SUN/181.12

Starlink Project
Starlink User Note 181.12

A.C. Davenhall & D.S. Berry

27th July 2020

CAT
Catalogue and Table Manipulation Library
Version 9.0
Programmer's Manual

Abstract

CAT is the Starlink Fortran subroutine library for manipulating astronomical catalogues and similar tabular datasets. This manual is intended for programmers who plan to write applications which use the CAT library to manipulate such datasets. It is introductory in the sense that it contains sufficient information to enable a programmer new to the CAT library to write applications which use it. The version of the library intended for use in conjunction with the Starlink ADAM programming environment is described. All the subroutines in the interface to the library are covered. However, the principles underlying the library, and their justification, are described only insofar as they affect its use.

Contents

List of Figures

Readership

This manual is intended for programmers who wish to use the Starlink CAT subroutine library to manipulate astronomical catalogues and similar tabular datasets. The reader is assumed to be familiar with the following: standard Fortran 77 and the extensions permitted by Starlink, Starlink programming practices and procedures and the various packages and libraries of the ADAM programming environment.

How to use this document

You will not usually need to read this document from beginning to end in order to use the CAT subroutine library. The document is divided into three parts:

Part I – Preliminaries,

Part II – Tutorial,

Part III – Reference.

Part I contains some preliminary details, such as an introduction to the terminology of CAT and details of how to access the library. Part II is a tutorial example of how to write a simple application which accesses the library and Part III contains reference material which describes the library.

If the CAT library is new to you, you should start by reading Part I and working through the tutorial in Part II. Finally, read Section 6 of Part III, but skipping the detailed descriptions of individual attributes.

If you are already familiar with the library you should consult the reference material in Part III, and probably also the details in Section 3 of Part I, as necessary.

Assistance and further information

I am happy to answer queries about using the CAT subroutine library and to receive comments or suggestions about how it could be improved. Details of how to contact me are included below.

Clive Davenhall.

Postal address: Institute for Astronomy, Royal Observatory, Blackford Hill, Edinburgh,
EH9 3HJ, United Kingdom.

Electronic mail: `acd@roe.ac.uk`

Fax:
from within the United Kingdom: 0131-668-8416

from overseas: +44-131-668-8416

Acknowledgments

The CAT subroutine library is far from being all my own work. Its specification evolved during extensive discussions with (alphabetically) Dave Giaretta, Clive Page, Rodney Warren-Smith and Alan Wood, all of whom have contributed substantially to its final form. Also, at various times, Brian Read, Malcolm Currie and Steven Beard have made useful comments.

Clive Page wrote the expression parser which CAT uses and Appendix B is based on documentation which he supplied. CAT accesses FITS tables through Bill Pence's FITSIO subroutine library and CHI/HDS catalogues through Alan Wood's CHI subroutine library.

Martin Bly assisted in preparing the Unix version for release. Alan Wood, Malcolm Currie and Peter Draper tested a pre-release version and suggested several useful improvements.

I am grateful to all these people for sharing their time and expertise.

Clive Davenhall

Department of Physics and Astronomy, University of Leicester
Saint Edmund's Day 1994

Revision history

(1) 23rd June 1994: Original draft (ACD).

(2) 9th October 1995: Version 1 (ACD).

(3) 12th April 1996: Version 2. Modified so that the Latex source could be used to create an HTML as well as a paper version (ACD).

(4) 23rd January 1997: Version 3. Modified for release 3.1 of the CAT library. The major change was the addition of support for the Small Text List (STL) catalogue format (ACD).

(5) 12th June 1997: Version 4. Modified for release 4.1 of the CAT library. There are no major enhancements in this release, just some bug fixes. Perhaps the most important of these is the ability to handle catalogue file names longer than fifteen characters (ACD).

(6) 9th November 1997: Version 5. Modified for release 4.2 of the CAT library. There are no major enhancements in this release, just a couple of bug fixes and some minor revisions to the document (ACD).

(7) 11th June 1998: Version 6. Modified for release 5.1 of the CAT library. There were major internal changes to the CAT library in order to speed it up and to reclaim internal workspace for reuse when a catalogue is closed. Most of these changes should not be visible to the user (except that the library should be somewhat faster). One visible change is that the capitalisation of the file type specified by the user when a new catalogue is opened is now preserved. One new routine has been added: `CAT_IINQ` (ACD).

(8) 26th November 1998: Version 7. Modified for release 6.1 of the CAT library. The major enhancement to the library in this release was additional features for reading sexagesimal angles from STL format catalogues. Rather than document these features here the entire description of the STL format was removed. The STL format is now documented solely in the CURSA manual, SUN/190. This rationalisation removes a duplication which was becoming increasingly cumbersome. Release 6.1 also includes some further improvements to the error reporting (ACD).

(9) 15th November 1999: Version 8. Modified for release 7.1 of the CAT library. The major change was the addition of support for the Tab-Separated Table (TST) catalogue format. All references to the VAX/VMS version of CAT were removed from the manual (ACD).

(10) 16th July 2000: Version 9. Modified for release 7.2 of the CAT library. There are no major enhancements in this release, just some bug fixes and minor changes, most notably to the implementation of the Tab-Separated Table (TST) format. There are also some minor revisions to the document (ACD).

(11) 4th April 2001: Version 10. Modified for release 8.1 of the CAT library. Support for the CHI/HDS catalogue format has been removed from this version of the library. There are also a few bug fixes and some minor revisions to the document (ACD).

(12) 17th January 2017: Modified for release 8.4 of the CAT library. Catalogues can now be stored in files with names that include non-alphanumeric characters. (DSB)

(13) 27th July 2020: Modified for release 9.0 of the CAT library. Null values are no longer supported for _LOGICAL columns. (DSB)

Part I
Preliminaries

1 Introduction

A harmless necessary cat.

The Merchant of Venice

CAT is the Starlink subroutine library for manipulating astronomical catalogues and similar tabular datasets. This manual provides you with the information necessary to write applications which use it. The manual describes version 9.0 of CAT. CAT is written in Fortran and is part of the Starlink ADAM programming environment. It conforms to all relevant Starlink standards and provides the facilities normally associated with ADAM libraries. A stand-alone version separate from ADAM is possible, but is not described here. Version 9.0 of CAT is available on all the variants of Unix supported by Starlink. The Fortran subroutine interface is identical in all these cases.

CAT provides basic facilities to:

- create,
- write,
- read

catalogues and tabular datasets. Unlike most similar Starlink libraries it supports catalogues held in various different formats. Currently the formats supported are:

- FITS tables (both ASCII and binary),
- STL (Small Text List; simple lists in text files),
- TST (Tab-Separated Table; tab-separated lists in text files).

There are idiosyncrasies and limitations associated with each of these formats and they are described in Appendix C.

2 Terminology

'When *I* use a word,' Humpty Dumpty said, in a rather scornful tone, 'it means just what I choose it to mean — neither more nor less.'

Through the Looking-Glass and What Alice Found There
Lewis Carroll

An astronomical **catalogue** is basically a **table** of values, consisting of the measurements of the same property for a set of objects, together with the auxiliary information necessary to describe this table. There are several different terminologies for describing the elements of such tables. In this manual a terminology which corresponds loosely to that used intuitively for the paper versions of astronomical catalogues is used:

row the values for all the properties associated with some particular object,

column the value of a single property for all the objects in a catalogue,

field the value of a single property for a single object (that is, the intersection of a row and a column).

Some of the other terminologies are shown for comparison in Table 1[1].

CAT	Fortran	Relational Database
table	file	relation
row	record	tuple
column	field	attribute
field	data item, field	component
format	format	schema
number of columns	number of fields	arity, degree
number of rows	number of records	cardinality

Table 1: Alternative terminologies for the components of tables

In the CAT library each **catalogue** can contain only one **table** and the two terms can usually be used interchangeably without introducing any ambiguity. However, where it is necessary to differentiate between the two sorts of entities, **table** is used to denote the simple matrix of rows and columns and **catalogue** is used to denote the combination of a table and its associated auxiliary information. Note that this usage implies nothing about the contents of the catalogue; it may contain a published astronomical catalogue, a set of private astronomical results or, indeed, data which are entirely non-astronomical.

In CAT the auxiliary information which applies to the entire catalogue comprises an arbitrary number of **parameters**. Each parameter comprises a single datum. Examples might be the epoch or equinox of celestial coordinates stored in a catalogue. CAT parameters are similar to FITS keywords (in fact, CAT interprets named keywords in a FITS table as parameters).

Columns and parameters both have a number of **attributes**, such as their name and data type. A full list of the attributes of columns and parameters is given in Sections 6.7 and 6.9 respectively, though normally you will only need to manipulate a few of them. All columns have the same set of attributes (though they take different values, of course) and similarly all parameters have the same set of attributes.

[1]This table is adapted from *Database Systems in Science and Engineering* by J.R. Rumble and F.J. Smith[7], p158.

Columns may either be **scalars** in which case each field comprises a single datum, or **vectors**, one-dimensional arrays where each field comprises a one-dimensional array of values. Parameters may only be scalars.

This section has introduced the terminology for the more important items in a CAT catalogue. Section 6 contains a more extensive discussion.

3 Getting started

CAT is an optional Starlink software item. Before proceeding you should check with your local site manager whether it is installed at your site, and if not attempt to persuade him to install it.

Version 9.0 of CAT comprises a subroutine library and two INCLUDE files. In addition, the CAT-EXAMPLES package provides five simple applications which are either examples of how to write CAT applications or simple utilities to examine catalogues (*cf* the `hdstrace` utility for examining HDS files). To write programs which use CAT you should have access to the library, the INCLUDE files and the CAT-EXAMPLES package. Some of the CAT-EXAMPLES applications serve as examples in Part II of this manual.

You do not need any special quotas or privileges to use CAT.

3.1 Accessing CAT-EXAMPLES

The following description applies to all the variants of Unix supported by Starlink. Simply type:

```
source /star/bin/examples/cat/cat-examples.csh
```

The following message should appear:

```
CAT example programs now available -- (for CAT Version 4.2-1)
```

If it does not, then the probable cause is that CAT-EXAMPLES is not installed correctly at your site; check with your local site manager.

3.2 INCLUDE files

The programming interface to the CAT library includes two INCLUDE files: `CAT_PAR` and `CAT_ERR`. These INCLUDE files define symbolic constants which your application programs may use:

`CAT_PAR` contains general constants pertaining to the CAT library,

`CAT_ERR` contains constants corresponding to the various error codes which can be set by the CAT library.

All the symbolic constants defined in both these files begin with the prefix CAT__ (in conformance with normal Starlink practice).

You can INCLUDE either or both of these files in a subroutine by including either or both of the following lines in the subroutine, as appropriate:

```
INCLUDE 'CAT_PAR'   ! CAT symbolic constants.
INCLUDE 'CAT_ERR'   ! CAT error codes.
```

If you are using a standard ADAM prologue for the subroutine these lines will go in the 'global constants' section.

The example applications included with CAT (see Section 4), and probably your own applications, also need to access the standard ADAM INCLUDE file SAE_PAR. For convenience the instructions given below include setting up access to this file.

The following description applies to all the variants of Unix supported by Starlink. The files are kept in directory /star/include with names cat_par and cat_err. You should set up soft links to these files, following the normal Starlink procedure, which is described in SUN/111.2[8]. See in particular Section 4.4, p6. Simply type:

```
star_dev
cat_dev
```

3.3 Linking

It might seem slightly perverse to describe how to link with the CAT library before describing how to write an application which calls it. However the rest of this manual describes various aspects of writing applications which call CAT and it seems sensible to get the somewhat separate question of linking out of the way first.

Your application should be written as an ADAM A-task. The Starlink shell script for linking an A-task, alink, will automatically compile the source code for the A-task. Type:

```
alink your_source_code `cat_link_adam`
```

This description applies to all the variants of Unix supported by Starlink.

3.4 Utility applications for examining catalogues

Once you have written an application which uses the CAT library to manipulate a catalogue you will often want to examine the contents of the catalogues which the application reads or writes. The catalogue browser xcatview in the Starlink package CURSA (see SUN/190[3]) is a convenient way to examine catalogues. It can access catalogues in any of the formats supported by CAT[2].

Catalogues written in the STL or TST formats (see Appendix C) are text files and can be examined with standard Unix commands such as more or cat.

[2]CURSA uses the CAT library to access catalogues.

In addition, CAT-EXAMPLES contains two simple utilities for examining the contents of catalogues. These utilities are largely of historical interest and are less easy to use than `xcatview`. They are mentioned here for completeness. The utilities are:

`details` reports the details of all the parameters and columns in a catalogue. All the attributes of every column and parameter are listed. The output is directed to the standard Fortran output stream.

`listout` lists selected columns from a catalogue to the terminal screen or a text file, or both. You are prompted for the columns which are to be displayed.

To invoke either of these applications simply type either `details` or `listout`, as appropriate, and answer the prompts, which are self-explanatory.

Part II
Tutorial

4 A simple example application

Example is always more efficacious than precept.

Lives of the English Poets vol. ii
Samuel Johnson

This section will walk you through the Fortran source code for a simple application which uses the CAT library to write a small catalogue. First, however, a few preliminaries.

- The programming interface to the CAT library comprises two components: the CAT subroutines and two INCLUDE files, `CAT_PAR` and `CAT_ERR`. These INCLUDE files define symbolic constants which application programs may use. Section 3.2 explains how to access these files. It is worth your while printing out and examining copies of both files. The comments included in the files should be sufficient to explain the purpose of each constant. `CAT_PAR` contains general constants pertaining to the CAT library. `CAT_ERR` contains constants corresponding to the various error codes which can be set by the CAT library. If your application needs to access one of these values you should *always* use the appropriate symbolic constant; *never* hard-code the actual value into your code. The values may (and probably will) change in subsequent releases of the CAT library.
- Within the CAT interface items in a catalogue, such as columns and parameters, and, indeed, the catalogue itself, are identified by an **identifier**. This approach is consistent with the treatment of data items in other ADAM libraries. Each identifier is an INTEGER number. The value of an identifier is unique (within a given invocation of an application) and is sufficient to identify the item to which it refers. You should observe the following simple rules when using identifiers in applications.
  - You should *never* set the value of a new identifier yourself; CAT will *always* generate an identifier for you.
  - You should *never* alter the value of an identifier once CAT has allocated it.
  - You never need to know the actual value of an identifier. I suppose that you can print them out, if you really want to (they are just numbers), but the information is of no use to you.
- Null values for fields are indicated by a separate flag indicating whether the associated datum is null or not. This flag is of type LOGICAL and is coded as follows:

  `.TRUE.` – the field is null; no datum is available,

  `.FALSE.` – the field is not null; a valid datum is available.

This scheme is adopted in both the routines to get a value from a catalogue and those to put a value to a field in a catalogue. It is adopted to avoid any possible ambiguity in interpreting null values.

When a null value is obtained from a catalogue the actual datum returned will be the ADAM 'bad' value for the appropriate data type (where one is available). This procedure facilitates passing values obtained from CAT into other ADAM subroutines. It means, however, that the null value returned through the CAT interface is not necessarily the same as the representation of the null stored in the catalogue. Existing catalogues, for example FITS tables from external sources, can come with a variety of values used to represent null values. The necessary checks and substitutions are performed automatically and invisibly within the catalogue-format specific parts of the CAT library. Your application will simply see the appropriate ADAM 'bad' value.

Not all catalogues support null values in all their columns. If a column does not support null values then the null value flag will always be returned set to `.FALSE.`

The treatment of null values is described in greater detail in Section 8.2. In particular, Section 8.2.5 prescribes how applications should handle null values.

We are now ready to examine a simple application which uses the CAT library. We will use one of the example applications released with the library, `EXAMPLE_WRITE`. This example creates and writes values to a small catalogue. I strongly recommend that you print out a copy of the source code and refer to it as you work through the example. The source code is available in file:

```
/star/share/cat/example_write.f
```

This example program is simpler than a real application. Starting at the beginning, the first thing to notice about the application is that, because it is an ADAM A-task, at its top level it is a subroutine, not a main program:

```
      SUBROUTINE EXAMPLE_WRITE (STATUS)
```

The status argument is used to keep track of the success of the application as it proceeds with its task.

```
*+
*  Name:
*     EXAMPLE_WRITE
      .
      .
      .
```

After the ADAM prologue comments the CAT symbolic constants are INCLUDEd. Some of the constants defined in this file will be used by the application.

```
*  Global Constants:
      INCLUDE 'SAE_PAR'
      INCLUDE 'CAT_PAR'
```

The various variables used in the application are defined next.

```
*  Status:
      INTEGER STATUS             ! Global status.
*  Local Variables:
      INTEGER
     :  CI,        ! Catalogue identifier.
     :  QII,       ! Identifier for a real parameter.
     :  QIR,       ! Identifier for a real parameter.
     :  QIC,       ! Identifier for a character parameter .
     :  FII,       ! Identifier for an integer column (or field).
     :  FIR,       ! Identifier for a real column (or field).
     :  FIC,       ! Identifier for a character column (or field).
     :  LOOP       ! Loop index.
      INTEGER
     :  VALI       ! Integer value.
      REAL
     :  VALR       ! Real value.
      CHARACTER
     :  VALC*10    ! Character value.
      LOGICAL
     :  NULI,      ! Null flag corresponding to VALI.
     :  NULR,      !  "    "         "       "  VALR.
     :  NULC       !  "    "         "       "  VALC.
*.
```

The first executable statement is the usual check that the status is ok. This check is mandatory in ADAM applications.

```
      IF (STATUS .EQ. SAI__OK) THEN
```

Once it is out of the way the application proper can start. It starts with a call to `CAT_CREAT`:

```
      CALL CAT_CREAT ('CNAME', CI, STATUS)
```

This subroutine will create a new catalogue, whose name it obtains from the ADAM parameter system (in practice the user will be prompted for it)[3]. The first argument of `CAT_CREAT` is the name of the ADAM parameter which will supply the catalogue name. The remaining two arguments are returned by `CAT_CREAT`; `CI` is the identifier for the catalogue. We will use this variable to refer to the catalogue throughout the application. `STATUS` is the running status argument which is usual in ADAM libraries. If the routine has succeeded its value is `SAI__OK`.

After `CAT_CREAT` has executed successfully a new catalogue has been created, but no columns or parameters have been defined for it, and it contains no data. The next step is to define some columns. The first column defined is an INTEGER column called `COLI`. Subroutine `CAT_PNEW0` is used for this task:

[3]It is possible to create or open a CAT catalogue without going through the ADAM parameter system by calling routine `CAT_TOPEN` rather than `CAT_CREAT`. This routine is not used in the present example, but is described in Section 7.5, below.

```
      CALL CAT_PNEW0 (CI, CAT__FITYP, 'COLI', CAT__TYPEI, FII,
     :   STATUS)
```

The first argument, CI, identifies the catalogue to which the new column belongs. The second argument tells CAT_PNEW0 that it has to create a new column; this argument should always be CAT__FITYP when a column is to be created. The third argument is the name of the column, COLI in the present case, and the fourth argument defines its data type. There are symbolic constants defined in CAT_PAR for each of the data types supported by CAT. COLI is an INTEGER column, so the code for an INTEGER is used. The codes for the different data types are listed in Table 4.

The fifth argument, FII, is returned rather than given and is an identifier for the column; subsequent references to the column will be via this identifier. The final argument is the usual ADAM running status.

The mandatory information which you must supply to define a column is: the catalogue to which it belongs, its name and its data type. All these values are specified using routine CAT_PNEW0. However, as explained in Section 2, a column is defined by a set of attributes, of which the name and data type are but two, albeit mandatory ones. Section 6.7 lists all the attributes for a column. If you do not specify the remaining attributes default values are adopted for them. You can, however, supply your own values to over-ride the defaults. The next line is an example of doing so.

```
      CALL CAT_TATTC (FII, 'COMM', 'Integer column', STATUS)
```

This routine is one of a family of similar routines, one for each data type (CAT_TATTC for attributes of data type CHARACTER, CAT_TATTI attributes of type INTEGER etc). Here it is used to set the comment attribute, COMM of column FII to the value 'Integer column'. Other attributes of column COLI could have been set in the same way, but in this example they are not, and the defaults are adopted[4].

