,ITEM;VALUE,STATUS)
  SSO_GETFITEM1<T>(LOC,FLD,ITEM,MAXVAL;VALUE,ACTVAL,STATUS)
  SSO_GETPAR0<T>(LOC,FILE,PAR;VALUE,STATUS)
  SSO_GETPAR1<T>(LOC,FILE,PAR,MAXVAL;VALUE,ACTVAL,STATUS)
  ```
- ```
  SSO_INIT()
  SSO_PUTNSRC(LOC,NSRC,STATUS)
  SSO_PUTFITEM0<T'>(LOC,FLD,ITEM,VALUE,STATUS)
  SSO_PUTFITEM0C(LOC,FLD,ITEM,LEN,VALUE,STATUS)
  SSO_PUTFITEM1<T'>(LOC,FLD,ITEM,NVAL,VALUE,STATUS)
  SSO_PUTFITEM1C(LOC,FLD,ITEM,LEN,NVAL,VALUE,STATUS)
  SSO_PUTPAR0<T'>(LOC,FILE,PAR,VALUE,STATUS)
  SSO_PUTPAR0C(LOC,FILE,PAR,LEN,VALUE,STATUS)
  SSO_PUTPAR1<T'>(LOC,FILE,PAR,NVAL,VALUE,STATUS)
  SSO_PUTPAR1C(LOC,FILE,PAR,LEN,NVAL,VALUE,STATUS)
  ```
- ```
  SSO_INIT()
  SSO_CREBOOK(LOC,NFILE,STATUS)
  SSO_CREFLD(LOC,FLD,TYPE,STATUS)
  SSO_CREFLDERR(LOC,FLD,TYPE,NDAT,NLEV,STATUS)
  ```

6 Linking with the SSO routines

The SSO routines reside in the ASTERIX library ASTLIB, and can be incorporated into an application using the standrard ASTERIX link options file ASTOPT, eg.

```
$ ASTDEV
$ ALINK <applic>,...,ASTOPT/OPT
```

7 Detailed Routine List

```
SSO_INIT()
```

Initialises SSO_ internal common block - should be run at beginning of application.

```
SSO_CLOSE()
```

Unmaps, annuls and resets everything leaving common blocks in a respectable state for following applications (in a monolith).
```

```
SSO_ASSOCI( PAR, MODE; LOC, SET, STATUS )

  PAR    <char>  The ADAM parameter name to associate
  MODE   <char>  Access mode, READ or UPDATE
 LOC <char> Locator to SSDS container structure
 SET <log> true if SSDS is a set
```

Associates an SSDS file with the ADAM parameter PAR, using access MODE. If input is not an SSDS, status is set bad.

```
SSO_RELEASE( LOC, STATUS )

 LOC <char> Locator to SSDS container structure
```

Frees all internal SSO storage and frees mapped field data.

```
SSO_VALID( LOC; OK, STATUS )

 LOC <char> Locator to SSDS container structure
  OK <log> True if LOC is a valid SSDS
```

Test whether LOC is a valid SSDS

```
SSO_CHKBOOK( LOC; OK, NFILE, STATUS )

 LOC <char> Locator to SSDS container structure
  OK <log> BOOK data exists?
  NFILE  <int>   Number of book components
```

```
SSO_CHKFLD( LOC, FLD; OK, STATUS )

 LOC <char> Locator to SSDS container structure
 FLD <char> The name of the field
  OK <log> Field data exists?
```

Checks the existence of the named field's data

```
SSO_CHKFLDERR( LOC, FLD; OK, STATUS )

 LOC <char> Locator to SSDS container structure
 FLD <char> The name of the field
  OK <log> Field error exists?
```

Checks the existence of the named field's error

```
SSO_CHKFITEM( LOC, FLD, ITEM; OK, STATUS )

 LOC <char> Locator to SSDS container structure
 FLD <char> The name of the field
 ITEM <char> The name of the field item
  OK <log> Field's item exists?
```

Checks the existence of the named field item

```
SSO_LOCBOOK( LOC; FILE, BLOC, STATUS )

 LOC <char> Locator to SSDS container structure
  FILE   <int>   Book component number
  BLOC   <char>  Locator to book component
```

Locate the FILE'th component of the BOOK structure in the specified SSDS

```
SSO_LOCFLD( LOC, FLD; FLOC, STATUS )

 LOC <char> Locator to SSDS container structure
 FLD <char> The name of the field
 FLOC <char> Locator to named field
```

Returns locator to the structured field component - note not the field data component.

```
SSO_MAPFLD( LOC, FLD, TYPE, MODE; PTR, STATUS )

 LOC <char> Locator to SSDS container structure
 FLD <char> The name of the field
  TYPE   <char>  The type to use in mapping
  MODE   <char>  Access mode, READ, WRITE or UPDATE
  PTR    <int>   The pointer to the mapped data
```

Maps the the field data of the field FLD using the specified TYPE.

```
SSO_MAPFLDERR( LOC, FLD, TYPE, MODE; PTR, STATUS )

 LOC <char> Locator to SSDS container structure
 FLD <char> The name of the field
  TYPE   <char>  The type to use in mapping
  MODE   <char>  Access mode, READ, WRITE or UPDATE
  PTR    <int>   The pointer to the mapped data
```

