SUN/262.1

Starlink Project
Starlink User Note 262.1

D.S. Berry
Malcolm J. Currie

2009 August 11

CTG — Accessing Groups of catalogues Version 3.0 Programmer's Manual

---

Abstract

This document describes the routines provided within the CTG subroutine library for accessing groups of catalogues.

Contents

1 Introduction

If an application prompts the user for a catalogue (or table) using the facilities of the CAT_ system (see SUN/181), the user may only reply with the name of a single catalogue. Some applications allow many input catalogues to be specified and the need to type in every catalogue name explicitly each time the program is run can become time consuming. The CTG package provides a means of giving the user the ability to specify a list (or *Group*) of catalogues as a reply to a single prompt for a parameter.

2 Interaction Between CTG and GRP

CTG uses the facilities of the GRP package and users of CTG should be familiar with the content of SUN/150 which describes the GRP package. Groups created by CTG routines should be deleted when no longer needed using GRP_DELET.

3 General overview of the CTG_ system

As a broad outline, applications use the CTG_ package as follows:

(1) A call is made to CTG_ASSOC which causes the user to be prompted for a single parameter. This parameter can be of any type. The user replies with a *group expression* (see SUN/150), which contains the names of a group of *existing* catalogues to be used as inputs by the application For instance, the group expression may be

```
coma_b[1-3].fits,coma_r[45]s.txt,coma_b[23]?{2}.fit,^files.lis
```

This is a complicated example, probably more complicated than would be used in practice, but it highlights the facilities of the GRP and CTG packages, *e.g.* wild cards ("?", "∗" or "`[..]`"), lists of files, or indirection through a text file ("^"). The braces indicate the second FITS extension.

The CTG_ASSOC routine produces a list of explicit catalogue names, which are stored internally within the GRP system.

(2) What happens next depends on the application, but a common example may be the initiation of a DO loop to loop through the input catalogues (CTG_ASSOC returns the total number of catalogue names in the group).

(3) To access a particular catalogue, the application calls routine CTG_CATAS supplying an index, n, within the group (i.e n is an integer in the range 1 to the group size returned by CTG_ASSOC). CTG_CATAS returns a CAT identifier to the nth catalogue in the group. This identifier can then be used to access the catalogue in the normal manner using the CAT_ routines (SUN/181). The identifier should be released when it is no longer needed, and freeing resoures using CAT_TRLSE in the normal way.

(4) Once the application has finished processing the group of catalogues, it calls GRP_DELET which deletes the group, releasing all resources reserved by the group.

(5) Routine CTG_ASSOC can also be used to append a list of catalogue names obtained from the environment, to a previously defined group.

The routine CTG_CREAT produces a group containing the names of catalogues that are to be created by the application. The routine CTG_CATCR will create a new catalogue with a name given by a group member, and returns a CAT identifier to it.

The names of output catalogues given by users usually relate to the input catalogue names. When CTG_CREAT is called, it creates a group of catalogue names either by modifying all the names in a specified input group using a *modification element* (see SUN/150), or by getting a list of new names from the user.

(6) Applications which produce a group of output catalogues could also produce a text file holding the names of the output catalogues. Such a file can be used as input to the next application, using the indirection facility. A text file listing of all the catalogues in a group can be produced by routine GRP_LIST (or GRP_LISTF).

See the detailed descriptions of CTG_ASSOC and CTG_CREAT below for details of the processing of existing and new catalogue names.