Two additional columns are created in the same way; the REAL column COLR and the CHARACTER column COLC. Note that in the case of COLC the CHARACTER size attribute CSIZE is set using routine CAT_TATTI.

```
      CALL CAT_PNEW0 (CI, CAT__FITYP, 'COLR', CAT__TYPER, FIR,
     :   STATUS)
      CALL CAT_TATTC (FIR, 'COMM', 'Real column', STATUS)

      CALL CAT_PNEW0 (CI, CAT__FITYP, 'COLC', CAT__TYPEC, FIC,
     :   STATUS)
      CALL CAT_TATTC (FIC, 'COMM', 'Character column', STATUS)
      CALL CAT_TATTI (FIC, 'CSIZE', 10, STATUS)
```

An important restriction to remember is that columns must be created, and their attributes set, before any rows of data are written to the catalogue. Once a table of values have been written to

[4]For convenience two additional routines, which are not used in the present example, are provided for creating columns. CAT_CNEWS creates a column and simultaneously sets some of the more commonly used attributes; CAT_CNEWA creates a column and simultaneously sets all of its attributes. These routines may be more convenient to use than a call to CAT_PNEW0 followed by multiple calls to CAT_TATT<t>. They are described in Section 7.8.3, below.

a catalogue the details of its existing columns are frozen and no new columns can be created for it.

After creating the columns, some parameters are created. The first of these is the INTEGER parameter PARI. It is created with the same routine that was used to create the columns, CAT_PNEW0:

```
      CALL CAT_PNEW0 (CI, CAT__QITYP, 'PARI', CAT__TYPEI, QII,
     :   STATUS)
```

Again the first argument is the catalogue identifier. The second argument indicates that a parameter is to be created; this argument should always be set to CAT__QITYP when a parameter is to be created. The remaining arguments are exactly the same as they were for creating a column: PARI is the parameter name, CAT__TYPEI the data type and QII and STATUS are the identifier for the parameter and the running status, respectively.

As for columns, only the minimum, mandatory attributes for the parameter are set with CAT_PNEW0, and default values are adopted for the remaining attributes. Section 6.9 lists the attributes of a parameter. Also just like columns, the attributes of parameters can be set using the CAT_TATT<t> family of routines (where t = C for CHARACTER attributes, I for INTEGER etc). In the example CAT_TATTC is used to set the comments attribute, COMM of parameter PARI and CAT_TATTI is used to set the value attribute VALUE. The latter point is worth remembering; having created a parameter with CAT_PNEW0 it is necessary to use one of the CAT_TATT<t> routines to set its value, and the routine chosen should correspond to the data type of the parameter[5].

```
      CALL CAT_TATTC (QII, 'COMM', 'Integer parameter', STATUS)
      CALL CAT_TATTI (QII, 'VALUE', 23, STATUS)
```

Two additional parameters are created; the REAL parameter PARR and the CHARACTER parameter PARC. Note that, as was the case for columns, for the CHARACTER parameter PARC the CHARACTER size attribute 'CSIZE' must be set using CAT_TATTI.

```
      CALL CAT_PNEW0 (CI, CAT__QITYP, 'PARR', CAT__TYPER, QIR,
     :   STATUS)
      CALL CAT_TATTC (QIR, 'COMM', 'Real parameter', STATUS)
      CALL CAT_TATTR (QIR, 'VALUE', 42.0, STATUS)

      CALL CAT_PNEW0 (CI, CAT__QITYP, 'PARC', CAT__TYPEC, QIC,
     :   STATUS)
      CALL CAT_TATTC (QIC, 'COMM', 'Character parameter', STATUS)
      CALL CAT_TATTI (QIC, 'CSIZE', 20, STATUS)
      CALL CAT_TATTC (QIC, 'VALUE', 'Example string', STATUS)
```

Note that, unlike columns, parameters can be created at any stage while writing a catalogue. They do not have to be created prior to writing the table of values for the catalogue.

Once the parameters have been created the example starts a loop which will write the table of values.

[5]For convenience two additional routines, which are not used in the present example, are provided for creating parameters. CAT_PPTS<t> creates a parameter and simultaneously sets some of the more commonly used attributes; CAT_PPTA<t> creates a parameter and simultaneously sets all of its attributes. These routines may be more convenient to use than a call to CAT_PNEW0 followed by multiple calls to CAT_TATT<t>. They are described in Section 7.8.4, below.

```
      DO LOOP = 1, 25
```

In the CAT interface the basic method of writing (and reading) a catalogue is one row at a time. Each increment of the loop will correspond to a separate row of the catalogue, and thus in total twenty-five rows will be written, numbered from one to twenty-five.

The first few lines of code inside the loop are concerned with inventing values to write to the catalogue. The next few lines set the null value flags (all the rows contain non-null values, except row ten, where the flags are set to null). In a real application, of course, these values would not be invented, but would either be the results of a calculation or read from an external file.

```
         VALI = LOOP
         VALR = 2.3E1 + REAL(LOOP)
         VALC = ' '
         WRITE(VALC, 4000) LOOP
4000     FORMAT(' Loop ',I3, '%')

         NULI = .FALSE.
         NULR = .FALSE.
         NULC = .FALSE.

*
*       Make all the columns contain null values for row 10.

         IF (LOOP .EQ. 10) THEN
            NULI = .TRUE.
            NULR = .TRUE.
            NULC = .TRUE.
         END IF
```

The line:

```
         CALL CAT_PUT0I (FII, VALI, NULI, STATUS)
```

writes a single value to column COLI. FII is the column identifier, VALI and NULI the value and null value flag respectively. STATUS is, of course, the running status. CAT_PUT0I is one of the family of CAT_PUT0<t> routines, with one routine per data type. (I for INTEGER, C for CHARACTER etc). Thus there are corresponding calls to write the REAL and CHARACTER columns COLR and COLC:

```
         CALL CAT_PUT0R (FIR, VALR, NULR, STATUS)
         CALL CAT_PUT0C (FIC, VALC, NULC, STATUS)
```

CAT has the concept of the 'current row buffer', which is an internal copy of the row of the catalogue which it is working on currently. The CAT_PUT0<t> routines put fields to the current row buffer (and the corresponding CAT_GET0<t> routines read values from it).

```
         CALL CAT_RAPND (CI, STATUS)
```

Writes out the current row buffer for catalogue `CI` to the actual catalogue file, appending it to the end of the catalogue. The internal current row buffer is initialized ready to receive new values, and the internal count of the actual catalogue row number to which the current row buffer corresponds is incremented by one. That is, `CAT_RAPND` takes care of writing the current row buffer to the catalogue and readies CAT to receive values for a new current row buffer. When a catalogue is created the current row buffer is initialized automatically and the row number to which it corresponds is set to one.

Thus a loop with one increment corresponding to one row in the catalogue and containing calls to `CAT_PUT0<t>` routines to put values to the current row buffer and a call to `CAT_RAPND` to append the current row buffer to the catalogue is all that is necessary to write a table of values to a catalogue. The current row has been written, so the loop generating each row can now terminate:

```
      END DO
```

There is one final call to a CAT routine:

```
      CALL CAT_TRLSE (CI, STATUS)
```

This call 'releases' catalogue identifier `CI` and closes the corresponding catalogue. Identifiers to items within the catalogue, such as the various columns and parameters, are also released. Any remaining values are written to disk, the files are closed etc. The creation of the catalogue is complete. Finally, if all has succeeded, the application reports a message and then terminates.

```
      IF (STATUS .EQ. SAI__OK) THEN
         CALL MSG_OUT (' ', 'Catalogue created successfully.',
     :     STATUS)
      ELSE
         CALL ERR_REP ('EXAMPLE_WRITE_ERR', 'Failed to create '/
     :     /'catalogue.', STATUS)
      END IF

    END IF

    END
```

Having worked through this example application you might like to try it out. Type:

```
example_write
```

You will be prompted for the catalogue name. Reply, for example:

```
test.fit
```

After a couple of moments the message '`Catalogue created successfully.`' should appear and a binary FITS table called `test.fit` should have been created in your current directory. You can examine its contents using the `xcatview` catalogue browser in CURSA (see SUN/190[3]) or the `listout` utility in CAT-EXAMPLES. For the latter simply type:

```
listout
```

and answer the prompts (see Section 3.4 for details).

Another example program is available which reads back the catalogue created by `EXAMPLE_WRITE`. It is called `EXAMPLE_READ` and the source code is available in file:

```
/star/share/cat/example_read.f
```

You might like to print out a copy and try to understand it. If you have followed the discussion for `EXAMPLE_WRITE` you should be able to do so without any difficulty.

5 Carrying on

This section will briefly introduce some features of the CAT library which were not covered in the preceding example: celestial coordinates, vector columns, expressions, selections and indices. The description is introductory and informal; for a full description of these topics you should see the reference material in Part III.

5.1 Celestial coordinates in catalogues

Most astronomical catalogues contain columns of celestial coordinates of some sort: usually Right Ascension and Declination for some equinox and epoch, or perhaps Galactic or ecliptic coordinates. The storage, manipulation and presentation for display of celestial coordinates in the computer-readable version of astronomical catalogues is something of a vexed topic which has caused a deal of confusion and difficulty, much of it, in principle, unnecessary. Celestial coordinates are angles. The basic conundrum in storing and processing them is as follows:

- for use in computations inside a program the only sensible way to represent coordinates is as DOUBLE PRECISION or REAL numbers expressed in radians,
- for display to a user the coordinates should almost invariably be expressed in units of hours or degrees and formatted as a sexagesimal value, with subdivisions into minutes and seconds.

For preexisting catalogues the format of the celestial coordinates will already be fixed and they must simply be used in whatever way is possible. For example, many catalogues contain the hours, or degrees, minutes and seconds which comprise a coordinate as separate columns; a form which is singularly inconvenient for further processing. However, CAT has some special facilities for processing and displaying angles which conveniently and automatically provide for:

- representation inside a program as a DOUBLE PRECISION or REAL number expressed in radians,

- representation for display to a user as a CHARACTER string expressed in hours or degrees and formatted as a sexagesimal value.

This facility works as follows:

- when the value for a field in a column containing angles is obtained using CAT_EGTOD or CAT_EGTOR it is returned as a DOUBLE PRECISION or REAL value in radians,
- when the value for a field in a column containing angles is obtained formatted for display using CAT_EGTOF it is returned as a CHARACTER string containing the value expressed in hours or degrees and formatted as a sexagesimal value.

In order to use this facility CAT must know that the column contains an angle and the units (hours or degrees) and format to use when formatting the angle for display. The prescription for creating a column of angles in a CAT catalogue is as follows.

(1) Create the column with data type DOUBLE PRECISION.

(2) The UNITS attribute should start with the string RADIANS to indicate that the column contains an angle in radians and this should be followed by a specifier enclosed in curly brackets '{}' indicating how it is to be formatted for display. For example:

RADIANS{HOURS} if it is to be displayed in hours,

RADIANS{DEGREES} if it is to be displayed in degrees.

Specifying 'HOURS' will cause the angle to be displayed in hours, minutes and seconds, with the seconds displayed to one place of decimals; 'DEGREES' will cause the angle to be displayed in degrees, minutes and seconds, with the seconds displayed as whole numbers[6].

(3) When writing the table of values for the catalogue, express the angles in radians as DOUBLE PRECISION variables and write them to the catalogue using CAT_PUTOD.

An example program is available which illustrates the creation of columns containing angular celestial coordinates. It creates a catalogue containing (fake) equatorial coordinates and B magnitudes for a set of stars. It is called EXAMPLE_ANGLES and the source code is available in file:

/star/share/cat/example_angles.f

The overall structure is very similar to the previous example of writing a catalogue, EXAMPLE_WRITE, but note the way that the three points outlined above are followed in order to create the angular columns.

[6]This discussion covers only the simplest specifiers for formatting angles for display, though these are adequate for most cases and are the easiest to use. The full set of specifiers, which allow considerable flexibility in representing angles, are described in Section 8.3.

5.2 Vector columns

CAT does not provide facilities to simultaneously GET or PUT all the values for a given field in a vector column; rather the individual elements of each field must be GOT or PUT separately. Individual vector elements have their own identifiers. To process a vector element you should first get an identifier for the element and then GET or PUT values using the identifier just as you would for a scalar column.

The name of a vector element is the name of the vector column followed by the number of the element[7] enclosed in square brackets. Thus, `FLUX[4]` corresponds to the fourth element of vector `FLUX`. Identifiers for vector elements are obtained using `CAT_TIDNT`, just as for scalar columns.

The following example illustrates obtaining the values of the fourth element of vector `FLUX` for the first twenty-five rows of a catalogue. The value obtained for each row is read into REAL variable `FLXVAL`. In the example the value is overwritten by each successive row; in a real application it would be processed or stored in some way.

```
 INTEGER
:  CI,      ! Catalogue identifier.
:  VEI,     ! Vector element identifier.
:  ROW      ! Current row.
 REAL
:  FLXVAL   ! Value read for current row.
 LOGICAL
:  NULFLG   ! Null value flag.
   .
   .
   .
```

First get an identifier for the vector element.

```
CALL CAT_TIDNT (CI, 'FLUX[4]', VEI, STATUS)
```

Then loop through the first twenty-five rows getting the value for the vector element.

```
DO ROW = 1, 25
   CALL CAT_RGET (CI, ROW, STATUS)

   CALL CAT_EGTOR (VEI, FLXVAL, NULFLG, STATUS)
     .
     .
     .

END DO
```

[7] The first element of a vector is numbered one.

5.3 Expressions

In CAT an expression is an algebraic expression involving the names of columns and parameters and constants, linked by arithmetic operators and mathematical and astronomical functions. The syntax for expressions is described in Appendix B. The first step towards manipulating an expression is to obtain an identifier for it. Routine `CAT_EIDNT` is used for this purpose. If the expression is invalid (for example because it contains the name of a column which does not exist in the catalogue) this routine will return with an error status. Once you have an identifier for an expression the expression can be evaluated using the same routines that GET the values of a column (you are 'getting the value of the expression'). Clearly, values cannot be written to expressions and there is no equivalent of PUTting a value to a column.

Suppose that a catalogue contained columns called `x` and `y` and parameter `p`. The following example illustrates evaluating the expression '`x + y + p + 2.0`' (that is, getting its value) for the first twenty-five rows of the catalogue. The evaluated expression for each row is read into REAL variable[8] `EXPVAL`. In the example the value is overwritten for each successive row; in a real application it would be processed or stored in some way.

```
 INTEGER
:  CI,      ! Catalogue identifier.
:  EI,      ! Expression identifier.
:  ROW      ! Current row.
 REAL
:  EXPVAL   ! Value read for current row.
 LOGICAL
:  NULFLG   ! Null value flag.
   .
   .
   .
```

First get an identifier for the expression.

```
CALL CAT_EIDNT (CI, 'x + y + p + 2.0', EI, STATUS)
```

Then loop through the first twenty-five rows getting the value for the expression.

```
DO ROW = 1, 25
   CALL CAT_RGET (CI, ROW, STATUS)

   CALL CAT_EGTOR (EI, EXPVAL, NULFLG, STATUS)
     .
     .
     .

END DO
```

[8]Internally, inside CAT, numeric expressions are always evaluated using data type DOUBLE PRECISION. If you are unsure about the data type to use for the evaluated result of an expression the safest choice is to use DOUBLE PRECISION in order to ensure that accuracy is not lost.

Null values for expressions work just like null values for scalar columns. If a null value is generated CAT_EGT0<t> returns the appropriate Starlink 'bad' value rather than a genuine datum, and the null value flag (argument NULFLG in the example) is set to .TRUE. Expressions can evaluate to null in a number of ways: perhaps individual fields in the expression are themselves null, or an arithmetic exception (such as ÷ by zero) might occur in the expression.

5.4 Selections

In CAT a selection is a set of rows in a catalogue which satisfy some criteria. Every selection that you create has a unique identifier. Once you have obtained a selection identifier it can be passed to CAT_RGET instead of a catalogue identifier, and CAT_RGET will operate on just the rows in the selection, rather than all the rows in the catalogue.

There are two routines for creating selections (that is, generating selection identifiers): CAT_SELCT and CAT_SFND<t>. CAT_SELCT allows selections to be made on complex criteria whereas CAT_SFND<t> selects values in a simple range for a given sorted column. CAT_SFND<t> will usually be faster than CAT_SELCT (a useful mnemonic is F for Fast and Find, S for Slow and Select), but is much less flexible. Both these routines optionally allow you to create a second selection comprising the rejected rows, as well as the primary selection of selected rows.

Suppose a catalogue was sorted on column DEC of data type REAL and you wished to find the rows for which DEC was in the range 10.0 to 20.0. Because this is a simple range selection on a sorted column routine CAT_SFND<t> can be used.

```
 INTEGER
:  CI,      ! Catalogue identifier.
:  FI,      ! Identifier for column DEC.
:  SI,      ! Selection identifier for selected rows.
:  NUMSEL,  ! Number of selected rows.
:  SIR,     ! Selection identifier for rejected rows.
:  NUMREJ,  ! Number of rejected rows.
:  ROW      ! Current row.
 REAL
:  DECVAL   ! Value read for DEC from current row in selection.
 LOGICAL
:  NULFLG   ! Null value flag.
   .
   .
   .
```

First get an identifier for column DEC.

```
CALL CAT_TIDNT (CI, 'DEC', FI, STATUS)
```

Next create the selection and get an identifier for it. Note that because column DEC is of type REAL, the REAL version of CAT_SFND<t>, CAT_SFNDR, is used, and in this example the optional second selection, comprising the rejected rows, is not generated.

```
CALL CAT_SFNDR (CI, FI, 10.0, 20.0, .FALSE., SI, NUMSEL,
:  SIR, NUMREJ, STATUS)
```

Loop through all the rows in the selection, getting values for column DEC. Note how the selection identifier, SI, rather than the catalogue identifier, is passed to routine CAT_RGET.

```
DO ROW = 1, NUMSEL
   CALL CAT_RGET (SI, ROW, STATUS)

   CALL CAT_EGTOR (FI, DECVAL, NULFLG, STATUS)
     .
     .
     .

END DO
```

As a second example, suppose that the catalogue was not sorted on column DEC. CAT_SFND<t> could not now be used to create the selection, and CAT_SELCT would have to be used instead. The procedure would then be as follows.

```
 INTEGER
:  CI,      ! Catalogue identifier.
:  FI,      ! Identifier for column DEC.
:  EI,      ! Expression identifier.
:  SI,      ! Selection identifier for selected rows.
:  NUMSEL,  ! Number of selected rows.
:  SIR,     ! Selection identifier for rejected rows.
:  NUMREJ,  ! Number of rejected rows.
:  ROW      ! Current row.
 REAL
:  DECVAL   ! Value read for DEC from current row in selection.
 LOGICAL
:  NULFLG   ! Null value flag.
   .
   .
   .
```

First CAT_EIDNT is used to get an expression identifier for the expression representing the criteria. This identifier is then passed to CAT_SELCT. Remember that the criterion is that DEC should be in the range 10.0 to 20.0.

```
CAT_EIDNT (CI, '(DEC >= 10.0) & (DEC <= 20.0)', EI, STATUS)

CAT_SELCT (CI, EI, .FALSE., SI, NUMSEL, SIR, NUMREJ, STATUS)
```

Next get an identifier for column DEC, then loop through all the rows in the selection, getting values for DEC. Again note how the selection identifier, SI, rather than the catalogue identifier, is passed to routine CAT_RGET.

```
CAT_TIDNT (CI, 'DEC', FI, STATUS)

DO ROW = 1, NUMSEL
   CALL CAT_RGET (SI, ROW, STATUS)
```

```
      CALL CAT_EGTOR (FI, DECVAL, NULFLG, STATUS)
        .
        .
        .

    END DO
```

5.5 Indices

In CAT an index is a mechanism for accessing the rows of a catalogue in the order that they would have were the catalogue sorted into ascending or descending order on some (numeric) column. That is, an index presents the application with a 'virtual catalogue' which appears to have been sorted on the specified column. Every index that you create has a unique identifier. Once you have obtained an index identifier it can be passed to `CAT_RGET` instead of a catalogue identifier, and `CAT_RGET` will return subsequent rows corresponding to ascending or descending order of the column used to generate the index.

Indices are created using routine `CAT_INEW`. *Version 9.0 of CAT supports only temporary indices which persist only for the duration of the application which generated them and perish when it terminates.* Future versions of CAT will support permanent indices which persist after the program which generated them terminates.

Suppose that you wished to generate an index on column `DEC` of data type REAL[9] and then process the rows of the catalogue using this index.

```
 INTEGER
:  CI,       ! Catalogue identifier.
:  FI,       ! Identifier for column DEC.
:  II,       ! Identifier for index generated from column DEC.
:  NUMROW,   ! Number of rows in the catalogue.
:  ROW       ! Current row.
 REAL
:  DECVAL    ! Value read for DEC from current row in index.
 LOGICAL
:  NULFLG    ! Null value flag.
   .
   .
   .
```

First get an identifier for column DEC.

```
CALL CAT_TIDNT (CI, 'DEC', FI, STATUS)
```

Next create an index and get an identifier for it.

```
CALL CAT_INEW (FI, 'TEMP', CAT__ASCND, II, STATUS)
```

[9]Indices can be generated on columns of any of the numeric data types.