Maps the the field error data of the field FLD using the specified TYPE.

```
SSO_UNMAPFLD( LOC, FLD, STATUS )

 LOC <char> Locator to SSDS container structure
 FLD <char> The name of the field
```

Unmap the named field data

```
SSO_UNMAPFLDERR( LOC, FLD, STATUS )

 LOC <char> Locator to SSDS container structure
 FLD <char> The name of the field
```

Unmap the named field error

```
SSO_GETNSRC( LOC; NSRC, STATUS )

 LOC <char> Locator to SSDS container structure
  NSRC   <int>   Number of records
```

Returns number of record in the SSDS

```
SSO_GETFITEM0<T>( LOC, FLD, ITEM; VALUE, STATUS )

 LOC <char> Locator to SSDS container structure
 FLD <char> The name of the field
  ITEM   <char>  The field item name
  VALUE  <TYPE>  The field item value
```

Retrieves the value of the named scalar field item. The routine exists in 0C,0I,0R,0D and 0L forms.

```
SSO_GETFITEM1<T>( LOC, FLD, ITEM, MAXVAL; VALUE, ACTVAL, STATUS )

 LOC <char> Locator to SSDS container structure
 FLD <char> The name of the field
  ITEM   <char>  The field item name
  MAXVAL <int>   The maximum number of values to be read
  VALUE  <TYPE>  The field item value
  ACTVAL <int>   The actual number of values read
```

Retrieves the value of the named vector field item. The routine exists in 0C,0I,0R,0D and 0L forms.

```
SSO_GETPAR0<T>( LOC, FILE, PAR; VALUE, STATUS )

 LOC <char> Locator to SSDS container structure
  FILE   <int>   File number
  PAR    <char>  The parameter name
  VALUE  <TYPE>  The field item value
```

Retrieves the value of the named scalar parameter. The routine exists in 0C,0I,0R,0D and 0L forms.

```
SSO_GETPAR1<T>( LOC, FILE, PAR, MAXVAL; VALUE, ACTVAL, STATUS )

 LOC <char> Locator to SSDS container structure
  FILE   <int>   File number
  PAR    <char>  The parameter name
  MAXVAL <int>   The maximum number of values to be read
  VALUE  <TYPE>  The field item value
  ACTVAL <int>   The actual number of values read
```

Retrieves the value of the named vector parameter. The routine exists in 0C,0I,0R,0D and 0L forms.

```
SSO_PUTNSRC( LOC, NSRC, STATUS )

 LOC <char> Locator to SSDS container structure
  NSRC   <int>   Number of records
```

Sets the number of record in the SSDS. Note that this routine must be executed before any CREFLD or CREFLDERR calls.

```
SSO_PUTFITEM0C( LOC, FLD, ITEM, LEN; VALUE, STATUS )

 LOC <char> Locator to SSDS container structure
 FLD <char> The name of the field
  ITEM   <char>  The field item name
  LEN    <int>   The length of the character component
                 to be created, ie. _CHAR*<len>
  VALUE  <char>  The field item value
```

Writes the value of the named CHARACTER field item.

```
SSO_PUTFITEM1<T'>( LOC, FLD, ITEM; NVAL, VALUE, STATUS )

 LOC <char> Locator to SSDS container structure
 FLD <char> The name of the field
  ITEM   <char>  The field item name
  NVAL   <int>   Number of values
  VALUE  <TYPE>  The field item value
```

Writes the value of the named vector field item.

```
SSO_PUTPAR0C( LOC, FILE, PAR, LEN; VALUE, STATUS )

 LOC <char> Locator to SSDS container structure
  FILE   <int>   File number
 PAR <char> The name of the parameter
  LEN    <int>   The length of the character component
                 to be created, ie. _CHAR*<len>
  VALUE  <char>  The field item value
```

Writes the value of the named CHARACTER scalar parameter.

```
SSO_PUTPAR0<T'>( LOC, FILE, PAR; VALUE, STATUS )

 LOC <char> Locator to SSDS container structure
  FILE   <int>   File number
 PAR <char> The name of the parameter
  VALUE  <TYPE>  The field item value
```

Writes the value of the named scalar parameter. The routine exists in 0I,0R,0D and 0L forms.

```
SSO_PUTPAR1<T'>( LOC, FILE, PAR; NVAL, VALUE, STATUS )

 LOC <char> Locator to SSDS container structure
  FILE   <int>   File number
 PAR <char> The name of the parameter
  NVAL   <int>   Number of values
  VALUE  <TYPE>  The field item value
```

Writes the value of the named vector parameter.

```
SSO_CREBOOK( LOC, NFILE, STATUS )

 LOC <char> Locator to SSDS container structure
  NFILE  <int>   Number of book components
```

Create a new BOOK structure with NFILE components

```
SSO_CREFLD( LOC, FLD, TYPE, STATUS )

 LOC <char> Locator to SSDS container structure
 FLD <char> The name of the field
  TYPE   <char>  The type of the field
```

Create new field. Any existing field with the same name is deleted first.

```
SSO_CREFLDERR( LOC, FLD, TYPE, NDAT, NFLD, STATUS )

 LOC <char> Locator to SSDS container structure
 FLD <char> The name of the field
  TYPE   <char>  The type of the field
  NDAT   <int>   Number of data items per error level
  NLEV   <int>   Number of error levels
```

Create new field error. Any existing field with the same name is deleted first.