SUN/31.5

Starlink Project
Starlink User Note 31.5

R.F. Warren-Smith &
A.J. Chipperfield

27 April 1998

REF
Routines for Handling References to HDS Objects
Version 1.1
Programmer's Manual

Abstract

It is sometimes useful to use the Hierarchical Data System HDS (SUN/92) to store *references* or *pointers* to other HDS objects. For instance, this allows the same data object to be used in several places without the need to have more than one copy. The REF library is provided to facilitate this data object *referencing* process and the subsequent accessing of objects which have been referenced in this way.

Contents

1 Introduction

This package enables the user to store references to HDS objects in special HDS reference objects. Although it would be possible for users to concoct their own scheme, the use of this package will assist in portability and will in any case avoid re-inventing the wheel.

2 Facilities

The package allows reference objects to be created and written and it allows locators to referenced objects to be obtained.

The referenced object may be defined as *internal* in which case it is assumed to be within the same container file as the reference object itself, even if the reference object is copied to another container file. In that case the reference must point to an object which has the same pathname within the new file as it had in the old one. References which are not *internal* will point to a named container file.

Reference objects may be copied and erased using DAT_COPY and DAT_ERASE. Care must be taken when copying reference objects or referenced objects; otherwise the reference may no longer point to the referenced object.

Referenced objects must exist at the time the reference is made or used.

The following subroutines are available:

REF_CRPUT — Create a reference object and put a reference in it.

REF_FIND — Obtain locator to an object (possibly *via* a reference).

REF_GET — Obtain a locator to a referenced object.

REF_NEW — Create an empty reference object.

REF_PUT — Put a reference into a reference object.

REF_ANNUL — Annul a locator which may have been obtained *via* a reference.

3 Using the package

Two main uses for this package are foreseen:

(1) To maintain a catalogue of HDS objects.

(2) To avoid duplicating a large dataset.

As an example of the second case, suppose that a large dataset is logically required to form part of a number of other datasets. To avoid duplicating the common dataset, the others may contain a reference to it.

For example:

```
          Name                type                  Comments

       DATA                DATA_SETS
         .SET1             SPECTRUM
            .AXIS1         _REAL(1024)          Actual axis data
            .DATA_ARRAY    _REAL(1024)
         .SET2             SPECTRUM
            .AXIS1         REFERENCE_OBJ        Reference to DATA.SET1.AXIS1
            .DATA_ARRAY    _REAL(1024)
         .SET3             SPECTRUM
            .AXIS1         REFERENCE_OBJ        Reference to DATA.SET1.AXIS1
            .DATA_ARRAY    _REAL(1024)
          -
         etc.
```

Then a piece of code which handles structures of type SPECTRUM, which would normally contain the axis data in .AXIS1 (as SET1 does), could be modified as follows to handle an object .AXIS1 containing either the actual axis data or a reference to the object which does contain the actual axis data.

```
*    LOC1 is a locator associated with a SPECTRUM object
*    Obtain locator to AXIS data
      CALL DAT_FIND(LOC1, 'AXIS1', LOC2, STATUS)
*    Modification to allow AXIS1 to be a reference object
*    Check type of object
      CALL DAT_TYPE(LOC2, TYPE, STATUS)
      IF (TYPE .EQ. 'REFERENCE_OBJ') THEN
          CALL REF_GET(LOC2, 'READ', LOC3, STATUS)
          CALL DAT_ANNUL(LOC2, STATUS)
          CALL DAT_CLONE(LOC3, LOC2, STATUS)
          CALL DAT_ANNUL(LOC3, STATUS)
      ENDIF
*    End of modification
*    LOC2 now locates the axis data wherever it is.
```

This code has been packaged into the subroutine **REF_FIND** which can be used instead of DAT_FIND in cases where the component requested may be a reference object.

When a locator which has been obtained in this way is finished with, it should be annulled using REF_ANNUL rather than DAT_ANNUL. This is so that, if the locator was obtained *via* a reference, the HDS_OPEN for the container file may be matched by an HDS_CLOSE. *Note that this should only be done when any other locators derived from the locator to the referenced object are also finished with.*

4 Implementation

The way in which the package is implemented is described here for interest. Programmers should not make use of this information; otherwise portability is compromised.