The second argument indicates that the index is temporary rather than permanent; 'TEMP' is the only permitted value here in version 9.0 of CAT. The third argument indicates whether the index is to correspond to ascending ('CAT__ASCND') or descending ('CAT__DSCND') order.

To access the rows via the index simply determine the number of rows in the catalogue and then loop through the rows, passing the index identifier, II, rather than then catalogue identifier, to routine CAT_RGET. The values obtained might be processed in some way, or written out to generate a sorted catalogue.

```
CALL CAT_TROWS (CI, NUMROW, STATUS)

DO ROW = 1, NUMROW
   CALL CAT_RGET (II, ROW, STATUS)

   CALL CAT_EGTOR (FI, DECVAL, NULFLG, STATUS)
     .
     .
     .

END DO
```

Part III
Reference

6 Components of a CAT catalogue

This section describes the components of a CAT catalogue. It is necessary to understand the structure of a CAT catalogue in order to use the CAT library effectively. An idealized computer-readable version of an astronomical catalogue, or similar tabular dataset, might comprise the following elements:

(1) the table of values which comprise the catalogue,

(2) a description of this table; the details of all the columns that it contains, the number of rows etc,

(3) textual information about the catalogue; perhaps a short description of the catalogue or a copy of a published paper describing it. This information is intended to be read by a human rather than interpreted by a computer.

The CAT library is mostly concerned with the first two items. However, it also provides some simple facilities to retrieve and write the textual information of the third item. These latter facilities are provided so that the textual information in a catalogue can be displayed to a user or copied when a new catalogue is created from an old one. The routines for manipulating textual information are described in Section 7.9. They do not interact with any other items in a CAT catalogue and they are not mentioned again in this section.

The table in a CAT catalogue is very similar to a **relation** in the theory of relational databases, and has many of the same properties. Each row in the table must contain the same number of fields. Corresponding fields in different rows must be of the same type. The table may contain an arbitrary number of rows. In the formal theory of relational databases, no two rows may be identical. CAT relaxes this rule by permitting identical rows, though it is difficult to see what purpose such rows might serve.

The internal organization of a CAT catalogue (the way it is formatted on disk) is unknown to an application using the CAT library. The values in the catalogue are accessed purely through the subroutine interface to the CAT library.

6.1 Provision for future enhancements

This manual describes version 9.0 of the CAT library. The original specification for the library is described in the document *The Starlink Subroutine Interface for Manipulating Catalogues* (StarBase/ACD/3.4)[2]. Version 9.0 of CAT is a subset of this full implementation and some of the items present in it serve no apparent purpose. These items correspond to features which were in the original specification but which are not currently implemented. These items may be implemented in future versions and have been included so that applications written now will be compatible with future versions of the library.

6.2 Symbolic constants

Various symbolic constants are referred to throughout this section. These constants are defined in INCLUDE file `CAT_PAR`, which may be INCLUDEd in the subroutines of an application. See Section 3.2 for details of how to access this file.

6.3 Catalogues, components and attributes

In the CAT model of a catalogue a catalogue comprises a number of **components**. In version 9.0 of CAT a catalogue may contain two sorts of components: **columns** and **parameters**:

columns define the individual columns (scalars or vectors) in the table,

parameters provide single items of information which apply to the entire catalogue. Examples might be the epoch or equinox of celestial coordinates in the catalogue.

A catalogue may contain an arbitrary number of columns and an arbitrary number of parameters. Columns and parameters are permanent entities which persist in between invocations of applications accessing the catalogue through the CAT library (typically as items in a disk file). In addition to these permanent components additional sorts of temporary components may be created by CAT: **expressions**, **selections** and **indices**.

expressions define a quantity computed from existing columns (and parameters) using some algebraic or logical (boolean) expression,

selections define a set of rows selected from the catalogue according to some criteria.

indices define an order for accessing rows in the catalogue equivalent to sorting the catalogue on a specified column.

Unlike columns and parameters, expressions, selections and indices are ephemeral entities[10] which perish when the application using CAT which created them terminates.

Every component consists of a number of **attributes**. Each type of component (permanent or temporary; column, parameter, expression, selection or index) has a fixed set of attributes, each identified by name. The values of the attributes differ between components, and their totality defines the component. Additionally there are two special attributes which apply to the entire catalogue, rather than to a particular component. This hierarchy is illustrated in Figure 1. Subsequent sections describe these catalogue attributes and the attributes of columns, parameters, expressions, selections and indices.

6.4 Identifiers

Catalogues, columns, parameters, expressions, selections and indices are all identified by an **identifier**. Each identifier is an INTEGER number. The value of an identifier is unique (within a given invocation of an application) and is sufficient to identify the item to which it refers. The following rules apply when using CAT identifiers in applications:

[10] Future versions of CAT will support permanent indices.

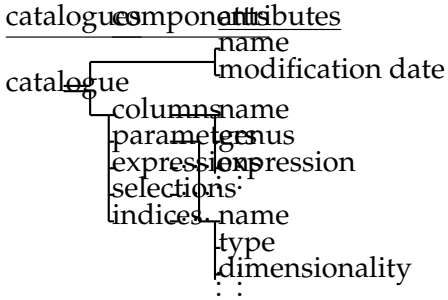

Figure 1: The hierarchy of catalogues, components and attributes

- an application should *never* set the value of a new identifier itself; the CAT library will *always* generate a new identifier,
- an application should *never* modify the value of an existing identifier once CAT has allocated it,
- an application never needs to know the actual value of an identifier.

An application can inquire what sort of item (catalogue, column, parameter etc.) an identifier represents using subroutine `CAT_TIDTP`. The various types of identifiers are represented using INTEGER codes, and symbolic constants are defined for these codes. They are listed in Table 2.

Type of identifier	CAT symbolic constant
Catalogue	`CAT__CITYP`
Column or field	`CAT__FITYP`
Vector column element	`CAT__FETYP`
Parameter	`CAT__QITYP`
Expression	`CAT__EITYP`
Selection	`CAT__SITYP`
Index	`CAT__IITYP`
Null identifier	`CAT__NOID`

Table 2: The types of identifiers

The catalogue to which a component (column, parameter etc.) belongs is referred to as the **parent** of that component. Subroutine `CAT_TIDPR` can be used to inquire the parent of an identifier. In CAT version 9.0 catalogues do not have parents. If `CAT_TIDPR` is used to try to find the parent of a catalogue then the null identifier is returned.

6.4.1 The null identifier

When CAT is asked to generate an identifier for an item which does not exist (such as the parent of a catalogue) it will return the 'null identifier'. The meaning of this identifier is that the specified component does not exist. The symbolic constant for the null identifier is `CAT__NOID`.

6.5 Attributes

Attributes do not have their own identifiers. An attribute is specified by the identifier of the component of which it is a part and its name. This combination is unique for a given attribute. For example the 'data type' attribute of a column (see Section 6.7) is specified by the identifier of the column and the name of the attribute ('DTYPE' in this case). Each attribute has a data type associated with it. Families of subroutines (one per data type) are available to set and inquire the values of attributes:

`CAT_TATT<t>` – set an attribute,

`CAT_TIQA<t>` – inquire the value of an attribute.

See Section 7.8.1 for details of using these subroutines.

All the attributes for a given component adopt values when the component is created. Some attributes are mandatory, in which case values must be supplied for them. For the remaining attributes values are optional, and if they are not supplied defaults are adopted.

Most attributes are immutable; they are specified once when the component is created and are fixed thereafter. A few, however, are mutable and may be changed at any stage during the life of the component. The immutable attributes of all the columns in a catalogue are frozen when values are first written to the table[11].

6.6 Catalogue attributes

In addition to its collection of column and parameter components, all with their individual attributes, a catalogue also has several attributes which apply directly to the entire catalogue, rather than to an individual component (see Figure 1 in Section 6.3). These attributes are described below.

NAME (data type: _CHAR, size = `CAT__SZCNM`) The name of the catalogue. It is specified when the catalogue is created and is mandatory and immutable. The NAME attribute is related to the file name of the catalogue as follows. It is the same as the file name, but without any preceding directory specification or trailing file type. Thus, if `CATNAME` is the NAME attribute then the corresponding file name is:

```
directory_specification/CATNAME.file_type
```

The file type corresponds to the format of the catalogue (FITS table, Small Text List etc). The various options are described in Appendix C.

[11] This description extends the discussion in Section 4, which for simplicity omitted to mention mutable attributes.

DATE (data type: _DOUBLE) In version 9.0 of CAT the modification date is present, but not used. It is set to 0.0D0 when the column is created.

BACK (data type: _INTEGER) The back-end type of the catalogue. It will be one of the CAT__BKFIT, CAT__BKSTL or CAT__BKTST. These symbolic constants are defined in include file CAT_PAR.

In version 9.0 of CAT the modification date is present, but not used. It is set to 0.0D0 when the column is created.

PATH (data type: _CHAR, size = `CAT__SZCNF`) The full path of the catalogue file. It is specified when the catalogue is created and is mandatory and immutable.

6.7 Columns

A column may contain either a single value for each row (as in standard relational database theory) or a one-dimensional array of values for each row. An array must be of fixed size, defined when the column is created. There is no upper limit to the number of elements which an array may contain. A single-valued column is called a **scalar** and a column containing an array is called a **vector**. The attributes of a column are listed in Table 3 and described below.

The attributes of an individual element of a vector column are somewhat different and are described in Section 6.8, below.

NAME (data type: _CHAR, size = `CAT__SZCMP`) The name of the column. The rules for column names are as follows.

- The name must be unique within the totality of parameters and columns for the catalogue. This condition is necessary in order that a component (parameter or column) may be identified unambiguously when its name is used in an expression.
- A name may comprise up to fifteen characters (`CAT__SZCMP`). This value is chosen for consistency with HDS and is adequate for FITS tables.
- The name can contain only: upper or lower case alphabetic characters (a-z, A-Z), numeric characters (0-9) and the underscore character ('_'). Note that lower case alphabetic characters must be allowed in order to access existing FITS tables. *However, corresponding upper and lower case characters are considered to be equivalent.* Thus, for example, the names: `HD_NUMBER`, `HD_Number` and `hd_number` would all refer to the same column.
- The first character must be a letter.

GENUS (Data type: _INTEGER) In version 9.0 of CAT the genus attribute is present, but not used. It is set to `CAT__GPHYS` when the column is created.

EXPR (Data type: _CHAR, size = `CAT__SZEXS`) In version 9.0 of CAT the expression attribute is present, but not used. It is set to blank (' ') when the column is created.

Attribute	Name	Data type	Mut--able	Mand--atory	Default
Name	NAME	_CHAR		•	
Genus	GENUS	_INTEGER			physical: `CAT__GPHYS`
Expression	EXPR	_CHAR			‘ ’
Data type	DTYPE	_INTEGER		•	
Character size	CSIZE	_INTEGER			20†
Dimensionality	DIMS	_INTEGER			scalar: `CAT__SCALR`
Size§	SIZE	_INTEGER			1
Null or locum	NULL	_INTEGER			HDS: `CAT__NULLD`
Exception values	EXCEPT	_CHAR			‘ ’
Scale factor	SCALEF	_DOUBLE			1.0D0
Zero point	ZEROP	_DOUBLE			0.0D0
Order	ORDER	_INTEGER			none: `CAT__NOORD`
Units	UNITS	_CHAR	•		‘ ’
External format	EXFMT	_CHAR	•		varies with data type
Preferential display	PRFDSP	_LOGICAL	•		true
Comments	COMM	_CHAR	•		‘ ’
Modification date	DATE	_DOUBLE	•		0.0D0

† The size of character strings; other data types have CSIZE = 0.

§ SIZE is a single-element array, not a scalar.

Table 3: Attributes of columns

HDS Type	DEC Fortran Type	CAT symbolic constant	Description	Standard Fortran 77?
_BYTE	BYTE	`CAT__TYPEB`	Signed byte	No
_WORD	INTEGER∗2	`CAT__TYPEW`	Signed word	No
_INTEGER	INTEGER	`CAT__TYPEI`	Signed integer	Yes
_REAL	REAL	`CAT__TYPER`	Single precision	Yes
_DOUBLE	DOUBLE PRECISION	`CAT__TYPED`	Double precision	Yes
_LOGICAL	LOGICAL	`CAT__TYPEL`	Logical	Yes
_CHAR[∗n]	CHARACTER[∗n]	`CAT__TYPEC`	Character string	Yes

n is the number of elements in the character string; it is a positive integer. In a CAT CHARACTER column the size of the string is stored in attribute CSIZE.

_BYTE and _WORD correspond exactly to the DEC Fortran data types BYTE and INTEGER*2 respectively; equivalent types exist in most other implementations of Fortran. The non-standard data types typically are required to accommodate raw data generated by instruments. The ranges of the primitive numeric types will be defined by the particular implementation of Fortran on the computer being used (this table is adapted from SUN/92[11]. See in particular the table in Section 2.2, p3).

Table 4: Permitted data types (adapted from SUN/92)

DTYPE (Data type: _INTEGER) The data type of values held in the column. The types permitted are listed in Table 4. They are deliberately the same as the types permitted in HDS and include the standard types of Fortran 77.

CSIZE (Data type: _INTEGER) For a CHARACTER column, the size of the column, otherwise not used and by convention set to zero.

DIMS (Data type: _INTEGER) The dimensionality of the column; a flag indicating whether it is a scalar or a vector. For a scalar column it is set to `CAT__SCALR` and for a vector to `CAT__VECTR`.

SIZE (Data type: _INTEGER; a single element array[12]) If the column is a vector this attribute contains the number of elements in the vector. If the column is a scalar it is set to one.

NULL (Data type: _INTEGER) A flag indicating whether or not null values are recognized in the column. Three cases are recognized:

- null values are present and are represented using the standard HDS null values (code: `CAT__NULLD`),
- null values are present and are represented using a value specified when the column was created (code: `CAT__NULLS`),
- null values are not present in the column (code: `CAT__LOCUM`).

The treatment of null values is discussed in Section 8.2, below.

EXCEPT (Data type: _CHAR, size = `CAT__SZVAL`) The value used to represent the null value, or the locum value generated if nulls are not supported in the column. See Section 8.2 for a full description.

SCALEF (Data type: _DOUBLE) The scale factor used to calculate the actual value of a scaled column from the scaled value stored. The actual value is computed according to the formula

$$\text{actual value} = (\text{SCALEF} \times \text{stored value}) + \text{ZEROP} \tag{1}$$

ZEROP (Data type: _DOUBLE) The zero point used to calculate the actual value of a scaled column from the scaled value stored. See above for the formula used.

ORDER (Data type: _INTEGER) The order in which individual fields in the column are stored. The three possibilities, together with the corresponding symbolic constants, are listed in Table 5.

[12] An array is used instead of a scalar to allow the possibility of introducing multi-dimensional arrays in a future version of CAT.

Column order	CAT symbolic constants
ascending	CAT__ASCND
descending	CAT__DSCND
unordered	CAT__NOORD

Table 5: Alternatives for the ordering of columns

UNITS (Data type: _CHAR, size = CAT__SZUNI) The units in which values stored in the column are expressed. The UNITS attribute is used to identify, and control the appearance of, columns of angles (see Section 8.3). Apart from this exception the units are treated purely as comments and no attempts are made to automatically propagate and convert units in calculations and selections. Case sensitivity is irrelevant for units since they are treated purely as comments. The units attribute can be left completely blank; a blank units attribute implies that the units are unknown. If it were desired to distinguish a dimensionless quantity from one with unknown units, the string 'DIMENSIONLESS' could be put in the units attribute[13].

EXFMT (Data type: _CHAR, size = CAT__SZEXF) The format used to represent a datum extracted from a column for external display on a screen or in a text file. These formats are used solely for external display, not internal conversion. The external format specifier should be a valid Fortran 77 format specifier for the data type of the column.

PRFDSP (Data type: _LOGICAL) The preferential display flag; a logical flag which indicates to reporting applications whether, *by default*, the column is to be displayed or not. It is coded as follows:

.TRUE. – display the column by default,

.FALSE. – do not display the column by default.

COMM (Data type: _CHAR, size = CAT__SZCOM) Explanatory comments describing the column. The comments may be up to eighty characters long (CAT__SZCOM).

DATE (Data type: _DOUBLE) In version 9.0 of CAT the modification date is present, but not used. It is set to 0.0D0 when the column is created.

6.8 Vector column elements

CAT treats vectors in quite a simple fashion. Values can only be GOT or PUT for individual vector elements; there are no routines for processing entire vectors. In order to access individual elements it is necessary to assign identifiers to them. Identifiers for vector elements, like those for scalar columns and parameters, are obtained using CAT_TIDNT. The name of a vector column element passed to CAT_TIDNT has the same syntax as the NAME attribute of the element, as

Attribute	Name	Data type
Name	NAME	_CHAR
Data type	DTYPE	_INTEGER
Character size	CSIZE	_INTEGER
Base identifier	BASEID	_INTEGER
Vector element	ELEM	_INTEGER

Table 6: Attributes of a vector column element

described below. The attributes of a vector column element identifier are different to the identifiers for the whole column; they are listed in Table 6.

All these attributes are created automatically when an identifier is obtained for the element; they are all mandatory and immutable.

The vector column to which a vector column element belongs is referred to as the column element's **base column**. The DTYPE and CSIZE attributes of a vector column element are necessarily identical to the corresponding attributes for its base column. The details of the remaining attributes are as follows.

NAME (data type: _CHAR, size = `CAT__SZCMP`) The name of the vector column element. That is, the name of the base column, followed by the number of the vector element, enclosed in square brackets. The number of the first element is one. Thus, the name of the fourth element of vector column `FLUX` would be `FLUX[4]`.

BASEID (data type: _INTEGER) The identifier of the base column of the vector element.

ELEM (data type: _INTEGER) The sequence number of the element in the column vector. The first element is numbered one. Thus, for example, if the name of the vector column element was `FLUX[4]` the value of the ELEM attribute would be four.

6.9 Parameters

Parameters are items of information which apply to the entire catalogue. Examples are the equinox or epoch of the celestial coordinates in the catalogue. The attributes of a parameter are given in Table 7. In CAT version 9.0 parameters must be scalars. However, they have the attributes dimensionality and size to allow for the possibility of vector parameters in future versions of CAT.

All these attributes, except **VALUE**, are deliberately identical to the corresponding attributes for columns (see Table 3 and Section 6.7, above, for details).

[13]Magnitudes, which properly are dimensionless, can, of course, have units of 'MAGNITUDES' or 'MAG' or whatever, if so desired.

Attribute	Name	Data type	Mand--atory	Default
Name	NAME	_CHAR	•	
Data type	DTYPE	_INTEGER	•	
Character size	CSIZE	_INTEGER		`CAT__SZVAL`†
Dimensionality	DIMS	_INTEGER		scalar: `CAT__SCALR`
Size§	SIZE	_INTEGER		1
Units	UNITS	_CHAR		‘ ’
External format	EXFMT	_CHAR		varies with data type
Preferential display	PRFDSP	_LOGICAL		true
Comments	COMM	_CHAR		‘ ’
Value	VALUE	varies		zero or ‘ ’
Modification date	DATE	_DOUBLE		0.0D0

† The size of character strings; other data types have CSIZE = 0.

§ SIZE is a single-element array, not a scalar.

Table 7: Attributes of parameters

VALUE (Data type: variable, corresponds to attribute DTYPE) The value of the parameter; it is mutable.

6.10 Expressions

Expressions define a quantity computed from the existing scalar columns, vector column elements and parameters of a catalogue using some algebraic or logical (boolean) expression. An expression adopts a value for every row in the catalogue. It is similar to a column, except that its value is computed 'on the fly' from existing columns (and parameters), rather than being stored in the catalogue. Usually an expression will evaluate to a numeric value, but it may equally well evaluate to a LOGICAL or CHARACTER value. For example, if a catalogue contained columns `x` and `y` an expression might be '`x + y`'. The syntax for specifying expressions is described in Appendix B.

An expression has a set of attributes which are identical to those for a scalar column (see Section 6.7 and Table 3), but with the following exceptions.

- The GENUS attribute is always set to `CAT__GVIRT`.
- The EXPR attribute is set to the algebraic expression used to compute the value of the expression.
- The DIMS attribute is always `CAT__SCALR`; expressions are always scalars and vector expressions are forbidden.