4 An example CTG application

The following gives a short example of how CTG routines might be used within an ADAM task.

```
      SUBROUTINE COPY( STATUS )

*  Global Constants:
      INCLUDE 'GRP_PAR'          ! Standard GRP constants

*  Local Variables:
      INTEGER CIN                ! GRP identifier for an input catalogue
      INTEGER COUT               ! GRP identifer for an output catalogue
      LOGICAL FLAG               ! Has group ended with flag character?
      INTEGER GIDIN              ! GRP identifier for group of input cat's
      INTEGER GIDOUT             ! GRP identifer for group of output cat's
      INTEGER I                  ! Loop counter
      INTEGER NUMIN              ! Number of input catalogues
      INTEGER NUMOUT             ! Number of output catalogues
      INTEGER STATUS             ! The global status
*.

*  Inialise the group identifiers to indicate that groups do not have
*  any initial members.
      GIDIN = GRP__NOID
      GIDOUT = GRP__NOID

*  Create group of input catalogues using the parameter IN.
      CALL CTG_ASSOC( 'IN', .TRUE., GIDIN, NUMIN, FLAG, STATUS )
```

```
*  Create group of output catalogues using the parameter OUT, that
*  possibly works by modifying the values in the input group.
      CALL CTG_CREAT( 'OUT', GIDIN, GIDOUT, NUMOUT, FLAG, STATUS )

*  Loop over group members.
      DO I = 1, NUMIN

*  Get the identifier for an existing catalogue from the input group.
         CALL CTG_NDFAS( GIDIN, I, 'READ', CIN, STATUS )

*  Get the identifier for the output catalogue from the output group.
         CALL CTG_NDFAS( GIDOUT, I, 'WRITE', COUT, STATUS )

*  Proceed to create output catalogue by editing the input catalogue
*  using the CAT library.
               :            :            :            :

*  Release CAT resources.
         CALL CAT_TRLSE( CIN, STATUS )
         CALL CAT_TRLSE( COUT, STATUS )
      END DO

*  Release GRP resources.
      CALL GRP_DELET( GIDIN, STATUS )
      CALL GRP_DELET( GIDOUT, STATUS )

      END
```

When this program is compiled and run, the user is asked for two ADAM parameters, IN and OUT. Each catalogue in the list specified by the IN parameter is simply copied to the corresponding name in the list specified by the OUT parameter. For instance, running

```
copy in=data[12].fits out=*-new
```

would write new catalogues `data1-new.fits` and `data2-new.fits` which were edited copies of the existing files `data1.fits` and `data2.fits`. If `data1` and `data2` do not represent FITS catalogues an error will be signalled and the user will be prompted to enter a different value for IN.

Note that a few corners have been cut in the above code, in particular checking that the input and output groups have the same size and STATUS testing. Additionally, no action is taken when the FLAG character is given at the end of a group specification—conventionally this would indicate that the user should be allowed to add further members.

5 Compiling and Linking with CTG

This section describes how to compile and link applications which use CTG subroutines, on UNIX systems. It is assumed that the CTG library is installed as part of the Starlink Software Collection.

The library only has an ADAM interface to obtain the groups of catalogues.

5.1 ADAM Applications

Users of the ADAM programming environment (SG/4) should use the **alink** command (SUN/144) to compile and link applications, and can access the CTG_ library by including execution of the command `ctg_link_adam` on the command line, as follows:

```
% alink prog.f ‘ctg_link_adam‘
```

where `prog.f` is the Fortran source file for the A-TASK. Again note the use of opening apostrophies (‘) instead of the more usual closing apostrophy (’) in the above **alink** command.

To build a program written in C (instead of Fortran), simply name the source file `prog.c`, instead of `prog.f`.

A List of Routines

CALL CTG_ASSO1(PARAM, VERB, MODE, CI, FIELDS, STATUS)
Obtain an identifier for a single existing catalogue using a specified parameter.

CALL CTG_ASSOC(PARAM, VERB, IGRP, SIZE, FLAG, STATUS)
Store names of existing catalogues specified through the environment.

CALL CTG_CATAS(IGRP, INDEX, MODE, CI, STATUS)
Obtain a CAT identifier for an existing catalogue.

CALL CTG_CATCR(IGRP, INDEX, CI, STATUS)
Obtain a CAT identifier for a new catalogue.

CALL CTG_CREA1(PARAM, FTYPE, NDIM, LBND, UBND, CI, NAME, STATUS)
Create a single new catalogue using a specified parameter.

CALL CTG_CREAT(PARAM, IGRP0, IGRP, SIZE, FLAG, STATUS)
Obtain the names of a group of catalogues to be created from the environment.

CALL CTG_GTSUP(IGRP, I, FIELDS, STATUS)
Get supplemental information for a catalogue.

CALL CTG_PTSUP(IGRP, I, FIELDS, STATUS)
Store suplemental information for a catalogue.

CALL CTG_SETSZ(IGRP, SIZE, STATUS)
Reduces the size of an CTG group.

B Full Fortran Routine Specifications

CTG_ASSO1
Obtain an identifier for a single existing catalogue using a specified parameter

Description:

This routine is equivalent to CAT_ASSOC except that it allows the catalogue to be specified using a GRP group expression (for instance, its name may be given within a text file, *etc.*). The first catalogue in the group expression is returned. Any other names in the group expression are ignored. Supplemental information describing the separate fields in the catalogue specification are also returned.