A reference object is an HDS structure of type `REFERENCE_OBJ` with two components, `FILE` and `PATH`, of type `_CHAR*(REF__SZREF)`. `REF__SZREF` is defined in the `REF_PAR` include file.

FILE contains the name of the container file for the referenced object. This is set to spaces if the reference is *internal*.

PATH contains the pathname of the referenced object (as supplied by HDS_TRACE). The name of the top level component of the pathname will not be used in finding the locator for the referenced object. This fact allows structures containing internal references to be copied but the path below the top level must still lead to an appropriate object.

Locators obtained *via* a reference are flagged as such by being linked to the group $$REFERENCED$ using the subroutine HDS_LINK. This fact is used by REF_ANNUL in determining whether or not HDS_CLOSE should be called for the container file of the object specified by the locator argument. Note that the effect of calling HDS_CLOSE is to counter the HDS_OPEN done in obtaining a locator to the referenced object. The container file will only be physically closed if the container file reference count goes to zero.

5 Error handling

The REF routines adhere throughout to the Starlink error-handling strategy described in the MERS document, SUN/104. Most of the routines therefore carry an integer inherited status argument called STATUS and will return without action unless this is set to the value SAI__OK[1] when they are invoked. When necessary, error reports are made through the EMS_ routines in the manner described in SSN/4. This gives complete compatibility with the use of ERR_ and MSG_ routines in applications (SUN/104).

6 Compiling and linking

Before compiling applications which use the REF library on UNIX systems, you should normally “log in” for REF software development with the following shell command:

```
% ref_dev
```

This will create links in your current working directory which refer to the REF include files. You may then refer to these files using their standard (upper case) names without having to know where they actually reside. These links will persist, but may be removed at any time, either explicitly or with the command:

[1]The symbolic constant SAI__OK is defined in the include file SAE_PAR.

```
% ref_dev remove
```

If you do not "log in" in this way, then references to REF include files should be in lower case and must contain an absolute pathname identifying the Starlink include file directory, thus:

```
INCLUDE '/star/include/ref_par'
```

The former method is recommended.

Applications which use the ADAM programming environment (SG/4) may be linked with the REF library by specifying `` `ref_link_adam` `` on the appropriate command line. Thus, for instance, an ADAM A-task which calls REF routines might be linked as follows:

```
% alink adamprog.f `ref_link_adam`
```

(note the use of backward quote characters, which are required).

"Stand-alone" (*i.e.* non-ADAM) applications which use the REF library may be linked by specifying `` `ref_link` `` on the compiler command line. Thus, to compile and link a stand-alone application called `'prog'`, the following might be used:

```
% f77 prog.f `ref_link` -o prog
```

A Routine Descriptions

REF_ANNUL
Annul a locator to a referenced object

Description:

This routine annuls the locator and, if the locator was linked to group $$REFERENCED$, issues HDS_CLOSE for the container file of the object.

Invocation:

```
CALL REF_ANNUL( LOC, STATUS )
```

Arguments:

LOC = CHARACTER ∗ (DAT__SZLOC) (Given and Returned)

Locator to be annulled.

STATUS = INTEGER (Given and Returned)

Inherited global status.

Notes:

This routine attempts to execute even if STATUS is set on entry, although no further error report will be made if it subsequently fails under these circumstances. In particular, it will fail if the locator supplied is not initially valid, but this will only be reported if STATUS is set to SAI__OK on entry.

REF_CRPUT
Create and write a reference object

Description:

This routine creates a reference object as a component of a specified structure and writes a reference to an HDS object in it. If the specified component already exists and is a reference object, it will be used. If it is not a reference object, an error is reported. The reference may be described as "internal" which means that the referenced object is in the same container file as the reference object.

Invocation:

```
CALL REF_CRPUT( ELOC, CNAME, LOC, INTERN, STATUS )
```

Arguments:

ELOC = CHARACTER ∗ (DAT__SZLOC) (Given)

A locator associated with the structure which is to contain the reference object.

CNAME = CHARACTER ∗ (DAT__SZNAM) (Given)

The component name of the reference object to be created.

LOC = CHARACTER ∗ (∗) (Given)

A locator associated with the object to be referenced.

INTERN = LOGICAL (Given)

Whether or not the referenced object is "internal". Set this to .TRUE. if the reference is "internal" and to .FALSE. if it is not.