6.11 Selections

Selections define a set of rows selected from a catalogue according to some criteria. For example, if a catalogue contained column `x` then the selection criterion might be '`x > 10.0`', that is, the selection would comprise the set of rows for which the field of column `x` was greater than 10.0. The syntax for specifying expressions is described in Appendix B.

The attributes of a selection are listed in Table 8 and described below. With the exception of the comments attribute, COMM, they are all mandatory and immutable and are set automatically when the selection is created.

Attribute	Name	Data type
Expression	EXPR	_CHAR
Number of rows	NUMSEL	_INTEGER
Comments	COMM	_CHAR
Modification date	DATE	_DOUBLE

Table 8: Attributes of a selection

EXPR (Data type: _CHAR, size = `CAT__SZEXS`) The expression which rows in the catalogue must satisfy in order to be included in the selection.

NUMSEL (Data type: _INTEGER) The number of rows in the selection.

COMM (Data type: _CHAR, size = `CAT__SZCOM`) Explanatory comments describing the selection.

DATE (Data type: _DOUBLE) In version 9.0 of CAT the modification date is present, but not used. It is set to 0.0D0 when the selection is created.

6.12 Indices

Indices are a mechanism for accessing the rows of a catalogue as though they were sorted into ascending or descending order on some column. For example, if an ascending index was created on REAL column `DEC` and the rows of the catalogue were accessed through this index the rows would appear in ascending order of `DEC`. *In CAT version 9.0 indices are temporary entities which persist only for the duration of the application which generated them and perish when it terminates.* Future versions of CAT will support permanent indices which persist after the application which generated them terminates.

Indices can be created from columns of any of the numeric data types. They should not be created from columns of data type CHARACTER or LOGICAL. If an index is created on a column which contains null values then the rows for which the column is null will appear after all the rows with a valid value. The order of such rows is unpredictable.

The attributes of an index are listed in Table 9 and described below. They are all mandatory and immutable and are set automatically when the index is created.

Attribute	Name	Data type
Column identifier	COLID	_INTEGER
Order	ORDER	_INTEGER
Number of rows	NUMSEL	_INTEGER
Comments	COMM	_CHAR
Modification date	DATE	_DOUBLE

Table 9: Attributes of an index

COLID (Data type: _INTEGER) The identifier of the column from which the index was created.

ORDER (Data type: _INTEGER) The order of the index. The possibilities are:

`CAT__ASCND` – ascending,
`CAT__DSCND` – descending.

NUMSEL (Data type: _INTEGER) The number of rows in the index. In CAT version 9.0 the number of rows in the index is necessarily the number of rows in the catalogue. The attribute is present in order to allow indices to be created from selections in future versions of CAT.

COMM (Data type: _CHAR, size = CAT__SZCOM) Explanatory comments describing the index.

DATE (Data type: _DOUBLE) In version 9.0 of CAT the modification date is present, but not used. It is set to 0.0D0 when the selection is created.

7 Subroutine interface

This section introduces all the subroutines in the CAT library, with the subroutines arranged by function. For each subroutine the calling sequence and a brief description is given. Standard names are used for the calling arguments. A complete description of each subroutine is given in Appendix A. Following normal Starlink practice the subroutine arguments are arranged in the order:

given,
given and returned,
returned,
global (or running) status.

A semi-colon (';') is used to separate these various categories in the calling sequences given in this section.

7.1 Provision for future enhancements

This manual describes version 9.0 of the CAT library. The original specification for the library is described in the document *The Starlink Subroutine Interface for Manipulating Catalogues* (StarBase/ACD/3.4)[2]. Version 9.0 of CAT is a subset of this full implementation and some of the items present in it serve no apparent purpose. These items correspond to features which were in the original specification but which are not currently implemented. These items may be implemented in future versions and have been included so that applications written now will be compatible with future versions of the library.

7.2 Symbolic constants

Two INCLUDE files of symbolic constants are available for use with these subroutines: CAT_PAR and CAT_ERR. CAT_PAR contains symbolic constants which specify the size of various items, codes corresponding to various types of items, values for various flags etc. CAT_ERR contains symbolic constants corresponding to the various error codes which the CAT library can set. Section 3.2 explains how to access these files. I recommend that you print out copies of these files and have them to hand when writing applications. The comments included in the files should be sufficient to explain the purpose of each constant. If your application needs to access one of these values you should *always* use the appropriate symbolic constant; *never* hard-code the actual value into your code. The values may (and probably will) change in subsequent releases of the CAT library.

7.3 Subroutine names

The CAT subroutine names follow the normal Starlink format of a prefix, an underscore ('_') and a five-character routine name (see SGP/16[9]).

For most routines, the first letter of the main body of the subroutine name denotes the sort of item that the subroutine operates on, according to the following scheme:

T - total, or whole catalogue,

P - part (column or parameter),

C - columns (entire columns),

R - rows,

Q - parameters,

E - expressions (algebraic expressions),

S - selections,

The ADAM routines (see Section 7.6) are exceptions; they have names chosen to be consistent with the corresponding routines in other ADAM libraries.

7.4 Identifiers

CAT identifies catalogues and components by various sorts of **identifiers**; INTEGER numbers, each with a unique value. In the calling arguments for the subroutines the following names are usually used for the various sorts of identifier:

GI - generic,

CI - catalogue,

PI - part (column or parameter),

FI - column (F for field),

QI - parameter,

EI - expression,

SI - selection

The following rules apply:

(1) A generic identifier may be substituted by *any* type of identifier,

(2) there are circumstances where it is sensible to consider columns and parameters to be a single sort of component; in these circumstances, components and parameters are referred to collectively as *parts;* a part identifier may be substituted by either a column or a parameter identifier.

(3) an expression identifier may be substituted by a field, parameter (or hence part) identifier.

7.5 Initialization, opening and closing catalogues

Every open catalogue is identified to CAT by a catalogue identifier; an INTEGER number with a unique value. The catalogue identifier is subroutine argument `CI` in the following notes. The ADAM routines (see Section 7.6, below) provide an alternative to `CAT_TOPEN` for opening or creating a catalogue.

`CAT_TOPEN (CNAME, STATE, MODE; CI; STATUS)`
Open a catalogue and obtain an identifier to it.
STATE = one of: NEW or OLD,
MODE = one of: WRITE or READ.

`CAT_TRLSE (CI; STATUS)`
Release a catalogue identifier.

`CAT_RSET (CI, ROWS, STATUS)`
Set the number of rows which a new catalogue is expected to contain. This routine is optional. If used, then it is ignored by some catalogue formats. With others, such as the Small Text List, then the catalogue may be created more efficiently.

7.6 ADAM subroutines

These subroutines interact with the ADAM parameter system. They provide alternatives to `CAT_TOPEN` (see Section 7.5, above) for opening a catalogue. They are provided for compatibility with other ADAM libraries and each provides analogous functionality to equivalently named routines in other ADAM libraries.

`CAT_ASSOC (PCNAME, MODE; CI; STATUS)`
Open an existing catalogue; the name of the catalogue is obtained from an ADAM parameter. Argument `MODE` must always be `'READ'` in version 9.0 of CAT.

`CAT_CREAT (PCNAME; CI; STATUS)`
Create a new catalogue; the name of the catalogue is obtained from an ADAM parameter.

`CAT_EXIST (PCNAME, MODE; CI; STATUS)`
Attempt to open a catalogue, the name being taken from the ADAM parameter system. If the attempt fails then instead of re-prompting the subroutine returns with an error status. This subroutine can be used to check the existence of a catalogue.

7.7 Catalogue inquiry routines

These routines perform general 'high-level' inquiries on a catalogue. `CAT_TCOLS` is not strictly necessary; its function is provided by `CAT_TDETL`. However, it is convenient in practice.

`CAT_TDETL (CI, COLFLG; NUMROW, NUMCOL, NUMIND, NUMPAR, DATE; STATUS)`
Get the details of a catalogue. In CAT version 9.0 argument `COLFLG` should be set to `CAT__GPHYS`.

`CAT_TROWS (CI; NUMROW; STATUS)`
Get the number of rows in a catalogue, selection or index. This routine may be given a catalogue, selection or index identifier. It will return the number of rows in the catalogue, selection or index, as appropriate.

`CAT_TCOLS (CI, COLFLG; NUMCOL; STATUS)`
Get the number of columns in a catalogue. In CAT version 9.0 argument `COLFLG` should be set to `CAT__GPHYS`.

`CAT_TIDTP (GI; IDTYP; STATUS)`
Determine the type of an identifier. The codes for the various types of identifiers are shown in Table 2.

7.8 Component manipulation

`CAT_TIDNT (CI, GNAME; GI; STATUS)`
Get an identifier for a named pre-existing component.

`CAT_TNDNT (CI, IDTYP, N; GI; STATUS)`
Get an identifier for the *n*th (pre-existing) component of a given type. Note that if component *n* could not be found then the routine returns an ok status, but `GI` is set to null identifier `CAT__NOID`.

`CAT_TIDPR (GI; CI; STATUS)`
Determine the parent of a component.

7.8.1 Manipulating attributes of a component

`CAT_TATT<t> (GI, ATTRIB, VALUE; STATUS)`
Set an attribute of a component to a given value. Type conversions are performed if necessary.

`CAT_TIQA<t> (GI, ATTRIB; VALUE; STATUS)`
Inquire the value of a single attribute for a component.

7.8.2 Parts: columns and parameters

`CAT_PNEWO (CI, PTYPE, PNAME, DTYPE; PI; STATUS)`
Create a scalar part (column or parameter).

7.8.3 Columns

`CAT_CNEWA (CI, FNAME, EXPR, DTYPE, CSIZE, DIMS, SIZEA, NULL, EXCEPT, SCALEF, ZEROP, ORDER, UNITS, EXTFMT, PRFDSP, COMM; FI; STATUS)`
Create a column, simultaneously setting all its attributes.

`CAT_CNEWS (CI, FNAME, DTYPE, CSIZE, UNITS, EXTFMT, COMM; FI; STATUS)`
Create a column, simultaneously setting some of the more frequently used attributes. These attributes deliberately correspond to the ones usually used in FITS tables. Note that if a CHARACTER column is created then the default CHARACTER size of twenty will be adopted unless `CAT_TATTC` is used to set the character size.

`CAT_CINQ (FI, SZDIM; CI, FNAME, GENUS, EXPR, DTYPE, CSIZE, DIMS, SIZEA, NULL, EXCEPT, SCALEF, ZEROP, ORDER, UNITS, EXTFMT, PRFDSP, COMM, DATE; STATUS)`
Inquire the values of all the attributes for a column. Type conversions are performed if necessary. If the conversion fails a status is set (this is only likely to be important for null values). Note that the exception value is forced into type CHARACTER in order to avoid having a family of routines. Note also that the genus attribute is returned explicitly.

7.8.4 Parameters

`CAT_PPTA<t> (CI, QNAME, CSIZE, DIMS, SIZEA, UNITS, EXTFMT, PRFDSP, COMM, VALUE; QI; STATUS)`
Create a parameter, simultaneously setting all its attributes.

`CAT_PPTS<t> (CI, QNAME, VALUE, COMM; QI; STATUS)`
Create a parameter, simultaneously setting its value and comment attributes. These attributes deliberately correspond to the ones usually used in FITS tables. Note that if a CHARACTER parameter is created then the default character size of `CAT__SZVAL` will be adopted unless `CAT_TATTC` is used to set the character size.

`CAT_PINQ (QI, SZDIM; CI, PNAME, DTYPE, CSIZE, DIMS, SIZEA, UNITS, EXTFMT, PRFDSP, COMM, VALUE, DATE; STATUS)`
Inquire the values of all the attributes for a parameter.

7.8.5 Expressions

`CAT_EIDNT (CI, EXPR; EI; STATUS)`
Get an identifier for an expression. The syntax of expressions is described in Appendix B.

7.8.6 Selections

The selection routines return a selection identifier which corresponds to the set of rows which satisfy the criteria of the selection. There are two routines for generating a selection: `CAT_SELCT` and `CAT_SFND<t>`. `CAT_SELCT` allows complex selections to be performed according to complicated criteria. A logical (or boolean) expression defining the selection is supplied and all the rows in the catalogue for which this expression evaluates to `.TRUE.` are selected. `CAT_SFND<t>` allows a simple range of values to be selected for a sorted column. Usually `CAT_SELCT` will execute more slowly than `CAT_SFND<t>` because whereas `CAT_SELCT` must necessarily access every row in the catalogue, `CAT_SFND<t>` can exploit the ordering of the chosen column to immediately identify the required rows (remember the mnemonic F for Find and Fast, S for Select and Slow).

`CAT_SLIST` allows a specialized application to create a non-standard selection.

All the selection routines may operate on either an entire catalogue or some previous selection. They all uniformly and consistently provide an option to create a second selection of all the rejected rows.

`CAT_SELCT (CI, EI, REJFLG; SI, NUMSEL, SIR, NUMREJ; STATUS)`
Create a selection of rows satisfying some expression.

`CAT_SFND<t> (CI, FI, MINRNG, MAXRNG, REJFLG; SI, NUMSEL, SIR, NUMREJ; STATUS)`
Create a selection of the rows in a catalogue for which the fields of a specified sorted column lie within a given range.

`CAT_SLIST (NUMSEL, SELIST, CRIT, REJFLG, CI, SI, SIR, NUMREJ, STATUS)`
Create a selection from an array of row numbers.

`CAT_SINQ (SI; CI, EXPR, NUMSEL, COMM, DATE; STATUS)`
Inquire all the attributes of a selection.

7.8.7 Indices

The index generation routine generates an index from a given numeric column and returns an identifier which allows the rows in the catalogue to be accessed as though they were sorted on the column. *In CAT version 9.0 only temporary indices, which persist for the duration of the application which generated them, are supported.*

`CAT_INEW (FI, DISP, ORDER; II; STATUS)`
Create an index on a column.

`CAT_IINQ (II; CI, FI, ORDER, NUMSEL, COMM, DATE; STATUS)`
Inquire all the attributes of an index.

7.8.8 Row level manipulation routines

These routines manipulate a single row in a catalogue. They are based around the concepts of the 'current row' and the 'current row buffer'. The current row is a single row in a catalogue which CAT is currently operating on. A copy of the current row is kept in a buffer within CAT. The various row and field manipulation routines operate on the copy of the current row in the current row buffer. As appropriate, the current row buffer can be copied out to the catalogue and a new row copied into the current row buffer.

The 'current row buffer' is an abstract concept to describe how CAT behaves when an application calls it. It is my solution to obtaining an arbitrary number of fields of arbitrary type in a single pass through a catalogue. The trick is separating reading rows in the catalogue into the 'current row buffer' and having the GET and PUT operations operate on this buffer. Applications should be written in the form:

```
for all the rows to be processed
  Read the required row into the current row buffer.
  Get a field from the buffer.
  Get another field from the buffer.
  Get yet another field from the buffer.
  and so on...
end for
```

None of the solutions to this problem are ideal and the disadvantage of the one adopted is that the resulting subroutine interface is quite 'low level' and verbose to use. However, it does allow

an arbitrary number of columns of arbitrary type to be extracted in a single pass through the catalogue.

New rows can be added to an existing catalogue only by appending them to the end of the catalogue.

When a catalogue is opened (that is, an identifier is obtained for it), its first row is copied into the current row buffer. Thus, if the current row buffer is accessed without first GETting a specified row with `CAT_RGET` the contents of the first row will be obtained.

`CAT_RGET (CI, ROWNO; STATUS)`
: Read a specified row from the catalogue into the current row buffer. `CI` may be either a catalogue, selection or index identifier.

`CAT_RAPND (CI; STATUS)`
: Append the current row buffer as a new row at the end of the catalogue. `CI` must necessarily be a catalogue identifier.

7.8.9 Getting and putting values

The basic routines for getting and putting values GET from and PUT to the current row buffer. The following rules apply:

- scalar values are always GOT from expressions (that is, an expression is evaluated using the values of fields extracted from the current row buffer). However, simple column names and parameter names are valid expressions. Therefore, any one of an: expression, column, vector column element or parameter identifier may be used in the GET routine `CAT_EGT0<t>`,
- values can only be PUT to scalar columns or vector column elements, not to expressions or parameters (the notion of PUTting to an expression is a nonsense and there are alternative routines for setting the value of a parameter).

Section 8.2.5 prescribes how applications should handle null values.

`CAT_EGT0<t> (GI; VALUE, NULFLG; STATUS)`
: Get the value of a scalar expression, evaluated from the current row buffer. `GI` may be either: an expression, column, vector column element or parameter identifier.

`CAT_EGT0F (GI; VALUE, NULFLG; STATUS)`
: Get the value of a scalar expression, evaluated from the current row buffer and formatted into a character string using the external display format for the column, parameter or expression. `GI` may be either: an expression, column, vector column element or parameter identifier.

`CAT_PUT0<t> (FI, VALUE, NULFLG; STATUS)`
: Put a value to a field. The value is written to the current row buffer. `FI` may be either a column or vector column element identifier.

For convenience two additional routines are provided for getting values which include the row from which the value is to be obtained. Otherwise they are similar to CAT_EGT0<t> and CAT_EGT0F (indeed they are wrap-arounds of CAT_RGET followed by CAT_EGT0<t> or CAT_EGT0F).

CAT_FGT0<t> (CI, ROWNO, GI; VALUE, NULFLG; STATUS)
: Get the value of a scalar expression, evaluated from a specified row. CI may be either: a catalogue, selection or index identifier. GI may be either: an expression, column, vector column element or parameter identifier.

CAT_FGT0F (CI, ROWNO, GI; VALUE, NULFLG; STATUS)
: Get the value of a scalar expression, evaluated from a specified row. and formatted into a character string using the external display format for the column, parameter or expression. CI may be either: a catalogue, selection or index identifier. GI may be either: an expression, column, vector column element or parameter identifier.

7.9 Textual information

This set of routines provide access to any textual information associated with the catalogue[14]. They try to keep access to the textual information as simple as possible. The text is accessed one line at a time, with each line corresponding, for example, to a single FITS COMMENT or HISTORY keyword. No facilities are provided to interpret a line. It is assumed that each line will either be displayed to the user or copied to another catalogue.

CAT_GETXT returns a single line of text. Subsequent calls to CAT_GETXT work sequentially through the textual information, returning the lines in sequence. The application cannot specify that it wants a particular line. However CAT_RSTXT is provided to 'reset' the sequence for a catalogue, so that the textual information can be read through an arbitrary number of times.

Each line has a class associated with it. When a line is read the class is returned; an application cannot prescribe that it wants a class of a particular value. The classes supported vary for the different catalogue formats, though on output the standard classes COMMENT and HISTORY are always available. Appendix C lists the classes supported by individual catalogue formats.

A call to CAT_PUTXT appends a line of textual information to the end of the existing textual information for a catalogue. The only way to add new textual information is by appending it to the end.

In all these routines CI must be a catalogue identifier.

CAT_GETXT (CI; FINISH, CLASS, TEXT; STATUS)
: Get the next line of textual information from a catalogue.

CAT_PUTXT (CI, CLASS, TEXT; STATUS)
: Put a line of textual information to a catalogue.

CAT_RSTXT (CI; STATUS)
: Reset the access to the textual information in a given catalogue. A subsequent call to CAT_GETXT would return the first line of textual information.

[14]These routines were not included in the original specification for the CAT subroutine interface, as described in *The Starlink Subroutine Interface for Manipulating Catalogues* (StarBase/ACD/3.4)[2]; they were added later.

`CAT_SZTXT (CI, ACCESS; LINESZ; STATUS)`
Return the maximum permitted size of a line of textual information in a catalogue. The maximum size is determined by the format of the catalogue and the access mode (READ or WRITE).

7.10 Miscellaneous

`CAT_TUNES (CATPRM, VALUE; STATUS)`
Set a CAT tuning parameter. Note that all these parameters are of type CHARACTER.

`CAT_TUNEG (CATPRM; VALUE; STATUS)`
Get a CAT tuning parameter. Note that all these parameters are of type CHARACTER.

`CAT_TYFMT (DTYPE, CSIZE; STRING, POSN; STATUS)`
Construct a character string representation of a CAT data type and append it to a character string. This character string representation is (deliberately) identical to that used by HDS.

`CAT_SRNG<t> (CI, FI, MINRNG, MAXRNG; FIRSTR, LASTR; STATUS)`
Get the rows corresponding to a range for a sorted column. This routine is similar to `CAT_SFND<t>`, but it returns the first and last rows within the specified range, rather than generating a selection.

8 Features of the library

This section describes some aspects of the behaviour of the CAT library.