Invocation:

```
CALL CTG_ASSO1( PARAM, VERB, MODE, CI, FIELDS, STATUS )
```

Arguments:

PARAM = CHARACTER * (*) (Given)
Name of the ADAM parameter.

VERB = LOGICAL (Given)
If TRUE then errors which occur whilst accessing supplied catalogues are flushed so that the user can see them before re-prompting for a new catalogue (*verbose* mode). Otherwise, they are annulled and a general `"Cannot access file xyz"` message is displayed before re-prompting.

MODE = CHARACTER * (*) (Given)
Type of catalogue access required: `'READ'`, `'UPDATE'` or `'WRITE'`.

CI = INTEGER (Returned)
catalogue identifier.

FIELDS(5) = CHARACTER * (*) (Given)
Each element contains the following on exit.

- – FITS extension specification (*e.g.*`"{3}"`) if any
- – File type
- – Base file name
- – Directory path
- – Full catalogue specification

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

CTG_ASSOC
Store names of existing catalogues specified through the environment

Description:

A group expression is obtained from the environment using the supplied parameter. The expression is parsed (using the facilities of the GRP routine GRP_GROUP,see SUN/150) to produce a list of explicit names for existing catalogues which are appended to the end of the supplied group (a new group is created if none is supplied). If an error occurs while parsing the group expression, the user is re-prompted for a new group expression. CAT identifiers for particular members of the group can be obtained using CTG_CATAS.

Invocation:

```
CALL CTG_ASSOC( PARAM, VERB, IGRP, SIZE, FLAG, STATUS )
```

Arguments:

PARAM = CHARACTER*(*) (Given)
The parameter with which to associate the group expression.

VERB = LOGICAL (Given)
If TRUE then errors which occur whilst accessing supplied catalogues are flushed so that the user can see them before re-prompting for a new catalogue *verbose* mode). Otherwise, they are annulled and a general `"Cannot access file xyz"` message is displayed before re-prompting.

IGRP = INTEGER (Given and Returned)
The identifier of the group in which the catalogue names are to be stored. A new group is created if the supplied value is GRP__NOID. It should be deleted when no longer needed using GRP_DELET.

SIZE = INTEGER (Returned)
The total number of catalogue names in the returned group.

FLAG = LOGICAL (Returned)
If the group expression was terminated by the GRP *flag character*, then FLAG is returned `.TRUE.`. Otherwise it is returned `.FALSE.`. Returned `.FALSE.` if an error occurs.

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

Notes:

- Any file names containing wildcards are expanded into a list of catalogue names. The supplied strings are intepreted by a shell (/bin/tcsh if it exists, otherwise /bin/csh, otherwise /bin/sh), and so may contain shell meta-characters (*e.g.* twiddle, $HOME, even command substitution and pipes - but pipe characters `"|"` need to be escaped using a backslash `"\"` to avoid them being interpreted as GRP editing characters).
- Only the highest priority file with any give file name is included in the returned group. The priority of a file is determined by its file type. Priority decreases along the following list of file types: .FIT, .fit, .FITS, .fits, .GSC, .gsc, .TXT, .txt, .Txt, .sdf. If no file type is given by the user, the highest priority available file type is used. If an explicit file type is given, then that file type is used.
- Names of catalogues stored in FITS format may include an FITS extension number. For instance, `"/home/dsb/mydata.fit{3}"` refers to a catalogue stored in the third extension of the FITS file `mydata.fit`.

- Catalogues stored in HDS format must be stored as the top level object within the .sdf file.
- All matching files are opened in order to ensure that they are valid catalogues. The user is notified if there are no valid catalogues matching a supplied name, and they are asked to supply a replacement parameter value.
- Each element in the returned group contains a full specification for a catalogue. Several other groups are created by this routine, and are associated with the returned group by means of a GRP *o*wner-slave relationship. These supplemental groups are automatically deleted when the returned group is deleted using GRP_DELET. The returned group should not be altered using GRP directly because corresponding changes may need to be made to the supplemental groups. Routines CTG_SETSZ, CTG_GTSUP and CTG_PTSUP are provided to manipulate the entire chain of groups. The full chain (starting from the head) is as follows:

  - FITS extension numbers (if any)
  - File types
  - Base file names
  - Directory paths
  - Full catalogue specification (this is the returned group IGRP)
- If an error is reported the group is returned unaltered. If no group is supplied, an empty group is returned.
- A null value (`!`) can be given for the parameter to indicate that no more catalogues are to be specified. The corresponding error is annulled before returning unless no catalogues have been added to the group.
- If the last character in the supplied group expression is a colon (`:`), a list of the catalogues represented by the group expression (minus the colon) is displayed, but none are actually added to the group. The user is then re-prompted for a new group expression.