STATUS = INTEGER (Given and Returned)

Inherited global status.

REF_FIND
Get locator to data object (via reference if necessary)

Description:

This routine gets a locator to a component of a specified structure or, if the component is a reference object, it gets a locator to the object referenced. Any locator obtained in this way should be annulled, when finished with, by REF_ANNUL so that the top-level object will also be closed if the locator was obtained via a reference.

Invocation:

```
CALL REF_FIND( ELOC, CNAME, MODE, LOC, STATUS )
```

Arguments:

ELOC = CHARACTER $*$ ($*$) (Given)
The locator of a structure.

CNAME = CHARACTER $*$ ($*$) (Given)
The name of the component of the specified structure.

MODE = CHARACTER $*$ ($*$) (Given)
Mode of access required to the object. ('READ', 'WRITE' or 'UPDATE'). This is specified so that the container file of any referenced object can be opened in the correct mode.

LOC = CHARACTER $*$ (DAT__SZLOC) (Returned)
A locator associated with the object found.

STATUS = INTEGER (given and Returned)
Inherited global status.

REF_GET
Get locator to referenced data object

Description:

This routine gets a locator to an HDS object referenced in a reference object and links it to the group $$REFERENCED$. Any locator obtained in this way should be annulled, when finished with, by REF_ANNUL so that the top-level object will also be closed.

Invocation:

```
CALL REF_GET( ELOC, MODE, LOC, STATUS )
```

Arguments:

ELOC = CHARACTER ∗ (DAT__SZLOC) (Given)
A locator associated with the reference object

MODE = CHARACTER ∗ (∗) (Given)
Mode of access required to the object. ('READ', 'WRITE' or 'UPDATE'). This is specified so that the container file of any referenced object can be opened in the correct mode.

LOC = CHARACTER ∗ (DAT__SZLOC) (Returned)
A locator pointing to the object referenced.

STATUS = INTEGER (Given and Returned)
Inherited global status.

REF_NEW
Create a new reference object

Description:

This routine creates a reference object as a component of a specified structure. If the component already exists, an error is reported.

Invocation:

```
CALL REF_NEW( ELOC, CNAME, STATUS )
```

Arguments:

ELOC = CHARACTER $*$ (DAT__SZLOC) (Given)
A locator associated with the structure which is to contain the reference object.

CNAME = CHARACTER $*$ (DAT__SZNAM) (Given)
The name of the component to be created in the structure located by ELOC.

STATUS = INTEGER (Given and Returned)
Inherited global status.

REF_PUT
Write a reference into a reference object

Description:

This routine writes a reference to an HDS object into an existing reference structure. An error is reported if an attempt is made to write a reference into an object which is not a reference object. The reference may be described as "internal" which means that the referenced object is in the same container file as the reference object.

Invocation:

```
CALL REF_PUT( ELOC, LOC, INTERN, STATUS )
```

Arguments:

ELOC = CHARACTER $*$ ($*$) (Given)
A locator associated with the reference object.

LOC = CHARACTER $*$ ($*$) (Given)
A locator associated with the object to be referenced.

INTERN = LOGICAL (Given)
Whether or not the referenced object is "internal". Set this to .TRUE. if the reference is "internal" and to .FALSE. if it is not.

STATUS = INTEGER (Given and Returned)
Inherited global status.

B ADAM/Stand-alone Differences

Note that when using the stand-alone version of the REF library, it is currently necessary to ensure that HDS_START is called to activate HDS prior to making calls to any REF routines. This requirement will be removed in future, and is currently not required with the ADAM version.

C Machine-dependent Features

The REF library contains no explicit use of machine-dependent features, so its behaviour should be the same on all platforms on which it is implemented.

However, external references to HDS objects (those not identified as "internal" to the REF library) will contain explicit file names, so true portability of data (in the manner provided by HDS) cannot be expected with REF when working with operating systems which have different file naming conventions. If complete data portability is required, then use of the REF library should be restricted to internal references only.

D Software Dependencies

The REF library explicitly depends on the following other Starlink packages:

HDS — Hierarchical data system (SUN/92)

EMS — Error message service (SSN/4)

Note that these packages may also depend on other sub-packages. Please consult the relevant documentation for details.