8.1 Copying parameters and fields

Parameters and fields have a defined data type (_DOUBLE, _REAL etc.). Families of subroutines are available for transferring the values of parameters, columns and fields between a catalogue and Fortran variables in an application program invoking the CAT interface. These Fortran variables may be of a variety of types. The type of the column or parameter need not correspond to the type of the Fortran variable; the interface will automatically attempt a type conversion. However, depending on the types involved, a conversion may not be possible for some values. The matrix of possibilities is shown in Table 10. This matrix is used for *all* conversions: both GET and PUT operations for both columns (or fields) and parameters.

If a type conversion fails the appropriate exception value (null or locum, see Section 8.2, below) is substituted for the missing value. The error status is not set (that is, it remains ok) and no error message is generated. This procedure is followed consistently for all type conversions. It is adopted because, in practice, conversion failures are common.

The rules for converting to and from the _LOGICAL data type are as follows

From logical

- Conversion from LOGICAL to CHARACTER*n gives 'TRUE' or 'FALSE' if n is greater than or equal to 5 and 'T' or 'F' otherwise.

Destination type	Start or initial type								
	UBYTE	BYTE	UWORD	WORD	INTEGER	REAL	DOUBLE	LOGICAL	CHAR
UBYTE	yes	maybe	maybe	maybe	maybe	maybe	maybe	yes	maybe
BYTE	maybe	yes	maybe	maybe	maybe	maybe	maybe	yes	maybe
UWORD	yes	yes	yes	maybe	maybe	maybe	maybe	yes	maybe
WORD	yes	yes	maybe	yes	maybe	maybe	maybe	yes	maybe
INTEGER	yes	yes	yes	yes	yes	maybe	maybe	yes	maybe
REAL	yes	yes	yes	yes	yes	yes	maybe	yes	maybe
DOUBLE	yes	yes	yes	yes	yes	yes	yes	yes	maybe
LOGICAL	yes	yes	yes	yes	yes	yes	yes	yes	yes
CHAR	maybe	maybe	maybe	maybe	maybe	maybe	maybe	yes	maybe

Key:

yes conversion is always possible,

maybe conversion may be possible.

Note that converting a character string to a character string may result in an error because truncation may occur.

Table 10: Conversion matrix for copying different types

- Conversion from LOGICAL to INTEGER is defined such that true converts to 1 and false converts to 0.
- Conversion from LOGICAL to a numeric type except INTEGER is equivalent to converting LOGICAL to INTEGER and then INTEGER to the required numeric type.

To logical

- Any character string beginning with 'T' (upper or lower case) converts to true; any other string converts to false.
- Any even INTEGER converts to false; any odd INTEGER converts to true (this behaviour is specified for compatibility with HDS).
- Conversion for other numeric types is equivalent to conversion from the type to INTEGER and then from INTEGER to LOGICAL.

8.2 Null and locum values

8.2.1 Rationale and behaviour

Null values are used to represent a datum where no actual value is available. An example might be multi-colour photometry for a set of stars where measures for some colours are missing for some of the stars. Null values would be used to represent the missing values. Throughout CAT, and the applications that call it, nulls have the single, simple meaning that 'no value is available

for this datum'. It is possible to invent schemes where a set of null values are supported, each with a subtly different gradation of meaning (see, Roth *et al.*[6]). However, such schemes were adjudged un-necessary for manipulating astronomical catalogues and tables, and they are *not* supported.

The treatment of nulls in CAT was designed with the following assertions about null values in astronomical catalogues in mind:

- null values occur in existing catalogues. A wide variety of values are used to represent null values. Null values are common rather than rare,
- in general, it is *always* possible that a genuine datum will not be available when any field in a catalogue is being written; in general an arithmetic exception, such as ÷ by zero, is always possible. *It is not possible to proscribe exceptions when generating a column*,
- conversely, for some columns, all the values permitted by the data type of the column will be required for genuine values, leaving no value available for representing the null value. An example might be a column of type _UBYTE used to record values generated by an astronomical instrument. The entire range of values permitted by this data type is 0 to 255, and if the instrument could generate genuine values in this entire range, there is no unused value left to represent the null case.

An additional, though less compelling, constraint was that it may be necessary to generate tables where the value of the null was specified, rather than adopting a fixed, pre-defined value. This feature may be required, for example, so that the table can be input to existing programs.

In order to meet these various requirements, the following behaviour was specified for null values in CAT:

- when a column is created, it may be specified as either supporting, or not supporting, null values,
- if a column supports nulls then a fixed, pre-defined null value will be used by default. These default nulls are the standard HDS 'bad values' for the various data types, where they exist. The HDS bad values are defined in SUN/39[10]. See in particular Sections 1.2 and 5.1. It is necessary to define a set of values for each operating system and type of computer that CAT runs on. Table 11 gives the values for some of the types currently used by Starlink. Note that the HDS parametric constants for the values are used to define the corresponding CAT types, rather than merely making their values the same. This approach reduces the possibility of incompatibilities arising if the values change,
- alternatively, for a column which supports nulls, the null value may be replaced with a specified value,
- if a column does not support nulls, exceptions can still occur when its fields are generated (as described above). When an exception occurs a **locum** value is written. The locum value for a column *must* be specified when a column which does not support nulls is created.

The difference between null and locum values is subtle and, at first, a little confusing. It may be summarized as follows:

HDS type	HDS symbolic constant
_UBYTE	VAL__BADUB
_BYTE	VAL__BADB
_UWORD	VAL__BADUW
_WORD	VAL__BADW
_INTEGER	VAL__BADI
_REAL	VAL__BADR
_DOUBLE	VAL__BADD
_CHAR[$*n$]	-

HDS type	HDS bad value		Notes
	DECstation	SUN	
_UBYTE	'FF'X	'FF'X	
_BYTE	'80'X	'80'X	
_UWORD	'FFFF'X	'FFFF'X	
_WORD	'8000'X	'8000'X	
_INTEGER	'80000000'X	'80000000'X	
_REAL	'FF7FFFFF'X	'FF7FFFFF'X	
_DOUBLE	'FFEFFFFFFFFFFFFF'X	'FFEFFFFFFFFFFFFF'X	
_CHAR[$*n$]	'0'X	'0'X	1

Notes:

(1) The ASCII standard defines a byte with a value of 0 (that is, all bits set to 0) as a null character.

(2) No NULL value is defined for the _LOGICAL data type since by definition a _LOGICAL value can only take two values (true or false).

Table 11: HDS bad values

- on writing a field there is no difference; if an exception occurs, a column with nulls generates a null and a column without nulls generates a locum,
- on reading a field (and this is the crux of the matter), CAT can differentiate a null from a genuine data value, but it *cannot* differentiate a locum: *once written a locum is indistinguishable from a genuine datum.*

Thus, generating a locum corresponds to CAT inventing data, and in theory (and often in practice) will invalidate the contents of the column. This invalidation is the reason why the value of a locum *must* be specified, rather than making a default available; a value can at least be chosen to minimize the damage.

8.2.2 Reporting the generation of null and locum values

The generation of nulls when writing a new field is common. Therefore, the generation of a null is not reported in any way: no error message is generated and no error status is set.

8.2.3 CAT subroutine interface

The NULL and EXCEPT attributes of a column (see Section 6.7) control the way that null or locum values are handled for that column. They are specified when the column is created and are immutable thereafter. The NULL attribute defines the type of null or locum which the column supports. The possibilities are:

- standard, HDS bad value (`CAT__NULLD`),
- explicitly specified null value (`CAT__NULLS`),
- locum (`CAT__LOCUM`).

If the standard HDS null value is being used then it is not necessary to associate its actual value with the column; knowing that the standard value is to be used is adequate. Conversely, if either:

- the value of the null has been specified explicitly,
- a locum value has been specified,

then the required value must be associated with the column. In either case, this value is stored in the exception attribute of the column.

In the routines to GET and PUT values through the CAT subroutine interface, null values are indicated by flags which appear as separate arguments in the calling sequences for the GET and PUT subroutines. However, the appropriate Starlink bad value is also substituted for the value in the actual data variable or array. The advantages of this approach are:

- the separate flag for nulls is convenient for processing in database applications,
- the embedded values in the data variable or array mean that arrays (or variables, though variables are less likely to be important in practice) can be passed straight into existing Starlink subroutines.

8.2.4 Handling nulls in expressions

The following rules apply to evaluating an expression in which one or more of the fields is null.

(1) A numeric expression containing column names evaluates to null if any of the fields are null.

(2) Logical expressions can evaluate to one of three states: *true*, *false* or *null*. However, the expression is only satisfied if the value is *true*.

(3) A relational operation (><=, etc.) between two columns evaluates to null if either or both of the fields is null.

(4) The logical operators AND, OR and NOT operate on a three-valued logic[15]. Their truth tables are shown in Table 12.

(5) The parser provides the function `NULL` to detect null values. It returns the logical value `.TRUE.` if its argument is null and `.FALSE.` if its argument is not null.

AND

b \ a	T	N	F
T	T	N	F
N	N	N	F
F	F	F	F

OR

b \ a	T	N	F
T	T	T	T
N	T	N	N
F	T	N	F

NOT

a	
T	F
F	T
N	N

Table 12: Truth tables for three-valued logic

[15]Under Scottish law three verdicts are possible: guilty, not guilty and 'not proven.' If Scottish judges, advocates and juries can manage three-valued logic, it seemed reasonable that CAT could.

8.2.5 Handling nulls in applications

Normally *applications* should observe the following rules when manipulating null values. If applications usually follow these rules they will all present consistent and predictable behaviour to their users. In a perfect world applications would always follow the rules. However, I recognize that some *ad hoc* applications may need to behave differently because of some idiosyncrasy of their required functionality. Nonetheless, applications should follow the rules unless there are very good reasons not to. The recommended rules are:

(1) *applications* extracting a single column from a table (for example, to plot a histogram) should skip (that is, detect, but not process) null values. They may, however, count and report the number of null values encountered,

(2) *applications* extracting two or more columns from a table (for example, to plot a scattergram) should skip (that is, detect, but not process) all rows where any of the extracted values is null. They may, however, count and report the number of null values and/or rows encountered,

8.2.6 Inquiring null values

An application may determine the null value specified for a column by inquiring the value of the EXCEPT attribute using routine `CAT_TIQAC`. The value is forced into type CHARACTER. If it is impossible to represent the value adequately in a character string then it cannot be made accessible to the application.

8.3 Storing and representing columns of angles

CAT provides special facilities for representing columns which contain angles. Usually angular columns are used to store celestial coordinates such as Right Ascension and Declination, though they can contain any angular measure. There are two requirements for the treatment of angles stored in astronomical catalogues:

- inside a program angles should be expressed in radians and stored in DOUBLE PRECISION or REAL variables for ease of processing,
- usually they are best displayed to a user in units of hours or degrees, subdivided into sexagesimal minutes and seconds. Alternatively, small angles might be best displayed as minutes or seconds of arc or time and represented to some number of places of decimals.

CAT provides both these facilities, as follows:

- when the value of a field in an angular column is obtained using `CAT_EGT0D` or `CAT_EGT0R` it is returned as DOUBLE PRECISION or REAL variable in radians,
- when the value of a field in an angular column is obtained formatted for display using `CAT_EGT0F` it is returned as a CHARACTER string containing the value expressed in hours, degrees, minutes or seconds and optionally formatted as a sexagesimal value.

In order for CAT to know that a column contains an angle it must satisfy the following two requirements.

- The data type must be DOUBLE PRECISION or REAL[16],
- The units attribute of the column should be set to 'RADIANS' followed by an angular format specifier enclosed in curly brackets ('{}'). The simplest forms of this angular format specifier are simply 'HOURS' and 'DEGREES' for hours and degrees respectively. Thus, examples of the UNITS attribute are:

  RADIANS{HOURS} to display the column in hours,
  RADIANS{DEGREES} to display the column in degrees.

  The angular format specifiers are described in full in the following section.

If the external display format attribute of the column, EXFMT, is explicitly set it should be set to a normal Fortran 77 format specifier corresponding to the data type of the column, for example 'D16.8' for a DOUBLE PRECISION column or 'E14.6' for a REAL column[17].

When the catalogue is written fields should be written to angular columns using CAT_PUTOD or CAT_PUTOR with the angles expressed in radians.

8.3.1 Angular format specifiers

The angular format specifier forms part of the UNITS attribute for an angular column. The UNITS attribute for an angular column has the form:

```
RADIANS{angular format specifier}
```

The simplest angular format specifiers are 'HOURS' and 'DEGREES'.

HOURS will cause the angle to be displayed as hours, minutes and seconds, with the seconds displayed to one place of decimals,

DEGREES will cause the angle to be displayed as degrees, minutes and seconds, with the seconds displayed as a whole number.

If the angular format specifier is omitted altogether and the UNITS attribute simply set to 'RADIANS' or 'RADIANS{}' then the angle will be interpreted exactly as though the angular format specifier had been 'DEGREES'. There are additional simple angular format specifiers for displaying angles as minutes or seconds of arc or time to a specified number of decimal places:

ARCMIN.*n* minutes of arc,

[16] If the angle is to be stored to an accuracy of seconds of arc or fractions of a second of arc then it must be of type DOUBLE PRECISION. You should always store celestial coordinates in DOUBLE PRECISION columns unless you are certain that they are only of low accuracy. Angles are only supported in REAL columns for completeness; I do not expect that they will be used often.

[17] The external format attribute *must* always be a valid Fortran 77 format specifier because of the way that CAT catalogues are represented as FITS tables.

`ARCSEC`***.n*** seconds of arc,

`TIMEMIN`***.n*** minutes of time,

`TIMESEC`***.n*** seconds of time.

.n is the number of decimal places required. If *.n* is omitted then the value will be displayed as an integer number. Though these angular specifiers can be used to display any angle, obviously they are most likely to be useful for small angles.

These simple angular format specifiers will usually be adequate for representing columns of celestial coordinates. However, sometimes you might wish to specify a different representation for an angle. CAT accepts angular format specifiers which permit angles to be represented in a number of different formats. These specifiers are constructed from a selection from amongst the following elements:

`I B L + Z H D M S T` *.n*

The meaning of each of the individual elements is as follows.

`I` Use the ISO standard separator for expressing times, a colon (':'), to separate hours or degrees, minutes and seconds.

`B` Use a single blank space to separate hours or degrees, minutes and seconds.

`L` Use a letter (`h`, `d`, `m`, or `s`, as appropriate) to separate hours or degrees, minutes and seconds.

`+` Insert a plus sign ('+') before positive angles (a minus sign is, of course, always inserted before negative angles).

`Z` Insert leading zeros before the hours, degrees, minutes or seconds. Hours, minutes and seconds are assumed to be two-digit numbers and degrees three-digit.

`H` Express the angle in units of hours.

`D` Express the angle in units of degrees.

`M` If an `M` occurs when either `H` or `D` is present then it indicates that the hours or degrees are to be subdivided into sexagesimal minutes. If an `M` occurs when neither `H` nor `D` is present then it indicates that the units are minutes of either arc or time.

`S` If an `S` occurs when an `M` is present then it indicates that the minutes are to be subdivided into sexagesimal seconds (the minutes may be either the actual units or themselves a sexagesimal subdivision of hours or degrees; see `M` above). If an `S` occurs when an `M` is not present then it indicates that the units are seconds of arc or time.

`T` In the case where `H` and `D` are both absent and either or both of `M` and `S` are present then `T` indicates that the units are minutes or seconds of time. If it is omitted in this case then the units are minutes or seconds of arc. If either `H` or `D` is present then `T` is ignored.

.n Display the least significant unit (seconds, minutes, degrees or hours, as appropriate) to *n* decimal places.

Any of the items may be omitted, down to and including a completely blank specifier.

The items can occur in any order, except that *.n* must occur last. However, for human readability I recommend that the items occur in the order:

(any of: `I`, `B`, `L`, `+` or `Z`) (`H` or `D`) `M S T` *.n*

If items are omitted the following defaults apply.

- If neither `I`, `B` nor `L` is specified then `I` is assumed.
- If `+` is omitted then positive angles are not preceded by a '+' sign.
- If `Z` is omitted then leading zeros are omitted in the primary units (hours, degrees, minutes or seconds), but leading zeros are always included in any sexagesimal subdivisions.
- If none of `H`, `D`, `M` or `S` are specified then `D` is assumed (that is, the default units are degrees).
- If `H` or `D` are present but `M` is omitted then the hours or degrees are not subdivided into minutes.
- If `M` is present but `S` is omitted then the minutes are not subdivided into seconds.
- If `S` is present in addition to `H` or `D` but `M` is absent then `S` is ignored (this case is technically illegal).
- If *.n* is omitted then the least significant unit (seconds, minutes, degrees or hours, as appropriate) is displayed as a whole number, without any places of decimals.

Table 13 lists a number of examples of angular format specifiers which might be used to represent 'large' angles, such as celestial coordinates, together with examples of how they would represent an angle. Table 14 lists a number of examples of angular format specifiers which might be used to represent small angles, such as the great circle distance between two neighbouring objects or the angular size of an extended object, together with examples of how they would represent an angle.

The simple angular format specifiers, '`HOURS`', '`DEGREES`', '`ARCMIN`', '`ARCSEC`', '`TIMEMIN`' and '`TIMESEC`' are just synonyms for particular cases of the general specifiers. They are listed, together with the equivalent full specification in Table 15.

8.3.2 Displaying angles in radians

Displaying angles in units of hours, degrees, minutes or seconds, optionally formatted as sexagesimal values is usually the required behaviour. However, occasionally you may want to display angles as simple decimal numbers expressed in radians. CAT also provides this facility. It can be configured so that `CAT_EGT0F` returns an angle expressed in radians, written into a CHARACTER string using the Fortran 77 external format specifier for the column (defined by attribute EXFMT). In this case CAT is treating columns of angles just like any other column; `CAT_EGT0F` simply writes the value into a CHARACTER string using the external display format specifier.

Specifier	Example	Notes
`D`	`63`	Integer degrees
`D.2`	`62.86`	Degrees to two places of decimals
`DM`	`62:52`	Degrees and integer minutes
`DM.2`	`62:51.58`	Degrees and minutes to two places of decimals
`DMS`	`62:51:35`	Degrees, minutes and integer seconds
`DMS.2`	`62:51:34.65`	Degrees, minutes and seconds to two places of decimals
`H`	`4`	Integer hours
`H.2`	`4.19`	Hours to two places of decimals
`HM`	`4:11`	Hours and integer minutes
`HM.2`	`4:11.44`	Hours and minutes to two places of decimals
`HMS`	`4:11:26`	Hours, minutes and integer seconds
`HMS.2`	`4:11:26.31`	Hours, minutes and seconds to two places of decimals
`BHMS.2`	`4 11 26.31`	Space character as separator
`LHMS.2`	`4h11m26.31s`	Letter as separator
`ZHMS.2`	`04:11:26.31`	Leading zeros
`+HMS.2`	`+4:11:26.31`	Signed value
`L+ZDM.3`	`+062d51.577`	Letter separator, leading zeros and signed

The examples show how the various specifiers would represent an angle of 1.09710742 radians (or $62^\circ\,51'\,34''65$).

Table 13: Examples of sexagesimal format specifiers

Specifier	Example	Notes
`M`	`3`	Integer minutes of arc
`M.3`	`3.227`	Minutes of arc to three places of decimals
`MS`	`3:14`	Minutes and integer seconds of arc
`MS.3`	`3:13.600`	Minutes and seconds of arc to three places of decimals
`S`	`194`	Integer seconds of arc
`S.3`	`193.600`	Seconds of arc to three places of decimals
`MT`	`0`	Integer minutes of time
`MT.3`	`0.215`	Minutes of time to three places of decimals
`MST`	`0:13`	Minutes and integer seconds of time
`MST.3`	`0:12.907`	Minutes and seconds of time to three places of decimals
`ST`	`13`	Integer seconds of time
`ST.3`	`12.907`	Seconds of time to three places of decimals
`BMS`	`3 14`	Space character as separator
`LMS`	`3m14s`	Letter as separator
`ZMS`	`03:14`	Leading zeros
`+MS`	`+3:14`	Signed value
`L+ZMS`	`+03m14s`	Letter separator, leading zeros and signed

These specifiers might typically be used to represent the great circle distance between neighbouring objects or the angular size of an extended object. There is no reason why they should not be used to represent 'large' angles such as celestial coordinates, though the output would look a bit odd. The examples show how the various specifiers would represent an angle of 9.3860x10^{-4} radians (or $0^\circ\ 3'\ 13''66$).