CTG_CATAS
Obtain a CAT identifier for an existing catalogue

Description:

The routine returns a CAT identifier for an existing catalogue. The name of the catalogue is held at a given index within a given group. It is equivalent to CAT_ASSOC.

Invocation:

```
CALL CTG_CATAS( IGRP, INDEX, MODE, CI, STATUS )
```

Arguments:

IGRP = INTEGER (Given)

A GRP identifier for a group holding the names of catalogues. This will often be created using CTG_ASSOC, but groups created *by hand* using GRP directly (*i.e.* without the supplemental groups created by CTG_ASSOC) can also be used.

INDEX = INTEGER (Given)

The index within the group at which the name of the catalogue to be accessed is stored.

MODE = CHARACTER * (*) (Given)

Type of catalogue access required: `'READ'`, or `'WRITE'`.

CI = INTEGER (Returned)

catalogue identifier.

STATUS = INTEGER (Given and Returned)

The global status.

Notes:

- If this routine is called with STATUS set, then a value of CAT__NOID will be returned for the CI argument, although no further processing will occur. The same value will also be returned if the routine should fail for any reason. The CAT__NOID constant is defined in the include file `CAT_PAR`.

CTG_CATCR
Obtain a CAT identifier for a new catalogue

Description:

The routine returns a CAT identifier for a new catalogue. The name of the new catalogue is held at a given index within a given group. It is equivalent to CAT_CREAT, except that any existing catalogue with the specified name is first deleted (unless the catalogue specification includes a FITS-extension specifier).

Invocation:

```
CALL CTG_CATCR( IGRP, INDEX, CI, STATUS )
```

Arguments:

IGRP = INTEGER (Given)

A GRP identifier for a group holding the names of catalogues. This will often be created using CTG_CREAT, but groups created *by hand* using GRP directly can also be used.

INDEX = INTEGER (Given)

The index within the group at which the name of the catalogue to be created is stored.

CI = INTEGER (Returned)

Catalogue identifier.

STATUS = INTEGER (Given and Returned)

The global status.

CTG_CREA1
Create a single new catalogue using a specified parameter

Description:

This routine is equivalent to CAT_CREAT except that it allows the catalogue to be specified using a GRP group expression (for instance, its name may be given within a text file, *e*tc.), and it also ensures that any existing catalogue with the same name is deleted before the new one is created (so long as no FITS extension number is included in the catalogue specification). The first catalogue in the group expression is returned. Any other names in the group expression are ignored. Any modification elements in the supplied group expression will be treated literally.

Invocation:

```
CALL CTG_CREA1( PARAM, CI, NAME, STATUS )
```

Arguments:

PARAM = CHARACTER * (*) (Given)
Name of the ADAM parameter.

CI = INTEGER (Returned)
Catalogue identifier.

NAME = CHARACTER * (*) (Returned)
The file specification for the catalogue.

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

CTG_CREAT
Obtain the names of a group of catalogues to be created from the environment

Description:

A group expression is obtained from the environment using the supplied parameter. The expression is parsed (using the facilities of the GRP routine GRP_GROUP, see SUN/150) to produce a list of explicit catalogue names. These names are appended to the group identified by IGRP. The user is re-prompted if an error occurs while parsing the group expression. If IGRP has the value `GRP__NOID` on entry, then a new group is created and IGRP is returned holding the new group identifier.

If IGRP0 holds a valid group identifier on entry, then the group identified by IGRP0 is used as the basis for any modification element contained in the group expression obtained from the environment. If IGRP0 holds an invalid identifier (such as `GRP__NOID`) on entry then modification elements are included literally in the output group.

Invocation:

```
CALL CTG_CREAT( PARAM, IGRP0, IGRP, SIZE, FLAG, STATUS )
```

Arguments:

PARAM = CHARACTER*(*) (Given)
The parameter with which to associate the group.

IGRP0 = INTEGER (Given)
The GRP identifier for the group to be used as the basis for any modification elements. If a valid GRP identifier is supplied, and if the supplied group expression contains a modification element, then:

- the basis token (an asterisk) is replaced by the file basename associated with the corresponding element of the basis group (the *basis catalogue*); else
- if no directory specification is included in the group expression, the directory specification associated with the basis catalogue is used.