Table 14: Examples of angular format specifiers for small angles

Simple Specifier	Equivalent Full Specifier	Example	Notes
HOURS	IHMS.1	14:11:26.3	1
DEGREES	IDMS	62:51:35	1
ARCMIN	M	3	2
ARCSEC	S	194	2
TIMEMIN	MT	0	2
TIMESEC	ST	13	2
ARCMIN.3	M.3	3.227	3
ARCSEC.3	S.3	193.600	3
TIMEMIN.3	MT.3	0.215	3
TIMESEC.3	ST.3	12.907	3

Notes

(1) The number of decimal places is fixed for these specifiers.

(2) The number of decimal places has been omitted so integers without any decimal places are assumed.

(3) Three places of decimals were specified.

The example for the first two specifiers is an angle of 1.09710742 radians; for the remaining specifiers the example is an angle of 9.3860×10^{-4} radians.

Table 15: The simple angular format specifiers and their equivalents

To control whether angles are converted to hours, degrees, minutes or seconds and optionally formatted as sexagesimal values or simply written as a decimal number in radians you must set a configuration (or 'tuning') parameter in CAT. This parameter is called 'ANGLE_LIST' and is set using routine CAT_TUNES. Proceed as follows.

- to configure CAT to return angles as radians:

```
CALL CAT\_TUNES ('ANGLE_LIST', 'RADIANS', STATUS)
```

- to configure CAT to return angles as hours, degrees, minutes or seconds, optionally formatted as sexagesimal values:

```
CALL CAT_TUNES ('ANGLE_LIST', 'SEXAGESIMAL', STATUS)
```

Once one of these alternatives has been set it will remain in effect until another call is made to CAT_TUNES to reset it. The default if the representation is never explicitly set is that angles are converted to hours, degrees, minutes or seconds and optionally output as sexagesimal values.

You can inquire the current value of tuning parameter 'ANGLE_LIST' using routine CAT_TUNEG:

```
CHARACTER
:  TVALUE*75   ! Current value for angle representation.
  .
  .
  .


CALL CAT_TUNEG ('ANGLE_LIST', TVALUE, STATUS)
```

and TVALUE will be returned set to 'RADIANS' or 'SEXAGESIMAL', as appropriate.

A Detailed subroutine specifications

This Appendix gives detailed specifications for the individual subroutines in the CAT library. It is probably best used in conjunction with the discussion of the routines in Section 7. The routines are listed in alphabetical order.

CAT_ASSOC
Open an existing catalogue; the name of the catalogue is obtained from an ADAM parameter

Description:

Open an existing catalogue; the name of the catalogue is obtained from an ADAM parameter. If an existing catalogue is opened with MODE = 'WRITE' then it is overwritten.

Invocation:

```
CALL CAT_ASSOC (PCNAME, MODE; CI; STATUS)
```

Arguments:

PCNAME = CHARACTER∗(∗) (Given)

Name of the ADAM parameter from which the catalogue name will be obtained.

MODE = CHARACTER∗(∗) (Given)

Mode in which the catalogue will be accessed. One of: READ - the catalogue may only be read from, WRITE - a new catalogue is to be written.

CI = INTEGER (Returned)

Catalogue identifier.

STATUS = INTEGER (Given and Returned)

The global status.

Copyright :

Bugs:

None known

CAT_CINQ
Inquire the values of all the attributes for a column

Description:

Inquire the values of all the attributes for a column. Type conversions are performed if necessary. If the conversion fails a status is set (this is only likely to be important for null values). Note that the exception value is forced into type character in order to avoid having a family of routines. Note also that the genus attribute is returned explicitly.

Invocation:

```
CALL CAT_CINQ (FI, SZDIM; CI, FNAME, GENUS, EXPR, DTYPE, CSIZE, DIMS, SIZEA, NULL, EXCEPT,
SCALEF, ZEROP, ORDER, UNITS, EXTFMT, PRFDSP, COMM, DATE; STATUS)
```

Arguments:

FI = INTEGER (Given)
Identifier to the column.

SZDIM = INTEGER (Given)
Maximum permitted dimensionality for a column (defines the size of array SIZEA).

CI = INTEGER (Returned)
Catalogue identifier.

FNAME = CHARACTER∗(∗) (Returned)
Name of the column (or field).

GENUS = INTEGER (Returned)
The genus of the column, coded as follows: CAT__GVIRT - virtual column, CAT__GPHYS - physical column.

EXPR = CHARACTER∗(∗) (Returned)
The expression defining a virtual column; blank for a physical column.

DTYPE = INTEGER (Returned)
Type of the column. The permitted types are identical to HDS types.

CSIZE = INTEGER (Returned)
Size of a character column. If the column is not of type character CSIZE is set to zero.

DIMS = INTEGER (Returned)
Dimensionality of the column. For a scalar DIMS = 0.

SIZEA(SZDIM) = INTEGER (Returned)
The size of the array in each of its dimensions. The exception is a scalar column, when SIZEA becomes a single-element, one-dimensional array, set to 0.

NULL = INTEGER (Returned)
The way that null values are handled for the column. The permitted values are: CAT__NULLD - default, HDS null values used, CAT__NULLS - null values explicitly specified for the column, CAT__LOCUM - null values not supported for the column and a locum value used instead.

EXCEPT = CHARACTER∗(∗) (Returned)
In the cases where either the column supports explicitly specified null values, or nulls are not supported and a locum is used instead, EXCEPT contains the required value, written into a character string.

SCALEF = DOUBLE PRECISION (Returned)
Scale factor for scaled columns. 0.0D0 for columns which are not scaled.

ZEROP = DOUBLE PRECISION (Returned)
Zero point for scaled columns. 0.0D0 for columns which are not scaled.

ORDER = INTEGER (Returned)
The order in which values occur in the column, coded as follows: CAT__ASCND - ascending, CAT__DSCND - descending, CAT__NOORD - none.

UNITS = CHARACTER∗(∗) (Returned)
The units of the column.

EXTFMT = CHARACTER∗(∗) (Returned)
The external format for the column.

PRFDSP = LOGICAL (Returned)
The preferential display flag for the column: .TRUE. - display the column by default, .FALSE. - do not display the column by default.

COMM = CHARACTER∗(∗) (Returned)
Comments about the column.

DATE = DOUBLE PRECISION (Returned)
The modification date of the column.

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

Copyright :

Bugs:
None known

CAT_CNEWA
Create a column, simultaneously setting all its attributes

Description:

Create a column, simultaneously setting all its attributes.

Note that the GENUS attribute is automatically set to CAT__GPHYS.

Invocation:

```
CALL CAT_CNEWA (CI, FNAME, EXPR, DTYPE, CSIZE, DIMS, SIZEA, NULL, EXCEPT, SCALEF, ZEROP,
ORDER, UNITS, EXTFMT, PRFDSP, COMM; FI; STATUS)
```

Arguments:

CI = INTEGER (Given)
Catalogue identifier.

FNAME = CHARACTER∗(∗) (Given)
Name of the column (or field).

EXPR = CHARACTER∗(∗) (Given)
The expression defining a virtual column; blank for a physical column.

DTYPE = INTEGER (Given)
Type of the column.

CSIZE = INTEGER (Given)
Size of a CHARACTER column. If the column is not of type CHARACTER CSIZE is irrelevant; it is conventional to set it to zero.

DIMS = INTEGER (Given)
Dimensionality of the column. The permitted values are: CAT__SCALR - scalar, CAT__VECTR - vector.

SIZEA(1) = INTEGER (Given)
If the column is a vector this attribute should be set to the number of elements in the vector. For a scalar it should be set to one.

NULL = INTEGER (Given)
The way that null values are handled for the column. The permitted values are: CAT__NULLD - default, HDS null values used, CAT__NULLS - null values explictly specified for the column, CAT__LOCUM - null values not supported for the column and a locum value used instead.

EXCEPT = CHARACTER∗(∗) (Given)
In the cases where either the column supports explicitly specified null values, or nulls are not supported and a locum is used instead, EXCEPT contains the required value, written into a character string.

SCALEF = DOUBLE PRECISION (Given)
Scale factor for scaled columns. 1.0D0 for columns which are not scaled.

ZEROP = DOUBLE PRECISION (Given)
Zero point for scaled columns. 0.0D0 for columns which are not scaled.

ORDER = INTEGER (Given)
The order in which values occur in the column, coded as follows: CAT__ASCND - ascending, CAT__DSCND - descending, CAT__NOORD - none.

UNITS = CHARACTER∗(∗) (Given)
The units of the column.

EXTFMT = CHARACTER∗(∗) (Given)
The external format for the column.

PRFDSP = LOGICAL (Given)
The preferential display flag for the column: .TRUE. - display the column by default, .FALSE. - do not display the column by default.

COMM = CHARACTER∗(∗) (Given)
Comments about the column.

FI = INTEGER (Returned)
Identifier for the column.

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

Copyright :

Bugs:
None known

CAT_CNEWS
Create a column, simultaneously setting some of its attributes

Description:

Create a column, simultaneously setting some of its attributes.

These attributes deliberately correspond to those usually used with FITS tables.

Invocation:

```
CALL CAT_CNEWS (CI, FNAME, DTYPE, CSIZE, UNITS, EXTFMT, COMM; FI; STATUS)
```

Arguments:

CI = INTEGER (Given)
Catalogue identifier.

FNAME = CHARACTER∗(∗) (Given)
Name of the column (or field).

DTYPE = INTEGER (Given)
Type of the column.

CSIZE = INTEGER (Given)
Size of a CHARACTER column. If the column is not of type CHARACTER CSIZE is irrelevant; it is conventional to set it to zero.

UNITS = CHARACTER∗(∗) (Given)
The units of the column.

EXTFMT = CHARACTER∗(∗) (Given)
The external format for the column.

COMM = CHARACTER∗(∗) (Given)
Comments about the column.

FI = INTEGER (Returned)
Identifier for the column.

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

Bugs:

None known

CAT_CREAT
Create a new catalogue; the name of the catalogue is obtained from an ADAM parameter

Description:

Create a new catalogue; the name of the catalogue is obtained from an ADAM parameter.

Invocation:

```
CALL CAT_CREAT (PCNAME; CI; STATUS)
```

Arguments:

PCNAME = CHARACTER∗(∗) (Given)
Name of the ADAM parameter from which the catalogue name will be obtained.

CI = INTEGER (Returned)
Catalogue identifier.

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

Copyright :

Bugs:

None known

CAT_EGT0F
Get the formatted value of a scalar exprn. field or parameter

Description:

Get the formatted value of a scalar expression, evaluated from the current row buffer, field or parameter. The value is formatted into a CHARACTER string using the appropriate external format specifier.

Remember that parameter and column identifiers correspond to valid expressions.

Invocation:

```
CALL CAT_EGT0F (GI; VALUE, NULFLG; STATUS)
```

Arguments:

GI = INTEGER (Given)
Identifier for either an expression, field or parameter.

VALUE = CHARACTER∗(∗) (Returned)
Value to which the expression evaluates for the current row buffer. If the expression evaluates to null the string returned is '<null>' if VALUE contains six or more characters, otherwise it is '?'.

NULFLG = LOGICAL (Returned)
A flag indicating whether or not the expression evaluates to the null value or not: .TRUE. - The expression is null, .FALSE. - The expression is not null; a genuine value is available.

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

Copyright :

Bugs:

None known

CAT_EGT0<t>
Get the value of a scalar expression, field or parameter

Description:

Get the value of a scalar expression, evaluated from the current row buffer, field or parameter.

Invocation:

```
CALL CAT_EGT0<t> (GI; VALUE, NULFLG; STATUS)
```

Arguments:

GI = INTEGER (Given)
Identifier for either an expression, field or parameter.

VALUE = <type> (Returned)
Value to which the expression evaluates for the current row buffer.

NULFLG = LOGICAL (Returned)
A flag indicating whether or not the expression evaluates to the null value or not: .TRUE. - The expression is null, .FALSE. - The expression is not null; a genuine value is available.

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

Copyright :

Bugs:

None known

CAT_EIDNT
Get an identifier for an expression

Description:

Get an identifier for an expression. A parse of the the expression is attempted, and if successful an identifier is returned. If the parse fails an error status is raised.

Invocation:

```
CALL CAT_EIDNT (CI, EXPR; EI; STATUS)
```

Arguments:

CI = INTEGER (Given)
Identifier to the catalogue to which the expression refers.

EXPR = CHARACTER∗(∗) (Given)
The expression.

EI = INTEGER (Returned)
Identifier to the expression.

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

Copyright :

Bugs:

None known

CAT_EXIST
Attempt to open a catalogue, the name being taken from the ADAM parameter system

Description:

Attempt to open a catalogue, the name being taken from the ADAM parameter system. If the attempt fails then instead of re-prompting the routine returns with an error status. This routine can be used to check the existence of a catalogue.

Invocation:

```
CALL CAT_EXIST (PCNAME, MODE; CI; STATUS)
```

Arguments:

PCNAME = CHARACTER∗(∗) (Given)
Name of the ADAM parameter from which the catalogue name will be obtained.

MODE = CHARACTER∗(∗) (Given)
Mode in which the catalogue will be accessed. One of: WRITE - an new catalogue is to be written.

CI = INTEGER (Returned)
Catalogue identifier. If the specified catalogue is not opened successfully, the null identifier is returned.

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

Copyright :

Bugs:

None known

CAT_FGT0F
Get the formatted value of a scalar expression or field for a given row

Description:

Get the formatted value of a scalar expression or field for a given row. The row may be in either a catalogue, selection or index.

Invocation:

```
CALL CAT_FGT0F (CI, ROWNO, GI; VALUE, NULFLG; STATUS)
```

Arguments:

CI = INTEGER (Given)
Catalogue, selection or index identifier. The row number, ROWNO (below), refers to the row number in the catalogue, selection or index, as appropriate.

ROWNO = INTEGER (Given)
Number of the row to be read.

GI = INTEGER (Given)
Identifier for either an expression, field or parameter.

VALUE = CHARACTER∗(∗) (Returned)
Value to which the expression evaluates for the current row buffer. If the expression evaluates to null the string returned is '<null>' if VALUE contains six or more characters, otherwise it is '?'.

NULFLG = LOGICAL (Returned)
A flag indicating whether or not the expression evaluates to the null value or not: .TRUE. - The expression is null, .FALSE. - The expression is not null; a genuine value is available.

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

Copyright :

Bugs:

None known

CAT_FGT0<t>
Get the value of a scalar expression or field for a given row

Description:

Get the value of a scalar expression or field for a given row. The row may be in either a catalogue, selection or index.

Invocation:

```
CALL CAT_FGT0<t> (CI, ROWNO, GI; VALUE, NULFLG; STATUS)
```

Arguments:

CI = INTEGER (Given)
Catalogue, selection or index identifier. The row number, ROWNO (below), refers to the row number in the catalogue, selection or index, as appropriate.

ROWNO = INTEGER (Given)
Number of the row to be read.

GI = INTEGER (Given)
Identifier for either an expression, field or parameter.

VALUE = <type> (Returned)
Value to which the expression evaluates for the current row buffer.

NULFLG = LOGICAL (Returned)
A flag indicating whether or not the expression evaluates to the null value or not: .TRUE. - The expression is null, .FALSE. - The expression is not null; a genuine value is available.

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

Copyright :

Bugs:

None known

CAT_GETXT
Get the next line of textual information from a catalogue

Description:

Get the next line of textual information from a catalogue.

Invocation:

```
CALL CAT_GETXT (CI; FINISH, CLASS, TEXT; STATUS)
```

Arguments:

CI = INTEGER (Given)
Catalogue identifier.

FINISH = LOGICAL (Returned)
Flag indicating whether a line of text was obtained, coded as follows: .FALSE. - a line was obtained ok; input of of textual information continues. .TRUE. - all the textual information has already been returned; input of textual information has terminated.

CLASS = CHARACTER∗(∗) (Returned)
Class of the textual information. A set of values are permitted for each type of catalogue back-end (or file format), and these sets are different for different back-ends. Note that this argument is returned rather than given; an application cannot prescribe to CAT what class of textual information is required.

TEXT = CHARACTER∗(∗) (Returned)
A single line of textual information.

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

Copyright :

Bugs:

None known

CAT_IINQ
Inquire all the attributes of an index

Description:

Inquire all the attributes of an index.

Invocation:

```
CALL CAT_IINQ (II; CI, FI, ORDER, NUMSEL, COMM, DATE, STATUS)
```

Arguments:

II = INTEGER (Given)
Index identifier.

CI = INTEGER (Returned)
Catalogue identifier for the parent catalogue of the index.

FI = INTEGER (Returned)
Column identifier for the column from which the index was created.

ORDER = INTEGER (Returned)
Order of the index, coded as follows: CAT__ASCND - ascending, CAT__DSCND - descending.

NUMSEL = INTEGER (Returned)
Number of rows in the index.

COMM = CHARACTER∗(∗) (Returned)
Comments for the index.

DATE = DOUBLE PRECISION (Returned)
Date the index was created.

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

Bugs:

None known

CAT_INEW
Create an index on a column

Description:

Create an index on a column.

Currently only temporary indices are created.

Invocation:

```
CALL CAT_INEW (FI, DISP, ORDER; II; STATUS)
```

Arguments:

FI = INTEGER (Given)
Column identifier for the column from which the index will be created. The identifier may refer to either a scalar column or a vector column element.

DISP = CHARACTER∗(∗) (Given)
The disposition of the index. The possibilities are: 'TEMP' - temporary, 'PERM' - permanent. Currently only temporary indices are implemented.

ORDER = INTEGER (Given)
Order of the index, coded as follows: CAT__ASCND - ascending, CAT__DSCND - descending.

II = INTEGER (Returned)
Identifier to the new index.

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

Copyright :

Bugs:

None known

CAT_PINQ
Inquire the values of all the attributes for a parameter

Description:

Inquire the values of all the attributes for a parameter. If the parameter is an array, the value of the first element is returned.

Invocation:

```
CALL CAT_PINQ (QI, SZDIM; CI, QNAME, DTYPE, CSIZE, DIMS, SIZEA, UNITS, EXTFMT, PRFDSP,
COMM, VALUE, DATE; STATUS)
```

Arguments:

QI = INTEGER (Given)
Parameter identifier.

SZDIM = INTEGER (Given)
Maximum permitted dimensionality for a column (defines the size of array SIZEA).

CI = INTEGER (Returned)
Catalogue identifier.

QNAME = CHARACTER∗(∗) (Returned)
Name of the parameter.

DTYPE = INTEGER (Returned)
Type of the parameter. The permitted types are identical to HDS types.

CSIZE = INTEGER (Given)
Size of a character parameter (set to zero if the column is note of data type character).

DIMS = INTEGER (Returned)
Dimensionality of the parameter. For a scalar DIMS = 0.

SIZEA(SZDIM) = INTEGER (Returned)
The size of the array in each of its dimensions. The exception is a scalar parameter, when SIZEA becomes a single-element, one-dimensional array, set to 0.

UNITS = CHARACTER∗(∗) (Returned)
The units of the parameter.

EXTFMT = CHARACTER∗(∗) (Returned)
The external format for the parameter.

PRFDSP = LOGICAL (Returned)
The preferential display flag for the parameter: .TRUE. - display the parameter by default, .FALSE. - do not display the parameter by default.

COMM = CHARACTER∗(∗) (Returned)
Comments about the parameter.

VALUE = CHARACTER∗(∗) (Returned)
The value of the parameter. If the parameter is an array the first element is returned.

DATE = DOUBLE PRECISION (Returned)
Modification date of the parameter.

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

Copyright :

Bugs:

None known

CAT_PNEW0
Create a scalar part (column or parameter)

Description:

Create a scalar part (column or parameter).

Invocation:

```
CALL CAT_PNEW0 (CI, PTYPE, PNAME, DTYPE; PI; STATUS)
```

Arguments:

CI = INTEGER (Given)
Identifier to the catalogue in which the part is to be created.

PTYPE = INTEGER (Given)
Type of part to be created, coded as follows: CAT__FITYP - column (or field), CAT__QITYP - parameter.

PNAME = CHARACTER∗(∗) (Given)
Name of the part to be created.

DTYPE = INTEGER (Given)
Integer code for the data type of the part to be created.

PI = INTEGER (Returned)
Identifier to the new part.

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

Bugs:

None known

CAT_PPTA<t>
Create a parameter, simultaneously setting all its attributes

Description:

Create a parameter, simultaneously setting all its attributes.

Note that the data type attribute can be deduced from the type of the routine. However the CHARACTER size attribute must still be passed.

Invocation:

```
CALL CAT_PPTA<t> (CI, QNAME, CSIZE, DIMS, SIZEA, UNITS, EXTFMT, PRFDSP, COMM, VALUE;
QI; STATUS)
```

Arguments:

CI = INTEGER (Given)
Catalogue identifier.

QNAME = CHARACTER∗(∗) (Given)
Name of the parameter.

CSIZE = INTEGER (Given)
Size of a CHARACTER parameter. If the parameter is not of type CHARACTER CSIZE is irrelevant; it is conventional to set it to zero.

DIMS = INTEGER (Given)
Dimensionality of the parameter. Currently the only permitted value is CAT__SCALR, corresponding to a scalar.

SIZEA(1) = INTEGER (Given)
For vector parameters this attribute would be set to the number of elements in the vector. However, currently only scalar parameters are supported and it should be set to one.

UNITS = CHARACTER∗(∗) (Given)
The units of the parameter.

EXTFMT = CHARACTER∗(∗) (Given)
The external format for the parameter.

PRFDSP = LOGICAL (Given)
The preferential display flag for the parameter: .TRUE. - display the parameter by default, .FALSE. - do not display the parameter by default.

COMM = CHARACTER∗(∗) (Given)
Comments about the parameter.

VALUE = <type> (Given)
Value for the parameter.

QI = INTEGER (Returned)
Identifier for the parameter.

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

Copyright :

Bugs:

None known

CAT_PPTS<t>
Create a parameter, simultaneously setting some of its attributes

Description:

Create a parameter, simultaneously setting some of its attributes.

Note that the attributes set correspond to the ones usually used with FITS tables. Also note that the data type attribute can be deduced from the type of the routine. However for CHARACTER parameters the CHARACTER size attribute should be set using CAT_TATTC.

Invocation:

```
CALL CAT_PPTS<t> (CI, QNAME, VALUE, COMM; QI; STATUS)
```

Arguments:

CI = INTEGER (Given)
Catalogue identifier.

QNAME = CHARACTER∗(∗) (Given)
Name of the parameter.

VALUE = <type> (Given)
Value for the parameter.

COMM = CHARACTER∗(∗) (Given)
Comments about the parameter.

QI = INTEGER (Returned)
Identifier for the parameter.

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

Copyright :

Bugs:

None known

CAT_PUT0<t>
Put a value to a scalar part (field or column)

Description:

Put a value to a scalar part (field or column). If the identifier refers to a field the value is written to the current row buffer.

Invocation:

```
CALL CAT_PUT0<t> (PI, VALUE, NULFLG; STATUS)
```

Arguments:

PI = INTEGER (Given)
Part identifier.

VALUE = <type> (Given)
Value to PUT to the part.

NULFLG = LOGICAL (Given)
A flag indicating whether or not the value is null or not: .TRUE. - The value is null, .FALSE. - The value is not null.

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

Copyright :

Bugs:

None known

CAT_PUTXT
Put a line of textual information to a catalogue

Description:

Put a line of textual information to a catalogue.

Invocation:

```
CALL CAT_PUTXT (CI, CLASS, TEXT; STATUS)
```

Arguments:

CI = INTEGER (Given)
Catalogue identifier.

CLASS = CHARACTER∗(∗) (Returned)
Class of the textual information. The standard classes 'COMMENT' and 'HISTORY' are available for all back-ends. Individual back-ends may support additional classes.

TEXT = CHARACTER∗(∗) (Given)
A single line of textual information.

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

Copyright :

Bugs:

None known

CAT_RAPND
Append the current row buffer to the end of the catalogue

Description:

Append the current row buffer as a new row at the end of the catalogue.

Invocation:

```
CALL CAT_RAPND (CI; STATUS)
```

Arguments:

CI = INTEGER (Given)
Catalogue identifier.

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

Copyright :

Bugs:

None known

CAT_RGET
Read a specified row into the current row buffer

Description:

Read a specified row from a catalogue, selection or index into the current row buffer for that catalogue.

Invocation:

```
CALL CAT_RGET (CI, ROWNO; STATUS)
```

Arguments:

CI = INTEGER (Given)

Catalogue, selection or index identifier. The row number, ROWNO (below), refers to the row number in the catalogue, selection or index, as appropriate.

ROWNO = INTEGER (Given)

Number of the row to be read.

STATUS = INTEGER (Given and Returned)

The global status.

Copyright :

Bugs:

None known

CAT_RSET
Set the number of rows which a new catalogue is expected to contain

Description:

Set the number of rows which a new catalogue is expected to contain.

The value set may be either used or ignored, depending on the type of the catalogue.

Invocation:

```
CALL CAT_RSET (CI, ROWS; STATUS)
```

Arguments:

CI = INTEGER (Given)
Catalogue identifier.

ROWS = INTEGER (Given)
Number of rows which the catalogue is to contain.

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

Copyright :

Bugs:

None known

CAT_RSTXT
Reset the access to the textual information in a catalogue

Description:

Reset the access to the textual information in a catalogue. A subsequent attempt access a line of textual information will return the first line of textual information.

Invocation:

```
CALL CAT_RSTXT (CI; STATUS)
```

Arguments:

CI = INTEGER (Given)
Catalogue identifier.

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

Copyright :

Bugs:

None known

CAT_SELCT
Create a selection of rows satisfying some expression

Description:

Create a selection of rows satisfying some expression.

The selection may be created from either a genuine catalogue or some previous selection from a catalogue.

The expression can involve both indexed and unindexed columns. If indexed columns are involved, they are NOT used to optimise the selection.

Invocation:

```
CALL CAT_SELCT (CI, EI, REJFLG; SI, NUMSEL, SIR, NUMREJ; STATUS)
```

Arguments:

CI = INTEGER (Given)
Input catalogue or selection from which the new selection is to be generated. Note that CI may be either a catalogue or a selection identifier.

EI = INTEGER (Given)
Identifier to the logical expression defining the selection criteria.

REJFLG = LOGICAL (Returned)
Flag indicating whether or not a second selection of the rejected rows is to be created: .TRUE. - create the catalogue of rejected rows, .FALSE. - do not create the catalogue of rejected rows.

SI = INTEGER (Returned)
Selection identifier to the set of selected rows.

NUMSEL = INTEGER (Returned)
Number of rows selected.

SIR = INTEGER (Returned)
Optional selection identifier to the set of rejected rows. If the rejected rows are not being retained then SIR is set to the null identifier.

NUMREJ = INTEGER (Returned)
Number of rows rejected.

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

Copyright :

Bugs:

None known

CAT_SFND<t>
Create a selection of rows within a given range

Description:

Create a selection of rows in a catalogue for which the fields for a specified column lie in a given range.

A field is selected if it lies in the range:

field value .GE. MINRNG .AND. field value .LE. MAXRNG

The selection may be created from either a genuine catalogue or some previous selection from a catalogue.

The specified column must be sorted in ascending or descending order (eventually the routine will be enhanced to work on indexed columns too).

Invocation:

```
CALL CAT_SFND<t> (CI, FI, MINRNG, MAXRNG, REJFLG; SI, NUMSEL, SIR, NUMREJ; STATUS)
```

Arguments:

CI = INTEGER (Given)
Input catalogue or selection from which the new selection is to be generated. Note that CI may be either a catalogue or a selection identifier.

FI = INTEGER (Given)
Identifier to the column whose fields will be selected to lie in the given range. The column must be sorted into ascending or descending order (and known by CAT to be so sorted).

MINRNG = <type> (Given)
Minimum value which a field must satisfy to be selected.

MAXRNG = <type> (Given)
Maximum value which a field must satisfy to be selected.

REJFLG = LOGICAL (Returned)
Flag indicating whether or not a second selection of the rejected rows is to be created: .TRUE. - create the catalogue of rejected rows, .FALSE. - do not create the catalogue of rejected rows.

SI = INTEGER (Returned)
Selection identifier to the set of selected rows.

NUMSEL = INTEGER (Returned)
Number of rows selected.

SIR = INTEGER (Returned)
Optional selection identifier to the set of rejected rows. If the rejected rows are not being retained then SIR is set to the null identifier.

NUMREJ = INTEGER (Returned)
Number of rows rejected.

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

Copyright :

Bugs:

None known

CAT_SINQ
Inquire all the attributes of a selection

Description:
Inquire all the attributes of a selection.

Invocation:

```
CALL CAT_SINQ (SI; CI, EXPR, NUMSEL, COMM, DATE; STATUS)
```

Arguments:

SI = INTEGER (Given)
Selection identifier.

CI = INTEGER (Returned)
Catalogue identifier for the parent catalogue of the selection.

EXPR = CHARACTER∗(∗) (Returned)
Expression defining the selection.

NUMSEL = INTEGER (Returned)
Number of rows in the selection.

COMM = CHARACTER∗(∗) (Returned)
Comments for the selection.

DATE = DOUBLE PRECISION (Returned)
Date the selection was created.

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

Copyright :

Bugs:
None known

CAT_SLIST
Create a selection from an array of row numbers

Description:

Create a selection from an array of row numbers.

The selection may be created from either a genuine catalogue or some previous selection from a catalogue.

Invocation:

```
CALL CAT_SLIST (NUMSEL, SELIST, CRIT, REJFLG, CI; SI, SIR, NUMREJ; STATUS)
```

Arguments:

NUMSEL = INTEGER (Given)
Number of rows to be selected.

SELIST(NUMSEL) = INTEGER (Given)
Array of row numbers to be selected. These row numbers must be rows in the catalogue or selection to which CI corresponds. That is, if CI corresponds to a catalogue they will be absolute row numbers. However, if CI is a selection they will be row numbers in the selection, NOT the corresponding row numbers in the underlying catalogue.

CRIT = CHARACTER∗(∗) (Given)
Character string summarising the selection criteria.

REJFLG = LOGICAL (Given)
Flag indicating whether or not a second selection of the rejected rows is to be created: .TRUE. - create the catalogue of rejected rows, .FALSE. - do not create the catalogue of rejected rows.

CI = INTEGER (Given)
Input catalogue or selection from which the new selection is to be generated. Note that CI may be either a catalogue or a selection identifier.

SI = INTEGER (Returned)
Selection identifier to the set of selected rows.

SIR = INTEGER (Returned)
Optional selection identifier to the set of rejected rows. If the rejected rows are not being retained then SIR is set to the null identifier.

NUMREJ = INTEGER (Returned)
Number of rows rejected.

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

Copyright :

Bugs:

None known

CAT_SRNG<t>
Get the rows corresponding to a range for a sorted column

Description:

Get the first and last rows in a catalogue for which the fields for a specified column lie in a given range.

A field is selected if it lies in the range:

field value .GE. MINRNG .AND. field value .LE. MAXRNG

The selection may be performed on either a genuine catalogue or some previous selection from a catalogue.

The specified column must be sorted in ascending or descending order (eventually the routine will be enhanced to work on indexed columns too).

If there are no rows in the specified range then FIRSTR and LASTR are returned both set zero.

Invocation:

```
CALL CAT_SRNG<t> (CI, FI, MINRNG, MAXRNG; FIRSTR, LASTR; STATUS)
```

Arguments:

CI = INTEGER (Given)
Input catalogue or selection from which the new selection is to be generated. Note that CI may be either a catalogue or a selection identifier.

FI = INTEGER (Given)
Identifier to the column whose fields will be selected to lie in the given range. The column must be sorted into ascending or descending order (and known by CAT to be so sorted).

MINRNG = <type> (Given)
Minimum value which a field must satisfy to be selected.

MAXRNG = <type> (Given)
Maximum value which a field must satisfy to be selected.

FIRSTR = INTEGER (Returned)
First row in the specified range.

LASTR = INTEGER (Returned)
Last row in the specified range.

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

Copyright :

Bugs:

None known

CAT_SZTXT
Return the maximum permitted size of a line of text for a cat

Description:

Return the maximum permitted size of a line of text for a catalogue. The maximum permitted size is defined by the back-end type.

Invocation:

```
CALL CAT_SZTXT (CI, ACCESS; LINESZ; STATUS)
```

Arguments:

CI = INTEGER (Given)
Catalogue identifier.

ACCESS = CHARACTER∗(∗) (Given)
Mode in which the text is to be accessed; 'READ' or 'WRITE'.

LINESZ = INTEGER (Returned)
Maximum permitted size of a line of textual information for the catalogue.

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

Bugs:

None known

CAT_TATT<t>
Set an attribute of a component to a given value

Description:

Set an attribute of a component to a given value. Type conversions are performed if necessary.

Invocation:

```
CALL CAT_TATT<t> (GI, ATTRIB, VALUE<t>; STATUS)
```

Arguments:

GI = INTEGER (Given)
Generic component identifier.

ATTRIB = CHARACTER∗(∗) (Given)
Name of the attribute to be set.

VALUE = <type> (Given)
Value to which the attribute is to be set.

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

Copyright :

Bugs:

None known

CAT_TCOLS
Get the number of columns in a catalogue

Description:

Get the number of columns in a catalogue.

Invocation:

```
CALL CAT_TCOLS (CI, COLFLG; NUMCOL; STATUS)
```

Arguments:

CI = INTEGER (Given)

Catalogue identifier.

COLFLG = INTEGER (Given)

Flag indicating the type of columns of which the number is to be obtained, coded as follows: CAT__GVIRT - virtual columns only, CAT__GPHYS - physical columns only, CAT__GALL - all columns (virtual and physical).

NUMCOL = INTEGER (Returned)

Number of columns in the catalogue.

STATUS = INTEGER (Given and Returned)

The global status.

Copyright :

Bugs:

None known

CAT_TDETL
Get the details of a catalogue

Description:

Get some summary details of a catalogue. This routine will work only if it is given a genuine catalogue identifier. If it is given a selection identifier (or any other sort of identifier) it will return with an error status.

Invocation:

```
CALL CAT_TDETL (CI, COLFLG; NUMROW, NUMCOL, NUMIND, NUMPAR, DATE; STATUS)
```

Arguments:

CI = INTEGER (Given)
Catalogue identifier. This routine will accept only a catalogue identifier; not a selection identifier.

COLFLG = INTEGER (Given)
Flag indicating the type of columns for which the details are to be obtained, coded as follows: CAT__GVIRT - virtual columns only, CAT__GPHYS - physical columns only, CAT__GALL - all columns (virtual and physical).

NUMROW = INTEGER (Returned)
Number of rows in the catalogue.

NUMCOL = INTEGER (Returned)
Number of columns in the catalogue.

NUMIND = INTEGER (Returned)
Number of indices to the catalogue.

NUMPAR = INTEGER (Returned)
Number of parameters in the catalogue.

DATE = DOUBLE PRECISION (Returned)
Creation date of the catalogue.

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

Copyright :

Bugs:

None known

CAT_TIDNT
Get an identifier for a named pre-existing component

Description:

Get an identifier for a named pre-existing component. The component may be of any type.

Invocation:

```
CALL CAT_TIDNT (CI, GNAME; GI; STATUS)
```

Arguments:

CI = INTEGER (Given)
Identifier to the catalogue to which the component belongs.

GNAME = CHARACTER∗(∗) (Given)
Name of the component. The component may be of any type.

GI = INTEGER (Returned)
Identifier to the component. The null identifier is returned if the specified component could not be found.

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

Copyright :

Bugs:

None known

CAT_TIDPR
Determine the parent to a component

Description:

Determine the parent to a component. That is, the routine is give the identifier of a component and it will return the identifier of that component's parent catalogue.

If the routine is given a catalogue identifier it will return the null identifier, because catalogues do not have parents.

Invocation:

```
CALL CAT_TIDPR (GI; CI; STATUS)
```

Arguments:

GI = INTEGER (Given)
Identifier to a component.

CI = INTEGER (Returned)
Identifier to the parent catalogue of the component.

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

Copyright :

Bugs:

None known

CAT_TIDTP
Determine the type of a component

Description:

Determine the type of a component. The possibilities are: catalogue, column (or field), parameter, expression, index, selection or join.

Invocation:

```
CALL CAT_TIDTP (GI; IDTYP; STATUS)
```

Arguments:

GI = INTEGER (Given)

Identifier to a component.

IDTYP = INTEGER (Returned)

Type of the identifier, coded according to the following scheme: CAT__CITYP - catalogue, CAT__FITYP - column (or field), CAT__QITYP - parameter, CAT__EITYP - expression, CAT__IITYP - index, CAT__SITYP - selection, CAT__JITYP - join.

STATUS = INTEGER (Given and Returned)

The global status.

Bugs:

None known

CAT_TIQA<t>
Inquire the value of a single attribute for a component

Description:

Inquire the value of a single attribute for a component. If the value of an array is inquired, the value of the first element is returned.

Invocation:

```
CALL CAT_TIQA<t> (GI, ATTRIB; VALUE<t>; STATUS)
```

Arguments:

GI = INTEGER (Given)
Generic component identifier.

ATTRIB = CHARACTER∗(∗) (Given)
Name of the attribute of the component.

VALUE = <type> (Returned)
Value of the named attribute.

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

Copyright :

Bugs:

None known

CAT_TNDNT
Get an identifier for the Nth (pre-existing) component of a given type

Description:

Get an identifier for the Nth (pre-existing) component of a given type. If N components of the required type could not be found for the specified catalogue then the null identifier (CAT__NOID is returned) and the running status remains ok.

Invocation:

```
CALL CAT_TNDNT (CI, IDTYP, N; GI; STATUS)
```

Arguments:

CI = INTEGER (Given)
Catalogue identifier.

IDTYP = INTEGER (Given)
Type of identifier required, coded as follows: CAT__CITYP - catalogue, CAT__FITYP - column (or field), CAT__QITYP - parameter, CAT__EITYP - expression, CAT__IITYP - index, CAT__SITYP - selection, CAT__JITYP - join.

N = INTEGER (Given)
Number of the component, of the specified type, that is required.

GI = INTEGER (Returned)
Identifier to the required component. If the required component could not be found GI is set to CAT__NOID.

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

Copyright :

Bugs:

None known

CAT_TOPEN
Open a catalogue and obtain an identifier to it

Description:

Open a catalogue and obtain an identifier to it.

The matrix of possibilities for STATE and MODE is:

STATE = 'NEW', then MODE = 'WRITE' is ok and MODE = 'READ' if forbidden.

STATE = 'OLD' then MODE = 'WRITE' is ok and MODE = 'READ' is ok.

Notes:

∗ If a catalogue already exists and an attempt is made to open it with STATE = 'NEW', CAT_TOPEN will fail with an error.

∗ If an existing catalogue is opened with STATE = 'OLD' and MODE = 'WRITE' it will be overwritten.

Invocation:

```
CALL CAT_TOPEN (CNAME, STATE, MODE; CI; STATUS)
```

Arguments:

CNAME = CHARACTER∗(∗) (Given)
Catalogue name.

STATE = CHARACTER∗(∗) (Given)
Required state of the catalogue. One of: NEW - A new catalogue is to be created, OLD - An existing (that is, old) catalogue is to be opened.

MODE = CHARACTER∗(∗) (Given)
Mode in which the catalogue will be accessed. One of: READ - the catalogue may only be read from, WRITE - an new catalogue is to be written.

CI = INTEGER (Returned)
Catalogue identifier.

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

Copyright :

Bugs:

None known

CAT_TRLSE
Release a catalogue identifier

Description:

Release a catalogue identifier.

Internal work space and arrays used by the catalogue are released and made available for re-use.

Note thatthis routine attempts to execute irrespective of the status on entry.

Invocation:

```
CALL CAT_TRLSE (CI; STATUS)
```

Arguments:

CI = INTEGER (Given and Returned)
On input: the catalogue identifier to be released. On output: the null identifier.

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

Copyright :

Bugs:

None known

CAT_TROWS
Get the number of rows in a catalogue, selection or index

Description:

Get the number of rows in a catalogue, selection or index.

Invocation:

```
CALL CAT_TROWS (CI; NUMROW; STATUS)
```

Arguments:

CI = INTEGER (Given)
Catalogue, selection or index identifier.

NUMROW = INTEGER (Returned)
Number of rows in the catalogue, selection or index.

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

Copyright :

Bugs:

None known

CAT_TUNEG
Get a CAT tuning parameter

Description:

Get a CAT tuning parameter. Note that all these parameters are of type CHARACTER.

Invocation:

```
CALL CAT_TUNEG (CATPRM; VALUE; STATUS)
```

Arguments:

CATPRM = CHARACTER∗(∗) (Given)

Name of the catalogue parameter which is to be got. Currently the only parameters supported are 'ANGLE_LIST' and 'QUIET'.

VALUE = CHARACTER∗(∗) (Given)

The current value of the specified parameter, expressed as a CHARACTER string.

STATUS = INTEGER (Given and Returned)

The global status.

Bugs:

None known

CAT_TUNES
Set a CAT tuning parameter

Description:

Set a CAT tuning parameter. Note that all these parameters are of type CHARACTER.