The supplied group will often be created by CTG_ASSOC, but groups created *by hand* using GRP directly can also be used (*i.e.* without the supplemental groups created by CTG). In this case, there are no defaults for directory path or file type, and the basis token (`"*"`) in the group expression represents the full basis file specification supplied in IGRP0, not just the file basename.

IGRP = INTEGER (Given and Returned)
The GRP identifier for the group to which the supplied files are to be appended.

SIZE = INTEGER (Returned)
The total number of file names in the returned group.

FLAG = LOGICAL (Returned)
If the group expression was terminated by the GRP *flag* character, then FLAG is returned `.TRUE.`. Otherwise it is returned `.FALSE.`. Returned `.FALSE.` if an error occurs.

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

Notes:

- Any FITS extensions specified in the group expression are ignored.
- If an error is reported the group is returned unaltered.
- A null value (`!`) can be given for the parameter to indicate that no more catalogues are to be specified. The corresponding error is annulled before returning unless no catalogues have been added to the group.
- If no file type is supplied in the group expression, then the first file type listed in the current value of the `CAT_FORMATS_OUT` environment variable is used. If this is `"*"` then the file type is copied from the corresponding input file if a modification element was used to specify the output file name (if the catalogue was not specified by a modification element, the second file type in `CAT_FORMATS_OUT` is used).
- If the last character in the supplied group expression is a colon (`:`), a list of the catalogues represented by the group expression (minus the colon) is displayed, but none are actually added to the group. The user is then re-prompted for a new group expression.
- The returned group has no associated groups holding supplemental information (unlike the group returned by CTG_ASSOC).

CTG_GTSUP
Get supplemental information for a catalogue

Description:

Returns the supplemental information associated with a given entry in a CTG group.

Invocation:

```
CALL CTG_GTSUP( IGRP, I, FIELDS, STATUS )
```

Arguments:

IGRP = INTEGER (Given)

The CTG group as returned by CTG_ASSOC, *e*tc.. This should be the last group in a GRP owner-slave chain.

I = INTEGER (Given)

The index of the required entry.

FIELDS(5) = CHARACTER * (*) (Returned)

The supplemental information associated with the entry specified by I. Each element of the returned array contains the following:

- – FITS extension specification (*e.g.*`"{3}"`) if any
- – File type
- – Base file name
- – Directory path
- – Full catalogue specification

This information is obtained from a set of groups associated with the supplied group IGRP by means of a chain of GRP *owner-slave* relationships. If any of these groups do not exist, the correponding elements of the above array are returned blank. Note, Element 5, the full catalogue specification, is obtained directly from the supplied group IGRP.

STATUS = INTEGER (Given and Returned)

The global status.

CTG_PTSUP
Store suplemental information for an catalogue

Description:

Stores the supplied items of supplemental information for a given entry in a CTG group. The GRP groups needed to store this supplemental information are created if they do not already exist, and associated with the supplied group by means of a chain of GRP *owner-slave* relationships. They will be deleted automaticaly when the supplied group is deleted using GRP_DELET.

Invocation:

```
CALL CTG_PTSUP( IGRP, I, FIELDS, STATUS )
```

Arguments:

IGRP = INTEGER (Given)

The CTG group as returned by CTG_ASSOC, *etc.*. This should be the last group in a GRP owner-slave chain.

I = INTEGER (Given)

The index of the required entry.

FIELDS(5) = CHARACTER * (*) (Given)

The supplemental information to be stored with the entry specified by I. Each element of the supplied array should contain the following:

- – FITS extension (*e.g.*`"{3}"`) if any
- – File type
- – Base file name
- – Directory path
- – Full catalogue specification

STATUS = INTEGER (Given and Returned)

The global status.

CTG_SETSZ
Reduces the size of a CTG group

Description:

This routine should be used instead of GRP_SETSZ to set the size of a group created by CTG. It sets the size of the supplied group, and also sets the size of each of the supplemental groups associated with the supplied group.

Invocation:

```
CALL CTG_SETSZ( IGRP, SIZE, STATUS )
```

Arguments:

IGRP = INTEGER (Given)

The CTG group as returned by CTG_ASSOC, *etc.* This should be the last group in a GRP owner-slave chain.

SIZE = INTEGER (Given)

The new group size. Must be less than or equal to the size of the smallest group in the chain.

STATUS = INTEGER (Given and Returned)

The global status.

C Changes Introduced in CTG Version 3.0

- It has a separate identity. It was previously bundled into KAPLIBS.
- There is documentation.