Invocation:

```
CALL CAT_TUNES (CATPRM, VALUE; STATUS)
```

Arguments:

CATPRM = CHARACTER∗(∗) (Given)

Name of the CAT tuning parameter which is to be set. Currently the tuning parameters supported are 'ANGLE_LIST' and 'QUIET'.

VALUE = CHARACTER∗(∗) (Given)

Value required for the catalogue parameter.

For parameter ANGLE_LIST the values may be as follows: 'RADIANS' - output angles in radians, 'SEXAGESIMAL' - output angles in hours or degrees, formatted as sexagesimal values.

For parameter QUIET the values may be as follows: 'YES' - do not issue informational messages (ie. be quiet), 'NO' - issue informational messages (ie. not quiet).

STATUS = INTEGER (Given and Returned)

The global status.

Copyright :

Bugs:

None known

CAT_TYFMT
Construct a character representation of a CAT data type

Description:

Construct a character string representation of a CAT data type and append it to a character string.

CAT data types are represented using an integer code (here argument DTYPE) with a further integer variable (here argument CSIZE) giving the size of character strings. A character representation of the data type is constructed from these integer numbers. This character string representation is (deliberately) identical to that used by HDS.

The constructed value is appended to the input string, starting at element POSN+1 of the input string (using the input value for POSN). On output POSN is set to the new length of the string (cf. the CHR routines).

If an illegal CAT data type is input, the assembled string contains an error text, but an error status is deliberately not raised. This behaviour is adopted because CAT_TYFMT is just formatting a character string and an invalid data type is not really an error for it.

Invocation:

```
CALL CAT_TYFMT (DTYPE, CSIZE; STRING, POSN; STATUS)
```

Arguments:

DTYPE = INTEGER (Given)
Code for a CAT data type.

CSIZE = INTEGER (Given)
Size of a CAT character string.

STRING = CHARACTER∗(∗) (Given and Returned)
The character string into which the data type is to be appended.

POSN = INTEGER (Given and Returned)
The last non-blank element of STRING. VALUE is inserted into STRING starting at the element given by input value of POSN+1. On output POSN is set to the new length of the string, again excluding trailing blanks.

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

Bugs:

None known

B Expression syntax

This appendix discusses the syntax permitted for expressions passed to the expression parser. Expressions are input to CAT as argument EXPR of subroutine CAT_EIDNT (see Section 7).

Expressions have an algebraic format, and comprise: columns, vector column elements, parameters and constants linked by arithmetic operators and mathematical functions. For example, suppose that a catalogue contained scalar columns called x, y and z and parameters called p and q. Some valid expressions are:

```
x
p
x + p                                                    (2)
(x + y + 2) / (p + q)
(2.0*x + y + 3.75*p) + 13.0) / (z + 1.8*q)
```

Remember that in CAT column and parameter names are not case-sensitive. Thus the following column or parameter names would all be considered equivalent:

```
HD_NUMBER
HD_Number
hd_number
```

Vector column elements occur in expressions with their usual syntax: the name of the base column followed by the element number enclosed in square brackets. The first element in a vector is numbered one. For example, an expression to add two to the fourth element of vector FLUX would be 'FLUX[4] + 2.0'.

B.1 Details of expressions

The arithmetic operators are:

- + addition,
- - subtraction,
- * multiplication,
- / division.

brackets ('(' and ')') may be used as required.

B.2 Mathematical functions provided

Table 16 lists the mathematical functions which are provided. The letters denote data types permitted, coded as follows: B = BYTE, H = half INTEGER, I = INTEGER, R = REAL, D =

DOUBLE PRECISION, C = CHARACTER, L = LOGICAL. The appearance of N as an argument means that any numeric type (BHIRD) is permitted, as a result it means that the type is the widest type of any of the arguments. R/D means that the result is REAL unless one or more arguments is of DOUBLE PRECISION type in which case D is the result.

B.3 Rules for expressions

The expression string can contain constants, column and parameter names, operators, functions, and parentheses. In general the usual rules of algebra and Fortran should be followed, with some minor exceptions, as noted below.

(1) Spaces are permitted between items, except that a function-name must be followed immediately by a left parenthesis. Spaces are not permitted within items such as names and numerical constants, but can be used within character strings and date/time values in curly braces.

(2) Lower-case letters are treated everywhere as identical to the corresponding upper-case letter.

(3) Column and parameter names can be up to `CAT__SZCMP` characters long, and may consist of letters, digits, and underscores, except that the first character must not be a digit.

(4) Vector elements are supported but with a restricted syntax: they may consist of a name followed by an unsigned integer constant subscript enclosed in square brackets, for example `FLUX[4]` or `MAGNITUDE[13]`. The first element of the vector is numbered one.

(5) CHARACTER constants may be enclosed in a pair of single or double quotes; embedded quotes of the same type may be denoted by doubling up on the quote character within the string, for example `'DON''T'` or `"DON""T"`.

(6) LOGICAL constants may be `.TRUE.` or `.FALSE.` but abbreviations of these words are allowed down to `.T.` and `.F.`

(7) Numerical constants may appear in any valid form for Fortran 77 (except that embedded spaces are not allowed). Some additional forms are also permitted, as shown below.

(8) %Xstring %Ostring %Bstring for hexadecimal, octal and binary INTEGER constants respectively.

(9) Angles in sexagesimal notation: colons must be used to separate items, for example hours:minutes:seconds (or degrees:minutes:seconds). If there is a leading sign then the value will be taken as degrees:minutes:seconds, otherwise hours:minutes:seconds. In either case the value is converted to RADIANS.

(10) A date/time value may be given as a string enclosed in curly braces; a range of common formats are permitted, with order year-month-day or day-month-year, and the month as a number or three-character abbreviation. The time may follow with colons separating hours:minutes:seconds. Examples of some valid dates:

```
1992-JUL-26 12:34:56
92.7.26
26/7/92T3:45
```

Function	Notes
B = BYTE(N)	convert to BYTE data type
H = HALF(N)	convert to INTEGER*2 data type
I = INT(N)	convert to INTEGER data type
R = REAL(N)	convert to REAL data type
D = DBLE(N)	convert to DOUBLE PRECISION data type
I = NINT(N)	convert to nearest INTEGER
N = MIN(N,N)	the function must have precisely two arguments
N = MAX(N,N)	the function must have precisely two arguments
N = MOD(N,N)	remainder
N = ABS(N)	absolute value
R/D = SQRT(N)	square root
R/D = LOG(N)	natural logarithm
R/D = LOG10(N)	logarithm to the base 10
R/D = EXP(N)	exponential
R/D = SIN(N)	sine; argument in radians
R/D = COS(N)	cosine; argument in radians
R/D = TAN(N)	tangent; argument in radians
R/D = ASIN(N)	arc-sine; result in radians
R/D = ACOS(N)	arc-cosine; result in radians
R/D = ATAN(N)	arc-tangent; result in radians
R/D = ATAN2(N,N)	arc-tangent (two arguments) result in radians
I = IAND(I,I)	bitwise logical AND
I = IOR(I,I)	bitwise logical OR
I = XOR(I,I)	bitwise logical exclusive OR
R/D = DTOR(N)	degrees to radians conversion
R/D = RTOD(N)	radians to degrees conversion
C = UPCASE(C)	convert character string to upper case
C = STRIP(C)	leading and trailing spaces are removed
C = SUBSTR(C,N,N)	returns characters from positions argument 2 to argument 3 inclusive, with the positions starting at 1
L = NULL(*)	.TRUE. if argument is NULL
D = HMSRAD(N,N,N)	converts 3 arguments hours, minutes and

(11) Relational operators are supported in both Fortran 77 form (for example `.GE. .NE.`) as well as in the Fortran 90 forms (for example, `>= /=`).

(12) Single-symbol forms for `.AND. .OR. and .NOT.` are provided as an alternative: `& | #` respectively.

(13) The dots may be left off the Fortran 77 forms of the relational operators and the logical operators `.AND.` and `.OR.` where spaces or parentheses separate them from names or constants, but the logical constants and the `.NOT.` operator need the enclosing dots to distinguish them from other lexical items in all cases.

(14) INTEGER division does not result in truncation (as in Fortran) but produces a floating-point result. The `NINT` or `INT` function should be used (as appropriate) if an INTEGER result is required.

(15) The functions `MAX` and `MIN` must have exactly two arguments.

(16) All arithmetic is carried out internally in DOUBLE PRECISION (but the compiler works out the effective data type of the result using the normal expression rules).

(17) Exponentiation is performed by log/exp functions, with use of ABS to avoid taking logs of negative arguments, thus `-2**3` will come out as '+8' not '-8'.

B.4 Operator precedence

The operator precedence rules are show in Table 17. The rules of Fortran 90 are used as far as possible; in this table the larger numbers denote higher precedence (tighter binding).

Note that all operators except `**` associate from left to right, but `**` and functions associate from right to left.

Precedence	Function/operator
2	start/end of expression
4	`( )`
6	`,`
8	`.EQV. .NEQV.`
10	`.OR. |`
12	`.AND. &`
14	`.NOT. #`
16	`.EQ. .GE. .GT. .LE. .LT. .NE. == >= > <= < /=`
18	`FROM TO`
20	`//`
22	`+ -` (binary operators)
24	`+ -` (unary operators)
26	`* /`
28	`**`
30	all functions

Table 17: Operator precedence rules

C Catalogue formats

CAT can access catalogues held in several different formats: FITS tables, STL lists and Tab-Separated Tables. The restrictions and peculiarities associated with each of these formats are described below.

CAT determines the type of a catalogue from the 'file type' component of the name of the file holding the catalogue. The file types for the various formats are included in the descriptions below. If a file name is specified without a file type then it is assumed to be a FITS table of file type `.FIT`.

C.1 FITS

File types: `.FIT .fit .FITS .fits .GSC .gsc`

Mixed capitalisations, such as `.Fit`, are also supported. The `.GSC` file types are supported in order to allow regions of the HST *Guide Star Catalog* to be accessed.

CAT can read both binary and formatted FITS tables. It can write only binary FITS tables. It should be able to handle most components of FITS tables, with the exception of variable length array columns. If a variable length array column is encountered a warning message will be reported and the column will be ignored.

If a column containing no data is encountered a warning message will be generated and the column will be ignored.

In common with other Starlink software, CAT does not support the COMPLEX REAL and COMPLEX DOUBLE PRECISION data types. If it encounters COMPLEX columns in a FITS table it represents them as follows:

- a COMPLEX REAL scalar column is represented as a REAL vector column of two elements,
- a COMPLEX REAL vector column of n elements is represented as a REAL vector column of $2n$ elements,
- a COMPLEX DOUBLE PRECISION scalar column is represented as a DOUBLE PRECISION vector column of two elements,
- a COMPLEX DOUBLE PRECISION vector column of n elements is represented as a DOUBLE PRECISION vector column of $2n$ elements.

Usually the table component of a FITS file occurs in the first FITS extension to the file. When reading an existing FITS file CAT will look for a table in the first extension. In cases where the table is located in an extension other than the first you may specify the required extension by giving its number inside curly brackets after the name of the file. For example, if the table occurred in the third extension of a FITS file called `perseus.FIT` you would specify:

```
perseus.FIT{3}
```

The closing curly bracket is optional. When CAT writes FITS tables the table is always written to the first extension.

C.1.1 Textual information

The textual information for a FITS table comprises the entire contents of the primary header and the appropriate table extension header of the FITS file containing the table. The entire contents of both headers are returned because this is the best way to present the maximum amount of information about the catalogue to the user in its full context. For example, a FITS table COMMENT keyword may be used to annotate other keywords and if only the COMMENT keywords were returned 'out of context' they would be difficult to understand, and perhaps even misleading.

In addition CAT invents two additional lines of textual information. The first precedes the primary header and serves to introduce it. The second is inserted between the primary header and the table extension header, and serves to introduce the table extension header. These two lines have class 'CAT' (because they are invented by CAT). Table 18 lists the text classes supported for FITS tables.

Access	Class	Description
Read	COMMENT	Comment line
	HISTORY	Line of history information
	KEYWORD	Named FITS keyword
	BLANK	FITS blank comment line
	CAT	Line invented by CAT
Write	COMMENT	Comment line
	HISTORY	Line of history information

Table 18: Classes of textual information for a FITS table

C.2 STL

File types: `.TXT .txt`

Mixed capitalisations, such as `.Txt`, are also supported.

CAT can read and write catalogues in the STL (Small Text List) format. Unlike the other formats which CAT can access the STL format is specific to the CAT library. Nonetheless the STL format exists in order to allow easy access to both private tables and versions of standard catalogues held as text files. It is usually straightforward to create an STL catalogue from a text file containing a private list or standard catalogue.

In the STL format both the table of values for the catalogue and the definitions of its columns, parameters etc. are held in simple ASCII text files. These files may be created and modified with a text editor. The information defining the catalogue is called the **description** of the catalogue and the file in which it is held is called the **description file**.

When you specify a small text list you give the name of the description file. The table of values comprising the catalogue may either be in the same file as the description or in a separate file. If

the table of values occurs in a separate file then the name of this file is specified in the description file and CAT places no restrictions on this name other than those imposed by the host operating system.

The STL format is fully documented in appendices to SUN/190[3], the manual for the catalogue and table manipulation package CURSA[18]. The documentation consists of a simple tutorial which is a good starting point and a complete description. In addition to the basic STL format there is a variant which allows STL format files to inter-operate with applications in the KAPPA image processing package (see SUN/95[3]). The KAPPA variant is also documented in SUN/190.

The CURSA documentation refers to various example files kept in directory:

```
/star/share/cursa
```

As a precaution against the unlikely eventuality that CAT is installed on your system but CURSA is not, there are copies of these files in directory:

```
/star/share/cat
```

The correspondence between items in an STL description and components in a CAT catalogue is obvious. In particular, STL data types convert to CAT ones in the obvious way.

CAT can read STL format catalogues with either a free format or a fixed-format table of values. However, CAT can only write STL format catalogues with a free format table. The KAPPA variant of the STL may be both read and written.

As its name implies, the Small Text List format is intended for use with relatively small catalogues and it is unsuitable for very large catalogues. Currently there is no upper limit to the size of catalogue for which it can be used. However, if you attempt to read a catalogue containing more than 15,000 rows a warning message is issued. A large STL format catalogue may take a while to open for reading and CAT may be unable to access a very large STL catalogue[19].

C.2.1 Textual information

The textual information for an STL catalogue comprises the entire contents of the description. This approach makes the maximum amount of information about the catalogue available to the user in its full context. Table 19 lists the text classes supported for STL catalogues.

C.2.2 Null values

The STL format provides limited support for null values; currently it does not provide all the options described in Section 8.2.

A null value for a field in an STL table is indicated by inserting the string '`<null>`' at the appropriate place in the input file. When CAT reads this string it will interpret it as a null value. Actually, if CAT encounters any value for a field which it cannot interpret given the data type of

[18]CURSA uses the CAT library to access catalogues.

[19]For information, the underlying reason for this behaviour is that CAT attempts to memory-map work arrays to hold the columns of an STL catalogue and then reads the table into these arrays when an input catalogue is opened. For a very large catalogue CAT may be unable to map the required arrays.

Access	Class	Description
Read	COLUMN	Column definition
	PARAMETER	Parameter definition
	DIRECTIVE	Catalogue directive
	CONTINUATION	Continuation line
	NOTE	Annotation of the catalogue description
	COMMENT	Comment line
	BEGINTABLE	Start of table flag
Write	COMMENT	Comment line
	HISTORY	Line of history information

Table 19: Classes of textual information for a STL catalogue

the column (such as a string containing alphabetic characters in a field for an INTEGER column) then the field is interpreted as null. However, when preparing STL files I recommend that you indicate nulls using the string '`<null>`'. This string is recognised as indicating a null value even for CHARACTER columns.

When CAT writes an STL catalogue null fields in the table are represented by the string '`<null>`'.

Null values are not permitted in the KAPPA variant of the STL format.

C.3 TST

File types: `.TAB .tab`

Mixed capitalisations, such as `.Tab`, are also supported.

CAT can read and write catalogues in the TST (Tab-Separated Table) format. The TST format is a standard for exchanging catalogue data and is commonly used to transfer subsets extracted from remote catalogues or archives across the Internet. It is used by GAIA (see SUN/214[5]) and *SkyCat*[20]. The TST format is described in SSN/75[4].

Compared to the other formats supported by CAT, the TST format is somewhat deficient in the amount of metadata that it includes. In particular, the details stored for each column do not include its data type or units. Consequently, when reading a TST catalogue produced by an external program CAT deduces a data type for each column by reading the values that it contains. This procedure usually works reasonably well, though occasionally it produces bizarre results. When CAT writes a TST catalogue it includes some of the column details. These details are written in a format which CAT can interpret if it subsequently reads the catalogue. Though this enhancement is specific to CAT it is entirely consistent with the TST format and does not

[20]http://archive.eso.org/skycat/

affect the ability of external programs to read the catalogues. The format in which the additional information is stored is documented in SSN/75.

The TST format does not support vector columns. If a catalogue containing vector columns is written as a tab-separated table each vector element is written as a scalar column.

Unsurprisingly, given its provenance as a medium for transporting subsets extracted from remote catalogues across the Internet, the tab-separated table format is intended for use with relatively small catalogues and is unsuitable for very large ones. Currently there is no upper limit to the size of catalogue for which it can be used. However, if you attempt to read a catalogue containing more than 15,000 rows a warning message is issued. A large TST format catalogue may take a while to open for reading and CAT may be unable to access a very large TST catalogue[21].

C.3.1 Textual information

The textual information for a tab-separated table comprises the entire description of the table. This approach makes the maximum amount of information about the catalogue available to the user in its full context. Table 20 lists the text classes supported for TST catalogues.

Access	Class	Description
Read	COLUMNS	List of column names
	PARAMETER	Parameter definition
	NOTE	Annotation of the catalogue description
	COMMENT	Comment line
	BEGINTABLE	Start of table flag
Write	COMMENT	Comment line
	HISTORY	Line of history information

Table 20: Classes of textual information for a TST catalogue

C.3.2 Null values

In a tab-separated table the values for adjacent fields in a given row are separated by a tab character. In tab-separated tables written by CAT null values are represented by two adjacent tab characters. That is, no value is included for the null field.

[21]For information, the underlying reason for this behaviour is that CAT attempts to memory-map work arrays to hold the columns of an TST catalogue and then reads the table into these arrays when an input catalogue is opened. For a very large catalogue CAT may be unable to map the required arrays.

References

[1] M.J. Currie and D.S. Berry, 20 October 2000, SUN/95.16: *KAPPA — Kernel Application Package*, Starlink.

[2] A.C. Davenhall, September 1993, *The Starlink Subroutine Interface for Manipulating Catalogues* (StarBase/ACD/3.4). See also A.C. Davenhall, December 1992, *Requirements for a Starlink RDBMS* (StarBase/ACD/2.1). These documents were written as part of the project to define and implement the CAT subroutine library. StarBase/ACD/3.4 gives the complete specification of the CAT subroutine interface, and describes much of the thinking that went into the specification. StarBase/ACD/2.1 lists the original requirements for the Starlink subroutine library to manipulate catalogues and similar tabular datasets. 6.1, 7.1, 14

[3] A.C. Davenhall, 25 July 2000, SUN/190.8: *CURSA — Catalogue and Table Manipulation Applications*, Starlink. 3.4, 4, C.2

[4] A.C. Davenhall, 26 July 2000, SSN/75.1: *Writing Catalogue and Image Servers for GAIA and CURSA*, Starlink. C.3

[5] P.W. Draper and N. Gray, 16 October 2000, SUN/214.8: *GAIA — Graphical Astronomy and Image Analysis Tool*, Starlink. C.3

[6] M.A. Roth, H.F. Korth and A. Silberschatz, 1989, *Acta Informatica* **26**, pp615-642. 8.2.1

[7] J.R. Rumble and F.J. Smith, 1990, *Database Systems in Science and Engineering* (Adam Hilger: Bristol). 1

[8] D.L. Terrett and P.M. Allan, 1 February 1993, SUN/111.2: *SPT – Software Porting Tools*, Starlink. 3.2

[9] P.T. Wallace, 23 March 1992, SGP/16.10: *Starlink Application Programming Standard*, Starlink. 7.3

[10] R.F. Warren-Smith, 28 February 1995, SUN/39.2: *PRIMDAT — Processing of Primitive Numerical Data*, Starlink. 8.2.1

[11] R.F. Warren-Smith and M.D. Lawden, 23 February 1999, SUN/92.11: *HDS — Hierarchical Data System*, Starlink. 6.7